Abstract: Ontologies are a formal explicit specification of a shared conceptualisation. In the E-Government domain they may be used not only for describing and composing services provided by governments and administrations, but also for modelling dependencies between decisions of the different stakeholders (e.g. politicians, public managers and software developers) in order to make services easier for the development and cheaper for the maintenance. In this document we analyse the possibilities to apply ontologies in the E-Government domain. Based on this analysis we define the requirements for an ontology language and we develop several ontologies that are necessary for modelling services in the E-Government domain. Further, since modelling is not sufficient for keeping the consistency of the E-Government services, we define a change management framework based on well-known MAPE management model. Finally, we discuss the role of the ontologies in this model and focus on the need for ontology evolution support.

Keyword List: Ontologies, Web Services, E-Government
# Table of Contents

1 INTRODUCTION ........................................................................................................4

2 PASSPORT ISSUANCE EXAMPLE ........................................................................6
   2.1 PROBLEM DESCRIPTION .............................................................................6
   2.2 DISCUSSION ...................................................................................................8

3 ONTOLOGIES ..........................................................................................................10
   3.1 ONTOLOGY DEFINITION .............................................................................10
      3.1.1 ONTOLOGY LANGUAGES ......................................................................11
      3.1.2 KAON ....................................................................................................12
      3.1.3 OWL .....................................................................................................13
   3.2 REQUIREMENTS .............................................................................................15
      3.2.1.1 MODELLING PRIMITIVES .................................................................16
      3.2.1.2 META-MODELLING .......................................................................17
      3.2.1.3 RULES ..............................................................................................17
      3.2.1.4 MODULARISATION ........................................................................18
      3.2.1.5 LEXICAL LAYER ............................................................................19
      3.2.1.6 EVOLUTION SUPPORT .................................................................19
      3.2.1.7 REASONING ...................................................................................21
      3.2.1.8 COMPLIANCE WITH THE STANDARD .............................................21
      3.2.1.9 SUPPORT FOR WEB SERVICES ....................................................22
      3.2.2 CONCLUSION .......................................................................................23
   3.3 E-GOVERNMENT PROJECTS RELATED TO THE ONTOLOGIES .................23
      3.3.1 SMARTGOV .........................................................................................23
         3.3.1.1 THE SMARTGOV PLATFORM ...................................................24
         3.3.1.2 ONTOLOGIES ..............................................................................24
      3.3.2 ICTE-PAN .............................................................................................26
         3.3.2.1 PA-OMIM (PROCESS MODELLING METHODOLOGY) ....................27
         3.3.2.2 ONTOLOGY ..................................................................................28
      3.3.3 E-POWER ...............................................................................................29
         3.3.3.1 ONTOLOGIES ..............................................................................31
         3.3.3.2 METALEX ....................................................................................32
      3.3.4 EG0V .....................................................................................................33
         3.3.4.1 THE EG0V TECHNICAL ARCHITECTURE ..................................33
         3.3.4.2 SEMANTICS ADDED TO WS IN EG0V .......................................34
         3.3.4.3 LIFECYCLE OF A GOVML DOCUMENT ACCORDING TO EG0V PROJECT .. 34
      3.3.5 CONCLUSION .......................................................................................36

4 WEB SERVICES .....................................................................................................37
   4.1 DEFINITION ....................................................................................................37
   4.2 SEMANTIC WEB SERVICES ..........................................................................38
   4.3 E-GOVERNMENT METADATA STANDARDS AND INITIATIVES .............39
      4.3.1 INTRODUCTION ...................................................................................39
      4.3.2 DEFINITIONS .......................................................................................40
      4.3.3 DUBLIN CORE ....................................................................................40
      4.3.4 NATIONAL METADATA STANDARDS .............................................41
      4.3.5 TOWARDS A UNIVERSAL E-GOVERNMENT METADATA STANDARD ...... 41
         4.3.5.1 GOVML – BASED ON DUBLINCORE METADATA ....................42
         4.3.5.2 A DUBLIN CORE E-GOVERNMENT PROFILE AND MODEL .........48
      4.3.6 CONCLUSIONS AND FUTURE WORK ..............................................50
   4.4 CURRENT LANGUAGES FOR SERVICE DESCRIPTIONS .........................51
5 BUSINESS PROCESS MODELLING ......................................................... 65
  5.1 BUSINESS PROCESS DEFINITION ..................................................... 65
  5.2 BUSINESS PROCESS MODELLING LANGUAGES ................................. 67
    5.2.1 BPML ............................................................................................. 68
    5.2.2 IDEF TECHNIQUES (IDEF0, IDEF3) ................................................... 69
  5.3 SERVICE ORCHESTRATION LANGUAGES AND PROTOCOLS .............. 70
    5.3.1 BPEL4WS ........................................................................................ 70
    5.3.1.1 PASSPORT ISSUANCE EXAMPLE IN BPEL4WS................................. 74
    5.3.2 WSCI ................................................................................................. 78
    5.3.3 ebXML .............................................................................................. 79
    5.3.4 ROSETTANET ..................................................................................... 79
    5.3.5 COMPARISON .................................................................................... 80
  5.4 TOOLS FOR THE BUSINESS PROCESS MODELLING ............................ 81
    5.4.1 ARIS ................................................................................................. 82
    5.4.2 ADONIS ............................................................................................. 84
  5.5 SPECIFIC REQUIREMENTS FOR E-GOVERNMENT PROCESS MODELLING ..... 90
  6 ONTOGOV LIFECYCLE FRAMEWORK ................................................. 92
    6.1 INTRODUCTION .................................................................................... 92
    6.2 MOTIVATING EXAMPLE ....................................................................... 93
    6.3 OUR APPROACH .................................................................................. 96
      6.3.1 EVOLUTION OF THE SEMANTIC WEB SERVICE ONTOLOGY .......... 97
        6.3.1.1 ONTOLOGIES USED FOR MODELLING SEMANTIC WEB SERVICES ...... 98
        6.3.1.2 CHANGES ................................................................................. 111
        6.3.1.3 CONSISTENCY ....................................................................... 113
      6.3.2 PROPAGATING CHANGES FROM BUSINESS RULES TO SERVICES ..... 115
        6.3.2.1 EVOLUTION ONTOLOGY ......................................................... 117
        6.3.2.2 LIFECYCLE ONTOLOGY ......................................................... 121
      6.3.3 PROPAGATING CHANGES WITHIN SERVICES ................................. 124
    6.4 THE CONCEPTUAL ARCHITECTURE .................................................. 126
      6.4.1 THE NEED FOR ONTOLOGY EVOLUTION .................................... 126
      6.4.2 THE MAPE MODEL ..................................................................... 128
    6.5 RELATED WORK ................................................................................. 131
    6.6 CONCLUSION ..................................................................................... 132
  7 REFERENCES ....................................................................................... 134
  8 APPENDIX ......................................................................................... 139
    8.1 SIPOC MODEL TEMPLATES ................................................................. 139
      8.1.1 PROCESS TEMPLATE ................................................................. 139
      8.1.2 ACTIVITY TEMPLATE ............................................................... 142
      8.1.3 PROCESS MAP SYMBOLS ......................................................... 145
        8.1.3.1 PROCESS ............................................................................... 145
        8.1.3.2 RESOURCES ...................................................................... 145
        8.1.3.3 DOCUMENTS ..................................................................... 145
        8.1.3.4 CONTROL STRUCTURE ...................................................... 146
    8.2 ONTOLOGIES FOR THE MODELLING OF THE E-GOVERNMENT SERVICES ...... 148
      8.2.1 PROCESS ONTOLOGY ............................................................... 148
      8.2.2 SERVICE ONTOLOGY FOR THE EXAMPLE SHOWN IN SECTION 6.3.1.1.4 .... 156
1 Introduction

Large, long living application systems in general are not developed to remain stable, but are subject to a continual change. Changes to a system can occur for a number of reasons:

- The environment in which systems operate can change unpredictably, thereby invalidating the assumptions that were made when the system was built. For example, acquiring a new subsidiary in an enterprise adds new business areas as well as functionalities to the existing system;

- Users’ requirements often change after the system is already built, warranting system’s adaptation. For example, hiring new employees may lead to new competencies and greater diversity within an organisation, which need to be reflected in the system;

- Some changes in the domain are implicit and can be discovered only through the analysis of users’ interactions with the system. For example, if many users are interested in two topics in conjunction (e.g. passport and children), and there is no resource matching this criterion, then an efficient system should signal that a resource about the combination of these topics is needed (e.g. a document explaining how a child can get a passport).

Thus, continuous modification is necessary. E-Government domain is not an exception. The legal environment in which an E-Government system works is not immune against changes. For example, ongoing technological development in the field of digital processing has made it much easier for criminals to forge documents. Therefore, the new security feature on identity cards and passports is needed. Further, public managers might want to reengineer the procedure for passport issuance by trying to reuse procedures that already have validated their quality. For example, the existing procedure for the birthday certificate can be included into the procedure for the passport issuance as a whole instead of realizing a new software module for the same functionality. Finally, the system should be improved continually and developed incrementally. For example, the initial procedure for the passport issuance that covers a typical case should be specialized into procedures for children, foreigners, etc.

Ad hoc management of the changes in application systems may work in the short-term, but to avoid unnecessary complexity and failures in the long run, the changes have to be interpreted at the conceptual level. The main reason is that systems are usually constructed based on models. A model is a conceptual description of a part of the real world. To improve the speed and to reduce costs of modification, changes have to be reflected on the model.

In ontology-based systems ontologies are used as a conceptual backbone for providing information about resources and for accessing to the resources [3], [28]. Therefore, in ontology-based systems the changes caused by above-mentioned factors should be applied to the ontology.

In the OntoGov project we investigate the possibility to apply ontologies in the E-Government domain in order to achieve self-managing systems, i.e. systems that can continually update themselves (at least to a certain extent automatically) according to the changes in the domain. This is the main difference of the OntoGov project in
comparison to the existing projects related to ontologies in the E-Government domain. Indeed, these projects use ontologies for modelling the front-desk office process. The goal of these projects is to achieve one-stop E-Government by:

(i) providing all relevant information needed to perform task;
(ii) requiring only the minimal set of information from citizens in order to avoid redundancy (e.g. one has to fill in similar forms with the same data several times when he/she changes the residence).

On the other hand, the OntoGov project aims at improving back-office processes by taking into account the whole process lifecycle. The goal of the project is to develop a framework (and a system based on it) that allows for change propagation and traceability, contributing in this way to the bridging of decision making with technical realisation. For example, when a passport issuance service has to be changed due to changes in national and European legalisation, the OntoGov system will enable to perform a change while keeping the consistency of the whole system.

To achieve these goals, in this document we investigate the possibilities to use ontologies to model the business logic as well as the dependencies between different stakeholders that define this business logic in a collaborative way. Moreover, we consider the specificities that the E-Government domain puts on these issues.

This document is organized as follows: A motivating example is given in section 2. In section 3 we define ontologies, discuss ontology languages and their applications in the E-Government domain. The section 4 is dedicated to web services and their role in the E-Government domain. An analysis of the experiences of using business process modelling techniques in the E-Government domain is given in section 5. The conceptual architecture of a self-managing E-Government system is discussed in section 6.

Since the whole deliverable covers a rather broad area of techniques, tools, methods and approaches, there is no overall section about related work and state of the art, but the respective remarks are made in the specifics sections where necessary and useful. The structure of this deliverable is illustrated in Figure 1.

![Figure 1. Structure of the deliverable](image-url)
2 Passport issuance example

One of the goals of this document is to specify what is needed for achieving consistency in the E-Government domain. These requirements will be motivated by a potential use case (e.g. the passport issuance) and general design objectives that take into account the difficulties in applying ontologies to the unique E-Government environment.

In this section we describe a typical E-Government service, i.e. passport issuance. We try to systematize this description by interpreting this service in the terminology of the SIPOC\(^1\) model [57]. Then from this description we elaborate on which modelling primitives are necessary to represent this service.

2.1 Problem description

A passport is a document, which is required for any individual who wishes to go abroad. An individual cannot travel abroad without the valid passport. It is a proof of an individual’s nationality, name, residential address, age, qualifications, number of times he has travelled abroad etc. Therefore, utmost care is taken by the concerned authorities in preparation of the passport.

The procedure for issue of a passport typically consists of several steps:

1. Application - Application has to be submitted to the passport office in the prescribed form. The form is available at any passport office. All the particulars in the form has to be filled correctly and legibly;
2. Verification of Application Form - The passport authority sends information about the application for issuance of passport to the nearest police station of the residence of applicant. The Police Officer may visit the residence of the applicant to confirm whether he/she stays at the address given in the passport application form. On verification, the police officer submits his report to the passport authorities;
3. Printing of a passport - On satisfaction of all the details the passport authority prepares the passport;
4. Sending a passport - Passport is delivered to the applicant by registered post at the address provided in the application form;

For the first step, i.e. application it is important that along with the form, several documents are required to be submitted, such as:

---
\(^1\) A SIPOC Model shows the suppliers, inputs, process, outputs and customers for a business process. The SIPOC Model establishes the boundaries of a particular business process by showing its beginning and ending points. It provides a process-driven approach to divide the entire scope of the Six Sigma project into manageable partitions. Processes in a SIPOC Model can be detailed within a workflow model, which provides the most effective method for visualizing the business processes performed in the organization. Easily understood by everyone, they provide a picture of how work is accomplished, who is responsible, and the complexity of the process (complex processes usually have more quality problems than simple ones).
- Copy of any of the document for the proof of **residence** duly certified by the applicant;
- Proof of the **date of birth** - copy of birth certificate or a school leaving certificate duly certified by the applicant;
- Proof of a **payment** – The **fee** has to be paid depending upon the type of passport;
- 6 passport size (35mm x 45 mm) photographs, etc.

From this textual description we derive the simplified process map shown in Figure 2. For the interpretation of the building blocks please refer to the appendix section 8.1.3. This graphical representation illustrates that the passport issuance should be considered as a composite service (process), since it has to be realized through the sequence of atomic or composite services (processes). An atomic service (process) is a service that cannot be decomposed into simpler services. It is directly invocable and is executed in single step.

![Figure 2. The process map of the passport issuance service](image-url)

The previously described procedure is a general procedure for a passport issuance. However, there are several variants, which require the modification in the application form as well as in the amount of how much that has to be paid. The following variants can be found:

- Passport for a minor child;
- Procedure for issue of a duplicate passport;
- Re-Issue of the Passport in case of Expiry;
- Procedure for Change of Name.

These variants require the specialisation of the generic application service shown in 8.1.3 into several more specialized services. This is shown in Figure 3. Note that each of these sub-services can be further decomposed. The granularity level depends on the goal of the system. For example, if the goal is to increase the reusability (and
consequently to decrease the effort required for the development and maintenance), then the atomic services (i.e. services that can not be decomposed into simpler services) should be as simple as possible.

![Diagram: Decomposition of the Application service]

**Figure 3. Decomposition of the Application service**

### 2.2 Discussion

We analyse process maps for the passport issuance and other typical E-Government services. Our goal was to discover which modelling constructs are necessary for modelling services in the E-Government domain. The following conclusions are derived.

Each service has at least one input. In most of cases inputs are structured as documents that need to be processed (e.g. application forms) or are required for processing (e.g. a document representing a birthday certificate is needed for getting a passport). Output of a service is either a document that is result of this service or some kind of notification.

Many services require additional information during their execution. Note that the requestor does not supply it. Rather they are internal for the procedure. For example, the existence of criminal proceedings must be checked during the passport verification procedure.

Regarding automation, there are two types of services: manual and automatic. Manual services require one or more persons with specific knowledge and skills to perform them. Automatic services do not require any human intervention, but rather some hardware and/or software equipment. For example, the verification service is a manual service, since a corresponding officer has to decide whether the applicant can obtain a passport or not. On the contrary, the printing procedure is automatic one, since there is no need for a human involvement.

Most E-Government services are composite services. Therefore, the possibility to establish cause-consequence dependencies between services (i.e. sequences) is essential. Further, these sequences are used to connect processes (activities) in an ordered list. However, they might contain conditional constructs (such as IF-THEN). For example, the next step after the verification service depends on the output of this service. Only on satisfaction of all the details the passport authority prepares the
passport. Thereupon it is delivered to the applicant by registered post at the address provided in the application form. However, if the passport authority refused to issue passport, the next step would be the notification about refusal.

One service can be specialized in a several ways. Therefore, a modelling language has to allow such specialisation. For example, the application procedure for the passport issuance depends on the age of an applicant, whether he/she already has a passport, etc. Consequently, a modelling language has to consider composite services in the same way as atomic services.

Although E-Government services are developed mostly for the end-users, they can be useful for the E-Government staff as well. For example, managers might control the efficiency of employees; it might be possible to discover skills needed to speed up some services, etc. It requires the definition of services at different abstraction levels, where deeper layers provide more details about realisation. These abstraction levels are important in order to avoid presenting managers unnecessary information.

Finally, the service granularity level is very important. For example, the application service can be considered as a generic application service where attached documents define which fields have to be shown to an applicant. This generic application services can be used for the birthday application, marriage application, car application, etc. On the other side, one can develop the application service only for the passport issuance, where all logic that is specific for this service is built in code. Higher granularity level speeds up the development of services and makes maintenance easier. However, it requires more complex logic to execute the complex service description. The lower granularity level improves the run-time performance of a system while reducing reusability.
3 Ontologies

In this section we first define the notion of ontologies. Then we discuss the existing ontology languages focusing on modelling primitives needed to represent knowledge specific for the E-Government domain. Finally, we explore the role of ontologies in E-Government domain by analysing existing ontology-based E-Government projects.

3.1 Ontology definition

The term ontology is borrowed from philosophy, where an ontology is a systematic account of existence. For computer science, what "exists" is that which can be represented. Thus, in the context of computer science, the following definition is adopted [1]:

**Definition 1** An ontology is a formal, explicit specification of a shared conceptualisation of a domain of interest.

*Conceptualisation* is an abstract, simplified view of the world that we wish to represent for some purpose. Ontologies have set out to overcome the problem of implicit and hidden knowledge by making the conceptualisation of a domain explicit. Ontology is used to make assumptions about the meaning of a specific concept. It can also be seen as an explication of the context for which the concept is normally used.

Moreover, everything (i.e., any knowledge-based system or any knowledge-level agent) is liable to some conceptualisation, explicitly or implicitly. Therefore, since there is consensus of terms, it is a shared conceptualisation.

Next, the purpose of an ontology is not to model the whole world, but rather a part of it - so-called domain. A *domain* is just a specific subject area or area of knowledge, like medicine, tool manufacturing, real estate, automobile repair, financial management, etc. Therefore, in order to define a domain, it is important to know what an ontology is for.

Further, ontologies serve as a means for establishing a conceptually concise basis for communicating knowledge for many purposes. In order to achieve this, an ontology has to be a *formal* description of the meaning of concepts and relationships between them. Therefore, the formal specification means that an ontology is specified by means of a formal language, e.g. first order logic.

Finally, this formal model is readable, understandable and processable not only for the people, but also for the machines. This is achieved through the *explicit* specification, while there is formal semantics of all statements, i.e. the semantics of the used language is formally specified as well. Therefore, ontologies have to be specified in a language that comes with formal semantics. Only in this way can the detailed, accurate, consistent, sound, and meaningful description be made.

The study of ontologies and their use is no longer just one of the fields in the computer science literature. Ontologies are now ubiquitous in many enterprise-wide information-systems: they are used in e-commerce, knowledge management and in
various application fields such as bioinformatics and medicine. Moreover, they constitute the backbone for the Semantic Web, which is discussed in the next section.

3.1.1 Ontology Languages

To be useful, ontologies must be expressed in a concrete notation. An ontology language is a formal language by which an ontology is built. There have been a number of languages for ontologies both proprietary and standards-based. Based on their formal semantics they can be split into two groups of languages [58]:

- **Frame-based ontology languages** – They have a long history in artificial intelligence. Their central modelling primitives are classes (known as frames) with properties (known as slots). A frame provides a context for modelling a class, which is generally defined as a subclass of one or more other classes, with slot-value pairs being used to specify additional constraints on instances of the new class. Many frame-based systems and languages with many additional refinements of these modelling primitives have been developed. Moreover, adapted to the object-oriented paradigm they have been very successfully applied in the software engineering. For example, the KAON ontology language [12], which is used in this thesis, incorporates the essential modelling primitives of frame-based systems, being based on the notion of a concept and the definition of its superclasses and slots. It also treats slots as first class objects that can have their own properties (e.g. domain and range) and can be arranged in a hierarchy;

- **Description logic based ontology languages** – They have been developed in knowledge-representation research, and describe knowledge in terms of concepts (comparable to classes, or frames) and roles (comparable to slots in frame systems). An important aspect of these languages is that they have very well understood theoretical properties. Description logic enables reasoning with concept descriptions and the automatic derivation of classification taxonomies. There are now efficient implementations of description logic reasoners able to perform these tasks. For example, the Ontology Web Language - OWL inherits from description logic both their formal semantics and efficient reasoning support.

In the rest of this section we first discuss the KAON ontology language as an example of the frame-based languages. Secondly, we elaborate on the OWL ontology language that is going to be a standard language for representing knowledge on the Web. Finally, we compare these two languages based on the set of the criteria arisen from the E-Government requirements.

---

2 DAML + OIL - http://www.daml.org/2001/03/reference.html;
KAON - http://kaon.semanticweb.org
OIL - http://www.ontoknowledge.org/oil/
OWL - http://www.w3.org/2001/sw/WebOnt/
3.1.2 KAON

The KAON ontology language is based on RDF(S), but comes with a clean separation of the modelling primitives from the ontology itself. KAON provides means for modelling metaclasses and incorporating several commonly used modelling primitives, such as transitive, symmetric and inverse properties, as well as cardinalities. Figure 3 shows a very simple KAON ontology.

An ontology in the KAON language consists of concepts (sets of elements) and properties (specification how objects may be connected). For example, the ontology shown in Figure 4 contains, among others, the concepts “Person”, “Department”, “Product”, etc. and the properties “worksIn”, “produces”, etc. Each property must have at least one domain concept (e.g. the domain concept for the property “worksIn” is the concept “Person”). Its range may either be a literal (c.f. the “hasName” property), or a set of at least one concept (e.g. the range concept for the property “worksIn” is the concept “Department”). Domain and range concept restrictions are treated conjunctively. Consequently, all of them must be fulfilled for each property instantiation. Some properties may be marked as symmetric (c.f. the “cooperatesWith” property) or the transitive (c.f. the “hasPart” property). Further, it is possible to say that two properties are inverse of each other (c.f. properties “owns”, “isOwnedBy”).

![Ontology Example in the KAON Ontology Language](image)

Figure 4. An ontology example in the KAON ontology language

For each concept-property pair it is possible to specify the minimum and maximum cardinalities, defining how many times a property may be specified for instances of that class. Concepts and properties can be arranged in a hierarchy. For example, the concepts “Software” and “Hardware” are subconcepts of the concept “Product”. The hierarchy relation relates directly connected concepts (properties) and it is defined as a transitive relationship.

Each ontology has an instance pool associated with it. An instance pool is constructed by specifying instances of different concepts and by establishing property instan-
tations between instances. Property instantiations must follow the domain and range constraints, and must obey the cardinality constraints, as specified by the property specifications.

Moreover, an ontology contains so-called lexical entries that reflect various lexical properties of ontology entities, such as a label, synonym, stem or textual documentation. There is an m:n relationship between lexical entries and ontology entities. Thus, the same lexical entry may be associated with several elements (e.g. the label “APPLE” may be associated with an instance representing an apple fruit or an apple computer).

All information is organized in so-called OI-models (ontology-instance models), containing both ontology entities (concepts and properties) and their instances. This allows grouping of concepts with their well-known instances into self-contained units. For example, the ontology shown in Figure 4 that models IBM products contains the concept “Software” along with its well-known instances, e.g. “DB2”, “Lotus Notes”, “Web Sphere”, etc. An OI-model may include another OI-model, thus making all definitions from the included OI-model automatically available.

3.1.3 OWL

The Web Ontology Language OWL is a semantic markup language for publishing and sharing ontologies on the World Wide Web. OWL is developed as a vocabulary extension of RDF\(^3\) (the Resource Description Framework) and is derived from the DAML+OIL\(^4\) Web Ontology Language. The goals of the OWL language are (i) formalization of a domain by defining classes and properties of those classes, (ii) definition of individuals and assertion of properties about them, and (iii) reasoning about these classes and individuals to the degree permitted by the formal semantics of the OWL language.

OWL provides three increasingly expressive sublanguages designed for use by specific communities of implementers and users:

- **OWL Lite** supports those users primarily needing a classification hierarchy and simple constraints. For example, while it supports cardinality constraints, it only permits cardinality values of 0 or 1;

- **OWL DL** supports those users who want the maximum expressiveness while retaining computational completeness (all conclusions are guaranteed to be computable) and decidability (all computations will finish in finite time). OWL DL includes all OWL language constructs, but they can be used only under certain restrictions (for example, while a class may be a subclass of many classes, a class cannot be an instance of another class);

- **OWL Full** is meant for users who want maximum expressiveness and the syntactic freedom of RDF with no computational guarantees. For example, in OWL Full a class can be treated simultaneously as a collection of individuals and as an individual in its own right. OWL Full allows an ontology to augment the meaning of the pre-defined (RDF or OWL) vocabulary. It is unlikely that

---

\(^3\) [http://www.w3.org/RDF/](http://www.w3.org/RDF/)

\(^4\) [http://www.daml.org/](http://www.daml.org/)
any reasoning software will be able to support complete reasoning for every feature of OWL Full.

Since there is no tool that supports OWL Full, in the rest of this section we discuss modelling primitives provided by OWL Lite / OWL DL.

A class defines a group of individuals that belong together because they share some properties. There are two built-in classes: Thing is the class of all individuals and is a superclass of all OWL classes and Nothing is the class that has no instances and is a subclass of all OWL classes. Class hierarchies may be created by making one or more statements that a class is a subclass of another class.

Moreover, properties can be used to state relationships between individuals or from individuals to data values. Property hierarchies may be created by making one or more statements that a property is a subproperty of one or more other properties. A domain of a property limits the individuals to which the property can be applied. The range of a property limits the individuals that the property may have as its value.

Finally, individuals are instances of classes, and properties may be used to relate one individual to another.

Besides these primitives related to the RDF Schema, OWL Lite includes primitives related to equality or inequality. For example, two classes (properties) may be stated to be equivalent. Equivalent classes have the same instances. Equivalent properties relate one individual to the same set of other individuals. Equality can be used to create synonymous classes (properties). Two individuals may be stated to be the same. These constructs may be used to create a number of different names that refer to the same individual. Moreover, an individual may be stated to be different from other individuals and a number of individuals may be stated to be mutually distinct.

There are special identifiers in OWL Lite that are used to provide information concerning properties and their values. One property may be stated to be the inverse of another property. Properties may be stated to be transitive and/or symmetric. Moreover, properties may be stated to have a unique value. This feature is shorthand for stating that the property's minimum cardinality is zero and its maximum cardinality is 1. Properties may be stated to be inverse functional, which has also been referred to as an unambiguous property.

OWL Lite allows restrictions to be placed on how properties can be used by instances of a class. The restrictions allValuesFrom and someValuesFrom are stated on a property with respect to a class. The first one means that this property on this particular class has a local range restriction associated with it. The second one means that a particular class may have a restriction on a property that at least one value for that property is of a certain type.

OWL Lite includes a limited form of cardinality restrictions (minCardinality and maxCardinality), since it allows statements concerning cardinalities of value 0 or 1.

Further, OWL Lite contains an intersection constructor but limits its usage. OWL uses the RDF mechanisms for data values. It supports notions of ontology inclusion and relationships and attaching information to ontologies.
3.2 Requirements

In this section we define the requirements for an ontology language to be applied in the E-Government domain. Then we discuss the degree to which the KAON and the OWL ontology languages and the tools based on them fulfil these requirements.

One typical E-Government service is the issuance of a passport that is described in section 2. Based on the lessons learned during modelling this service, we define the following set of criteria:

- **Modelling primitives** – An ontology language intends to capture the necessary modelling primitives that both provide adequate expressive power and are well understood thereby allowing the semantics to be precisely specified and complete inference to be viable. In the E-Government domain a high degree of flexibility and expressiveness is required. For example, the ability of an ontology to express constraints (e.g. one can get only one passport) would be an advantage.

- **Meta modelling** – One entity has to be considered as a concept and as an instance at the same time. For example, the concept “Passport” from the domain ontology is used to describe an instance of the “Passport issuance” web service.

- **Rules** – Two types of rules might be distinguished: axioms as a standard axiom schemata (e.g. symmetry, transitivity, inverse) and domain-specific rules. For example, if a person performs services, then a service is performed by a person. Therefore, “performs” and “isPerformedBy” are inverse properties. Further, if a person performs a service that is about some topic, then this person has knowledge about this topic. It can be formalised in a form of IF-THEN rules\(^5\): FORALL Person[hasKnowledge=>Topic] ← Person[performs=>Service] AND Service[isAbout=>Topic].

- **Modularisation** – It is a common engineering practice to extract a well-encapsulated body of information in a separate module that can later be reused in different contexts. However, modularisation of ontologies has some special requirements: both instances and schematic definitions may be subject to modularisation. In E-Government it would be useful to develop a generic ontology describing a service – this is a so-called Meta Ontology (see section 6.3.1.1). This ontology can be further reused and extended for developing any specific service ontology. For example, an ontology describing the “Passport Issuance” web service might include the Meta Ontology as well any ontology describing domain specific terms such a passport.

- **Lexical layer** – the idea of the E-Government initiative of EU is to bridge the gap not only between citizens and the state, but also between different nationalities. Therefore, prerequisite for such communication/interoperability is multi-lingual support. For example, to automate the procedure of residence change\(^6\), it may be necessary to describe terms in different languages. The

\(^5\) This rule is written in the F-Logic notation [64].

\(^6\) One has to sign out an existing residence and to register a new one.
existing place of residence and the new one might be in different countries that have different official languages. The whole procedure can only be performed automatically if a common understanding of used terms is achieved. For example, the service has to recognise that the term “passport” and “der Ausweis” have the same meaning.

- **Evolution support for ontologies** – The overall objective of the OntoGov project is to develop, test and validate an ontology-enabled platform that will facilitate the consistent configuration and re-configuration E-Government services. Since both configuration and re-configuration problems are realized by applying ontology changes either starting from scratch or from existing ontology, the evolution support is mandatory.

- **Efficient reasoning** - Inference mechanisms for deduction of information not explicitly asserted is an important characteristic of ontology-based systems. However, systems with very general inference capabilities often do not take into account other needs, such as scalability and concurrency. A lot of knowledge in the E-Government domain has to be modelled at the instance level. Therefore, an ontology reasoning system has to provide efficient means for reasoning at the instance level.

- **Compliance with standards** – An ontology language has to maximise compatibility with existing W3C standards. The compliance of E-Government services with standards for semantic web services is absolutely necessary, since this is the only realistic way that all of these distributed heterogeneous services can be connected.

- **Support for modelling of Web Services** – the aim of the OntoGov project is to apply semantic web services in the E-Government domain. Therefore, it is important to check whether and with which degree existing ontology languages and their infrastructures provide support for modelling of web services. This model includes description as well as composition of web services.

It is important to note that some of the criteria are related to the ontology language only and some of them to the tools that allow developing ontologies in the language or reasoning on ontologies. Table 1 shows the result of the analysis of KAON and OWL ontology languages/tools with respect to these requirements.

### 3.2.1.1 Modelling Primitives

The OWL ontology language is richer than the KAON ontology language. For example, it allows expressing equality between ontology entities. Thus, it provides more scope for modelling domains (e.g. information integration). The question is whether we need all expressiveness of OWL for modelling the E-Government domain. Practical experiences show that people prefer to use simple solutions that work, rather than advanced solutions, which do not scale [61].

---

7 W3C stands for World Wide Web Consortium (http://www.w3.org/)
3.2.1.2 Meta-modelling

The OWL DL language does not support meta modelling. Every URI that is used as a class name must be explicitly asserted to be of type owl:Class (and similarly for properties), every individual must be asserted to belong to at least one class (even if only owl:Thing), the URI's used for classes, properties and individuals must be mutually disjoint. The choice between OWL DL and OWL Full mainly depends on the extent to which users require the meta-modelling facilities of RDF Schema (e.g. defining classes of classes, or attaching properties to classes). When using OWL Full as compared to OWL DL, reasoning support is less predictable since complete OWL Full implementations do not currently exist.

To overcome that problem, the KAON ontology language introduces so-called spanning objects. A spanning object is defined as a triple that combines different interpretations of an ontology entity. Therefore, KAON supports concept meta modelling, which means that it is possible to treat concepts and properties as instances of meta-concepts. Consequently, a concept and an instance with the same URI may exist simultaneously in the same ontology. If this is the case the instance is called the spanning instance of the concept, and the concept is called the spanning concept of the instance. Precise semantics is associated with such cases [12].

Table 1. The results of the evaluation

<table>
<thead>
<tr>
<th>Modelling Primitives</th>
<th>KAON</th>
<th>OWL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Meta-modelling</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>Rules</td>
<td>≅</td>
<td>+</td>
</tr>
<tr>
<td>Modularisation</td>
<td>+</td>
<td>≅</td>
</tr>
<tr>
<td>Lexical Layer</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>Ontology Evolution Support</td>
<td>+</td>
<td>≅</td>
</tr>
<tr>
<td>Reasoning</td>
<td>≅</td>
<td>+</td>
</tr>
<tr>
<td>Compliance with the standard</td>
<td>≅</td>
<td>+</td>
</tr>
<tr>
<td>Support for Web Services Modelling</td>
<td>-</td>
<td>+</td>
</tr>
</tbody>
</table>

3.2.1.3 Rules

Based on our experience in modelling, we observed that rules in conceptual modelling are important. However, they are often used in some well-defined patterns. Hence, instead of building a system based on a general reasoning paradigm, the lightweight inferences may be sufficient. It is based on axiom patterns [10], which are predefined types of rules that sustain scalability and tractability. The list of axiom patterns in the

\[8\] Description: “-“ means that there is no support, “≅” states that support is partial and “+” corresponds to full support.
KAON language as well as in the OWL-Lite language is currently limited to common patterns in ontology structure, such as symmetric, transitive and inverse relations.

Therefore, regarding rules, KAON provides support only for a standard set of rules, which are called axioms. For example, a property can be defined as a symmetric, or a transitive property or two properties may be inverse to each other. Currently, we are working on the extension of the KAON ontology language with DL-safe rules [62]. In DL-safe rules, concepts and properties are allowed to occur as unary resp. binary predicates in the atoms of the rule head or body, but each variable in the rule is required to occur in an atom in the rule body whose predicate is neither a concept nor a role. The DL-safety requirement makes query answering decidable, since it ensures that rules are applicable to only those individuals explicitly occurring in the knowledge base.

On the other hand, a proposal for a rules extension⁹ to the OWL DL and OWL Lite sub-languages of the OWL Web Ontology Language is on the way. The proposed rules are of the form of an implication between an antecedent (body) and consequent (head). The intended meaning can be read as: whenever the conditions specified in the antecedent hold, then the conditions specified in the consequent must also hold.

### 3.2.1.4 Modularisation

It is inefficient and error-prone to always build the ontology from scratch. Rather, these models should be built by reusing smaller, well-defined components. An ontology can reuse definitions from other ontologies through modularisation.

The KAON ontology language incorporates an explicit mechanism for including ontologies [11]. Inclusion is supported by allowing an ontology to include other ontologies, thus obtaining the union of the definitions from all included models. All definitions from an included ontology are automatically available in the including ontology. Cyclic inclusions are not allowed. Inclusion is performed by-reference. It means that models are virtually merged. However, the information about the origin of each entity is represented explicitly.

Similarly, OWL supports notions of ontology inclusion through the owl:imports primitives. Importing another ontology brings the entire set of assertions provided by that ontology into the current ontology. In order to make best use of this imported ontology it would normally be coordinated with a namespace declaration. Notice the distinction between these two mechanisms. The namespace declarations provide a convenient means to reference names defined in other OWL ontologies. Conceptually, owl:imports is provided to indicate the intention to include the assertions of the target ontology. Importing another ontology, $O_2$, will also import all of the ontologies that $O_2$ imports.

Ontology inclusion allows reusing ontologies available within one node in the system. However, we envisage the Semantic Web where ontologies are spread across many different nodes, so the inclusion mechanisms cannot be used directly. There are two possible solutions how to achieve reuse in this case: dynamic reuse and reuse through replication [11]. A more practical solution to the problem in the WWW context is to replicate distributed ontologies locally and to include them in other ontologies.

---

contrast to KAON that supports reuse in the distributed case, OWL does not provide such support. Whether or not an OWL tool must load an imported ontology depends on the purpose of the tool [59]. If the tool is a complete reasoner (including complete consistency checking) then it must load all of the imported ontologies. Other tools, such as simple editors and incomplete reasoner, may choose to load only some or even none of imported ontologies.

### 3.2.1.5 Lexical Layer

Many applications, such as semantics-driven web content management, extensively depend on lexical information about entities in an ontology, such as labels for ontology entities in different languages. Hence, a consistent way of associating lexical information with ontology entities is mandatory [63].

In the KAON ontology language, lexical information about ontology entities is explicitly stored in the ontology and can be manipulated using the usual constructs. There is KAON_Lexical ontology that defines the formal semantics.

On the other side, OWL does not provide explicit support for lexical information. Equivalence can be used to create synonymous classes/properties. Further, lexical information can be simulated by defining the corresponding properties (such as “isSynonymOf”) and by making a common agreement about their meaning (i.e. accepting that the property “isSynonymOf” should be used between A and B iff A is a synonym of B).

### 3.2.1.6 Evolution Support

Ontology evolution is a process of modifying an ontology while keeping its consistency. The modification is achieved by applying ontology changes. The set of elementary changes heavily depends on the underlying ontology model. A full set of elementary changes can thus be defined by the cross product of the set of entities of the ontology model, which form the meta schema, and the set of meta-changes, which includes addition and removal. Therefore, a set of changes has to contain *AddConcept* and *RemoveConcept* changes. These elementary changes can be further combined into composite changes that represent a group of elementary or composite changes applied together. For example, *MoveConcept* would be a composite change, since it can be realized by the *AddSubConceptOf* change followed by the *RemoveSubConceptOf* change.

An ontology model is reflected in the set of changes. Therefore, since OWL provides more modelling primitives, the set of OWL changes is richer than the set of KAON changes.

The goal of ontology evolution is to ensure that the application of ontology changes produces as an output an ontology conforming to the set of ontology consistency constraints. Therefore, ontology evolution requires the definition of ontology consistency. Ontology consistency can be considered as logical coherence and accordance with the model of the ontology language. It is an agreement among ontology entities with respect to the semantics of the ontology language. Since different ontology languages have different semantics, their consistency constraints are different. For example, whereas the OWL ontology language allows having cycles
in a concept hierarchy, the KAON ontology language prevents cycles in a concept hierarchy.

Here we discuss the most important differences in the semantics of the KAON and OWL ontology languages. Firstly, the definition of domains and ranges in the KAON ontology language differs from that of RDFS and OWL. In these languages, domain and range specifications are axioms specifying sufficient conditions for an instance to be a member of some class. For example, although for an instance “instanceI” is not explicitly stated to be an instance of a concept “conceptC”, because it has a property “propertyP” instantiated and because “propertyP” has the concept “conceptC” as domain, it can be inferred that the instance “instanceA” is an instance of the concept “conceptC”.

From our experience, while sometimes such inferencing may indeed be useful, often it is not needed, or even desired in closed environments, such as e.g. presented by most E-Government applications. Most users without a formal background in logic, but with strong background in databases and object-oriented systems, intuitively expect domains and ranges to specify the constraints that must be fulfilled while populating ontology instances. In other words, unless “instanceI” is known to be an instance of the concept “conceptC”, then the “propertyP” cannot be instantiated for the instance “instanceI” in the first place, or the ontology becomes inconsistent.

This approach has the following benefits:

- Treating domains and ranges as constraints makes it possible to guide the user in the process of providing information about instances. It is easy to compute the set of properties that can be applied to an instance, and then to ask the user to provide values for them. On the other hand, if domains and ranges are treated as axioms, any property can be applied to any instance, which makes it difficult to constrain the user's input;

- Similar problems occur when evolving an ontology. E.g., if an instance is removed from the extension of a concept, it can be computed that all properties whose domain or range is that concept must be removed as well. On the other hand, if domains and ranges are axioms, then it is not clear how to change an ontology so that it still makes sense;

- Treating domains and ranges as axioms introduces significant performance overhead in query answering. For example, to compute the extension of some concept, it is not sufficient to take the union of the extension of all subconcepts. One must examine a larger part of the instance pool to see which instances may be classified under the concept according to the domain and range axioms. Therefore, if only the constraint semantics is needed, the system will suffer from unnecessary performance overhead.

Further, in the KAON ontology modelling approach cardinalities are treated as constraints regulating the number of property instances that may be specified for instances of each concept. This is different from OWL and other description logic languages, where cardinalities are axioms specifying that instances with particular number of property instances to some concept can be inferred to be instances of some concept. However, constraining the number of property instances that are allowed for some instance is extremely useful for guiding the user in providing ontology
instances. By knowing how many property instances can be provided for instances of some concept, the user can be asked to provide the appropriate number of values. Similar arguments as in the case of domain and range semantics apply here as well.

Finally, it is important mentioning that one of the most important advantages of the KAON ontologies is the evolution support provided by the KAON framework. It covers evolution of a single ontology as well as evolution between multiple ontologies. These ontologies may be within the same node on the Web or may be physically distributed. Up to now, there is still no evolution support for OWL ontologies.

3.2.1.7 Reasoning

The KAON ontology language tries (i) to follow as closely as possible the object-oriented modelling paradigm and (ii) to extend it with simple deductive features by keeping in mind some practical aspects. On the other hand, OWL Lite/DL allows inferencing, with decidable and complete inference procedures. However, it lacks of the previously mentioned practical considerations. This relates to the support for meta modelling, the interpretation of property domains and range, and cardinalities.

Moreover, there is a practical problem related to the choice of OWL. Namely, at the time this document was written, the tool support for OWL was quite limited. Existing tools can be separated into two classes:

- Inference engines for description logics, of which the most prominent example is RACER\textsuperscript{10}. Such tools usually provide a sound and complete inference procedure for (most of the features of) OWL. However, the performance of such tools is usually quite limited, in particular in case of ontologies containing a large number of instances;
- Tools evolved out of the RDFS initiative, of which examples are Jena and Protégé. Such tools usually provide reasonable performance. However, they often support a very limited subset of OWL, and the support is not at the maturity level that is desired.

However, an alternative to OWL might be the KAON language implemented in KAON – an ontology management system developed at FZI and AIFB at the University of Karlsruhe. The reasoning in the KAON system is based primarily on deductive database technology, which has proven to be indispensable for achieving inferencing tractability and practicability. On the other hand, OWL requires the description logic inferencing. Handling description logic typically requires algorithms such as tableau reasoning that do not integrate easily with the existing database infrastructure and are often intractable in practice. Most of them are limited to the T-Box reasoning, which means that ontology instances (i.e. A-Box) cannot be taken into account.

3.2.1.8 Compliance with the standard

OWL has been designed to meet this need for a Web Ontology Language. OWL is part of the growing stack of W3C recommendations related to the Semantic Web:

\footnote{http://www.sts.tu-harburg.de/~r.f.moeller/racer/}
• XML provides a surface syntax for structured documents, but imposes no semantic constraints on the meaning of these documents;
• XML Schema is a language for restricting the structure of XML documents;
• Resource Description Framework (RDF)\(^\text{11}\) is a datamodel for objects ("resources") and relations between them, provides a simple semantics for this datamodel, and these datamodels can be represented in an XML syntax;
• RDF Schema (RDFS)\(^\text{12}\) is a vocabulary for describing properties and classes of RDF resources, with semantics for generalization-hierarchies of such properties and classes;
• OWL adds more vocabulary for describing properties and classes: among others, relations between classes (e.g. disjointness), cardinality (e.g. "exactly one"), equality, richer typing of properties, characteristics of properties (e.g. symmetry), and enumerated classes.

When KAON\(^\text{13}\) development started, RDF and RDFS were the de-facto standard languages for ontology modelling in the Semantic Web [9]. Hence, these languages were chosen to be implemented by the platform. However, as development progressed, certain features of these languages were found to be inadequate for practice. Also, the languages in question have undergone a transformation themselves. Hence, the currently implemented KAON ontology language is based on RDF(S), but contains many additions and changes to the standard [12].

### 3.2.1.9 Support for Web Services

The Semantic Web Services arm of the DAML program\(^\text{14}\) is developing an OWL-based Web Service Ontology, **OWL-S** (formerly DAML-S), as well as supporting tools and agent technology to enable automation of services on the Semantic Web.

OWL-S supplies Web service providers with a core set of markup language constructs for describing the properties and capabilities of their Web services in unambiguous, computer-interpretable form. OWL-S markup of Web services will facilitate the automation of Web service tasks including automated Web service discovery, execution, interoperation, composition and execution monitoring. Following the layered approach to markup language development, the current version of OWL-S builds on top of OWL.

KAON is used in many projects related to the semantic web services (e.g. SWWS, DIP, etc.). Therefore, the support for semantic web services can be expected very soon.

\(^{11}\) [http://www.w3.org/TR/REC-rdf-syntax/](http://www.w3.org/TR/REC-rdf-syntax/)
\(^{12}\) [http://www.w3.org/TR/rdf-schema/](http://www.w3.org/TR/rdf-schema/)
\(^{13}\) [http://kaon.semanticweb.org](http://kaon.semanticweb.org)
\(^{14}\) [http://www.daml.org/services/](http://www.daml.org/services/)
3.2.2 Conclusion

This analysis shows that each language has its own advantages and disadvantages. The choice of the most suitable language primarily depends on the problem that has to be resolved, i.e. the complexity of services that have to be automated.

We are developing the new version of the KAON ontology management system, the so-called KAON2, which provides support for OWL DL ontologies without nominals. Moreover, we plan to extend the OWL DL with the decidable form of the meta-modelling. Further, we will investigate the possibilities to combine a certain type of rules with OWL ontologies with the goal to increase the expressivity of the logic, without affecting decidability [62]. These algorithms should provide alternative mechanisms for reasoning in the Semantic Web.

3.3 E-Government projects related to the ontologies

In this section we investigate the role of ontologies in the existing E-Government projects.

3.3.1 SmartGov

The overall aim of the SmartGov project\(^{15}\) is to specify, develop, deploy and evaluate a knowledge-based platform to assist public sector employees to generate online transaction services. This is achieved by simplifying their development, maintenance and integration with already installed IT systems. The SmartGov project, through its software platform, aims to minimise the reliance on IT skills to develop E-Government services. However, E-Government also brings new styles of communicating, new behaviours, new organisational structures, new processes, new paradigms, new threats and new opportunities. The framework for E-Government services includes reference models for:

- the **processes** behind the design and delivery of E-Government services;
- **co-operation** in public authorities, both internal and external;
- **social acceptance** of E-Government services.

It is intended to benefit any public authority that is planning or already delivering electronic transaction services, whether or not they have access to the SmartGov platform. It is designed to help improve co-operation, effectiveness and efficiency. The framework is underpinned by the **E-Government services ontology**. This is intended to provide a common understanding of the principles of E-Government services, an understanding from which people can communicate, discuss and build models of their own.

\(^{15}\) [http://www.smartgov-project.org/](http://www.smartgov-project.org/)
3.3.1.1 The SmartGov platform

The aim of the project was to develop a toolkit that will allow Public Authorities (PAs) to design, deploy and maintain online services with a minimum need of specialist IT assistance.

The key aspects of the SmartGov system are a knowledge base housing the domain knowledge required for E-Government services, and a SmartGov agent to allow the system to communicate with other IT systems, operating through an information exchange gateway. The information exchange gateway is the portal through which SmartGov communicates with other IT systems. It publishes an export schema containing the data items that need to be accessed by services running within the SmartGov framework. This schema serves as the vessel for communication but the responsibility for initiating and handling the communication lies with the SmartGov agent.

While a service is functioning within the operating environment, a SmartGov agent instance arranges for communication with other systems with which the service should exchange data. These systems include not only the organisation's installed systems (with which data should be exchanged) but extend to any system that may complement the running service. For instance the SmartGov agent might provide linkage to document repositories where detailed instructions on form filling or supporting legislation may be found.

There is also an application area housing the services created and a dissemination server to provide the interface between the users and the systems and the various public authority staff whose roles include designing services, administering them and managers who oversee it all.

3.3.1.2 Ontologies

Increasingly E-Government services are being developed that cut across old department lines and there is an increasing need for intra and inter governmental agencies to work more closely together, moving towards joined-up government. With this change comes the need for better communication between people and a need for a common vocabulary and understanding of terms that are being shared. An ontology provides such a communication between people and organisations. An ontology is an agreed set of concepts and relations that are meaningful to the members of the community it serves. It represents a view of a “world” that is commonly identifiable by those who know about the world. Its role is to be a common language through which knowledge about a specific domain can be described, organised and disseminated. Thus an ontology can be of considerable value to any large, complex organisation such as a public authority. The E-Government Services Ontology is at the core of the framework for the SmartGov processes, business process models and social aspects.

The modelling tool we are using for constructing the ontology was KAON (see section 3.1.2).
The enterprise ontology

We start by introducing the Enterprise Ontology. This work was undertaken by the Artificial Intelligence Applications Institute at The University of Edinburgh and its collaborative partners during the Enterprise Project, with the goal of creating a collection of terms and definitions relevant to business enterprises. Since its publication, the ontology has become widely accepted as a useful ontology of generic business activities. Recognising that many of these activities are common with public authorities, the E-Government service ontology can be built around it.

The Enterprise Ontology defines concepts within four broad categories: activity, organisation, strategy and marketing; it also imports a standard ontology of time. Some of the concepts formally defined within the Enterprise Ontology are listed below:

<table>
<thead>
<tr>
<th>Activity</th>
<th>(Activity, Execute, Effect…)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Organisation</td>
<td>(Person, Machine, Legal Ownership…)</td>
</tr>
<tr>
<td>Strategy</td>
<td>(Purpose, Hold Purpose, Risk…)</td>
</tr>
<tr>
<td>Marketing</td>
<td>(Actual Customer, Sale, Competitor…)</td>
</tr>
<tr>
<td>Time</td>
<td>(Time Interval…)</td>
</tr>
</tbody>
</table>

For the purposes of the SmartGov Project the first three categories of concepts of the Enterprise Ontology are very relevant. In the fourth category, marketing and selling are not activities typically undertaken by a public authority, and there are not usually any competitors. However, there exist many similarities between e.g a SALE and provision of a SERVICE, and with a degree of consideration and slight alteration of their definitions, many of these concepts can still be used.

While the purpose of an ontology is to produce a common understanding of a domain that can be shared, it cannot exist in isolation from the real world, and certain terms and concepts are required to be assumed in order to define the Ontology itself. This is the role of the Meta ontology presented in the next Section.

The Meta ontology

The Meta ontology provides the basic building blocks that are used to construct the ontology. These are primitives that are defined outside the context of the ontology and for the purposes of the ontology are assumed to have no other meaning than the ones is assigned to them. Since the ontology is based upon the Enterprise Ontology, the Enterprise Meta ontology is the most reasonable starting point for the Meta ontology that is required for the SmartGov project. The terms used in the Enterprise Meta ontology are given below:

<table>
<thead>
<tr>
<th>Entity:</th>
<th>a fundamental thing in the domain being modelled</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relationship:</td>
<td>the way that two or more Entities can be associated with each other</td>
</tr>
<tr>
<td>Role:</td>
<td>the way in which an Entity participates in a Relationship</td>
</tr>
<tr>
<td>Attribute:</td>
<td>a Relationship between two Entities (the “attributed entity” and the “value” entity) in which, within the scope of the model, for any</td>
</tr>
</tbody>
</table>
particular attributed Entity, the Relationship may exist with only one value Entity

**State of Affairs:** a situation; it consists of a set of Relationships between particular Entities; it can be said to hold, or be true (and conversely to not hold and be false)

**Achieve:** the realisation of a State of Affairs, *i.e.* being made true

**Actor Role:** a kind of Role in a Relationship whereby the playing of the Role entails some notion of doing or cognition

**Actor:** an Entity that actually plays an Actor Role in a Relationship

---

**Service Ontology**

The development of the E-Government Services Ontology for the SmartGov project consists of:

- Gathering the data (Interviews, Web documents, Word frequency counts of documents, Workshops);
- Defining the concepts;
- Structuring the ontology;
- Refer back to the experts.

3.3.2 ICTE-PAN

The ICTE-PAN\(^{16}\) Platform consists of some customisable modules – building blocks (e.g. collaboration workflow management system, electronic argumentation system, electronic forms system, etc.). Therefore, for every specific G2G collaboration process that is intended to be supported with the ICTE-PAN Platform, it is necessary to configure and customise it appropriately. This configuration process requires the selection of the appropriate modules, which are needed for supporting this process, and then their customisation and their integration according to the specific needs of the process. In order to support the user-centred and participative configuration and customisation of the ICTE-PAN Platform for any given process, with minimal involvement of technical experts, a Process Modelling Methodology is required. The objective of this Process Modelling Methodology is to enable the users to describe the process they want to support with the ICTE-PAN Platform in their own ‘language’ and based on PA business concepts (process, activity, case, actor, stakeholder, law, regulation, organisational unit, issue, alternative, etc.), which are familiar to them, excluding the use of technical concepts.

According to the user requirements analysis this Process Modelling Methodology should fulfil the following requirements:

- offer capabilities for modelling Collaboration Processes, with both ‘Single Person Activities’) and ‘Collaboration Activities’;

---

\(^{16}\) [http://www.provincia.genova.it/europa/progettieurop/iictepan/ictepan.htm](http://www.provincia.genova.it/europa/progettieurop/iictepan/ictepan.htm)
• offer capabilities to model for each Collaboration Activity what kind of collaboration it includes and in general to support structured collaboration;
• offer capabilities for modelling both ‘hard’ elements and ‘soft’ elements;
• incorporate the specificities of Public Administration.

Furthermore, the ICTE-PAN Process Modelling Methodology must comply with existing or emerging relevant standards in order to offer high interoperability levels. Initially, it was examined whether it is possible to use any of the existing Business Process Modelling Methodologies for achieving the above objectives. However, none of the existing Business Process Modelling Methodologies could fulfil all the above requirements and offer all these required capabilities. For this reason, the optimal solution was to develop a new Public Administration Operations Modelling Integrated Methodology (PA-OMIM) that could fulfil all these requirements.

3.3.2.1 PA-OMIM (Process Modelling Methodology)

PA-OMIM is a process modelling methodology (and should not be confused with a systems or software development methodology); i.e. it constitutes a language to create models, which are illustrating the operation of a single PA or the cooperation of several PAs. Any G2G collaboration process is modelled through PA-OMIM and consists of five views, each of them modelling a different perspective of the process. The five views supported by PA-OMIM are: the Environment View, the Process View, the Organisation View, the Resource View and the Information/Knowledge View.

The main innovative characteristics of PA-OMIM are the following:
• It enables modelling of complex Collaboration Processes, consisting of both ‘Single Person Activities’ (SPAs), and ‘Collaboration Activities’ (CAs);
• It enables for each Collaboration Activity (CA) to model what kind of collaboration it includes, i.e. to define what kinds of elements each actor can contribute (e.g. problems, alternatives, positions, criteria, actions, etc.), and which are the allowed associations between these elements, (e.g. an alternative can be associated with a problem, etc.). These definitions constitute the Specific Ontology of the CA and they are used in the subsequent stages for specifying the required configuration and customisation of the ICTE-PAN Platform, in order to create an electronic environment for supporting this Collaboration Activity (CA);
• It includes a General Ontology for the domains of Public Sector Collaborative Decision Making and Policy - Programmes Design and Management, which consists of the main concepts-elements used in these domains and the main associations among them. The proposed ontology aims at assisting and supporting end-users to model Collaboration Activities (CAs) and in particular for defining its Specific Ontology. The concepts-elements and the associations of this General Ontology are used for assisting and supporting the definition of the elements, which can be contributed by the participants-actors in a CA, and the associations, which are allowed among them. This General Ontology establishes a common vocabulary and thus
facilitates knowledge sharing and collaborative creation of new knowledge among public organisations.

In order to support the above-required modelling-definition for the Specific Ontology of each CA, a General Ontology has been developed in the domains of public sector policies and programmes development, monitoring and evaluation, and public sector decision-making. It consists of the main concepts (i.e. categories or kinds of discussion elements) used in these domains and their relations.

3.3.2.2 Ontology

There is a big variety of CA types in the PA, which differ in the kinds of elements contributed by the participants, and the kinds of associations allowed among them. For this reason it is necessary during the definition of the activities of a collaborative process in the Process View, for each CA to proceed to modelling this aspect of it. Therefore, it is necessary to define the kinds of elements (e.g. issues, alternatives, arguments, programmes, projects, tasks, etc.) which can be contributed by the participants in this CA, and also the kinds of associations which are allowed to be made among these elements (e.g. an alternative can be associated with an issue). Based on these definitions, the ICTE-PAN Platform for each case creates the appropriate electronic environment for the execution of this CA.

In order to support the above definitions, an Ontology is required for the domains of PA policies and programmes development, monitoring and evaluation, and also PA decision-making, consisting of the main concepts (i.e. categories or kinds of discussion elements) used in these domains and their associations. Such an Ontology was developed, based on the relevant research literature in this area, on the analysis of the pilots of ICTE-PAN Project and in general of the PAs of the four participating countries, and also on the general experience of the members of the project team in these domains.

Development of an Ontology

According to the previous chapter, a collaborative process in general consists of ‘Single Person Activities’ (SPAs) (in a SPA for each case only one individual is involved), and ‘Collaborative Activities’ (CAs) (in a CA for each case more than one individuals are involved, contribute, collaborate and interact). For each CA, in order to create an electronic collaborative environment based on the ICTE-PAN Platform for supporting it (where all the participants-actors will contribute, collaborate and interact electronically), it is necessary to determine/define during the modelling of the Process (and in particular during the definition of the specific CA) what kind of collaboration it includes: in particular it is necessary to determine/define what kinds of elements each participant-actor should contribute (e.g. problems, alternatives, positions, evaluation criteria, programs, subprograms, projects, tasks, etc.), and which are the allowed associations-relations between these elements (e.g. an alternative can be associated with a problem, a position can be associated with an alternative or with another position, etc.). In order to support the above determination/definition of the kinds of elements which can be contributed by the participants-actors in a CA, and the relations which are allowed among them, an Ontology is required for the domain of Public Sector Collaborative Public Policy & Programmes Development, Monitoring and Management, and Decision making, which is the main area of G2G collaboration.
This Ontology should include the main concepts (i.e. kinds of discussion elements) used in this domain and their relations.

For the above reasons such an ontology is developed for the domain of Public Sector Collaborative Decision Making, Policy Development and Programmes Design & Management.

**Usage of the Ontology**

For each Collaborative Activity (CA) it can be determined/defined the kind of collaboration it includes by determining/selecting, during the modelling of the Process (and in particular during the modelling of the specific CA), a small subset of the kinds of elements and relations of the Ontology, that will characterise this CA and constitute the ‘specific ontology’ of it; only these few kinds of elements will be contributed by the participants-actors in this CA, and only these few kinds of relations will be allowed among them. In this way it is possible to determine the nature of the CA and the nature of the collaboration and interaction that will take place among the participants-actors in this CA. This is necessary because it is not practically possible in the same CA all the numerous kinds and relations of the Ontology to be dealt with (i.e. to allow the participants-actors to contribute all these kinds of elements, and also to use all these kinds of relations), because in this case the whole ‘discussion’, collaboration and interaction would be inefficient and unproductive. For this reason a Decision Making Process, a Policy & Programmes Development Process, or a Programme Design & Management Process are defined into appropriate discrete Activities, each of them dealing with a subset of the kinds of elements and relations of the Ontology.

Also for each of the participants-actors in the CA a specific access permissions on these kinds of elements and relations is given. The access permissions are defined to characterise the CA, e.g. some participants-actors can be given only read permissions on all kinds and relations of the CA, so that they can read the contributions of other participants-actors, but they can not enter (write) contributions, therefore they will be just ‘observers’ of this CA.

For example, for a CA the following elements can be selected from the Ontology:

- the kinds of elements ISSUEs, ALTERNATIVEs and POSITIONs;
- the kind of relations RESOLVES between ISSUEs and ALTERNATIVEs, and the kinds of relations SUPPORTS, OBJECTS and COMMENTS between ALTERNATIVEs and POSITIONs.

An example is shown in Figure 5.

3.3.3 E-Power

E-POWER\(^{17}\) is an IST project under the fifth Framework focusing on a methodological approach for formalizing different forms of legislation into UML models. E-POWER will implement a knowledge management solution by providing a method and tools that help to improve the quality of legislation while the enforcement of law is being facilitated. Both method and tools will decrease the time to market.

\(^{17}\) [http://www.lri.jur.uva.nl/~epower/](http://www.lri.jur.uva.nl/~epower/)
new/changed legislation and to facilitate the maintenance of legislation and it will improve the access to the governmental body of knowledge by offering new E-services.

Furthermore the use of this method and tools will result in a more efficient use of scarce knowledge resources. The E-POWER project will result in transparency of pension arrangements for the (future) elderly citizens.

The project will develop tools that help with the harmonisation of pension regulations. By providing easy access (using the Internet) to vital information the project will contribute to the social inclusion of citizens. E-POWER will consequently improve the effectiveness and efficiency of public administrations and contribute to the completion of the internal market.

Figure 6 shows the E-Power toolkit.
3.3.3.1 Ontologies

In the last decades the interest in the problem of comparing and harmonizing legislation has been steadily increasing. One reason is the increasing legal convergence between governments in the European Union, and the increasing traffic of people over borders of jurisdictions. Another reason is the increasing globalization of companies.

The increase in comparing legal systems is evidenced by the number of consultancy firms that advertise their knowledge of multiple legal systems to companies. In addition, there are a number of initiatives for constructing international legal ontologies that expose the subsumption relations between legal vocabulary in multiple jurisdictions. It has also been suggested that the problem of comparing legislation is, at least if it concerns versions of a regulation in time in the same legal system, a special case of the general problem of ontology integration. The E-Power projects attempts to show that this is a promising approach, but that the problem of ‘comparing’ legislation is broader than the problem of ontology integration.

In the E-POWER project relevant tax legislation and business processes are modelled in Unified Modelling Language (UML) to improve the speed and efficiency with which the Dutch Tax and Customs Administration (DTCA) can implement decision support systems for internal use and for its clients. The ontologies have also proven their usefulness for efficient and effective analysis of draft legislation, allowing the DTCA to give immediate feedback to drafts of the new income tax law of 2001.
The available ontologies can be used to compare ‘similar’ legislation from different jurisdictions. Employees in the DTCA are increasingly confronted with requests that require them to understand European regulations and directives, and regulations of other EU member states. Better insight in the process of modelling and comparing legislation from different legislators is expected to improve the capacity of the DTCA to react to future consequences of increased movement of people, products, and money between EU member states and increased harmonization between tax authorities in Europe. In addition, the discovery of the requirements of comparing models is also expected to result in a more principled, more robust, and language-independent methodology for modelling legislation, because it decouples the model of legislation from the purpose for which it was modelled and makes assumptions about the surrounding context, and knowledge expected from users, explicit. Modelling one’s own regulations often fails to achieve this level of context-independence.

3.3.3.2 MetaLex

MetaLex\(^{18}\) is one of the first official results of the E-POWER project. It provides a generic and easily extensible framework for the XML encoding of the structure and contents of legal and paralegal documents. It differs from other existing metadata schemes for legal documents in two respects: it is language independent and accommodates the use of XML beyond search and presentation services.

MetaLex facilitates:

- accessibility of legislation: search, retrieval, presentation, multi-lingual aspects;
- exchangeability of legislation: in between applications as well as in between other actors, even across borders;
- quality aspects of legislation: by formalizing the legislation defects are detected in an early stage;
- maintainability and extendibility of legislation

In a world where civilians demand correct and instant information from their governments there is an increasing need to have access to legislation in a way that goes beyond the simple publication of the textual representation. MetaLex provides the necessary framework for opening up legislation in a more intelligent and accessible way to different target audiences.

The MetaLex standard has the following features:

- **Multiple Languages** - The same content written down in one MetaLex document using multiple languages; Jurisdiction(/locale)-specific languages for MetaLex XML-tags;
- **Version Management and Maintenance** - MetaLex uses a comprehensive way to determine the validity and activity of a part of legal text, using four attributes (date-enacted, date-repealed, date-publication, date-effective). This supports automatic generation of current and past versions of a legal text;

\(^{18}\) [http://www.metalex.nl/pages/welcome.html](http://www.metalex.nl/pages/welcome.html)
- **Extensibility** - MetaLex supports mixing with other XML-based standards such as XHTML. MetaLex is ready for embedding in techniques for knowledge representation, code generation, rule generation and verification and validation techniques;

- **Exchange** - MetaLex is a comprehensive format for exchange between different parties;

- **Presentation** - MetaLex documents can be easily converted into different presentation formats, such as XHTML, PDF, RTF etc.;

- **Search and Filtering** - MetaLex facilitates search and filtering on legal documents;

- **Overview** - MetaLex captures the relations between different (bodies of) legal sources. This gives insight into the structure of the law.

### 3.3.4 eGov

Within the eGOV\(^{19}\) project, an integrated platform for realizing online one-stop government was developed. This one-stop government platform allows the public sector to provide citizens, business partners and administrative staff with information and public services based on life events and business situations hence increasing the effectiveness, efficiency and quality of public services. Even though the eGov project does not use ontologies, it is very relevant for the OntoGov project. For example, it defines the metadata standard that can be extended into an ontology for the semantic description of the E-Government services. Consequently, the short description of this project is included in the related work.

More specifically, the eGOV online one-stop government platform:

- Implements the concept of online one-stop government. Allows public authorities to provide integrated services. Service integration is based on the open technology of Web Services:
  - Universal Description Discovery and Integration (UDDI);
  - Web Services Description Language (WSDL);
  - Simple Object Access Protocol (SOAP);

- Achieves interoperability with legacy systems of different public authorities by means of open standards such as XML;

- Uses ontologies in order to enhance the navigation of the end-users to the E-Government portal. The navigation is based on a life-event hierarchy, which are implemented by means of an RDF schema.

### 3.3.4.1 The eGOV technical architecture

From a technical point of view, the eGOV project identified and addressed three key development and implementation modules:

\(^{19}\) [http://falcon.ifs.uni-linz.ac.at:8080/eGOV/](http://falcon.ifs.uni-linz.ac.at:8080/eGOV/)
• The eGOV one-stop government portal and network architecture;
• The eGOV middleware containing the content and service repository, the service creation environment and the service runtime environment;
• A common format for the data flow between the portal and the service repositories in different public authorities, which is termed Governmental Markup Language (GovML). GovML (see section 4.3.5.1) is based on open standards such as XML, RDF and Dublin Core. It contains data vocabularies for describing content of life events and public services, and it provides a standard set of metadata for life events and public service resources.

3.3.4.2 Semantics added to WS in eGOV

For every WS semantics are inserted into the RDF engine (SIR) via the GUI of SIR (see Figure 7). For example the title of the service, subject, date of publication, the service name as stored within the UDDI etc are inserted into a relational database (conceptual rdf graph repository) by filling in the respective input fields and submitting the HTML form. Some of the semantics inserted for each WS are illustrated in the Services portlet: More specific in the web service contained in the portlet the following semantics are provided to the portal user:

• Title: “Marriage Registration Service”
• Subject: “Test Service for Marriage registration” etc
• Date of creation

Unless the appropriate semantics were inserted into the SIR engine, the web service could not appear in the Services portlet.

3.3.4.3 Lifecycle of a GovML Document According to eGOV Project

Certain processes at public authorities support the production and provision of GovML documents. In order to demonstrate the life cycle of a GovML document, a simplified scenario is described. It covers all stages, from document creation until its eventual depiction to consumers.

Supposing that a public authority is planning to provide citizens with the ability to obtain a birth certificate online. Firstly, a public servant is assigned with the administrator role. At a next step, the description of the public service should be generated in GovML (see section 4.3.5.1). For this purpose, a web application named Service Description Tool (SDT) is utilised. SDT has been implemented within the eGOV project, to enable the management of GovML documents. SDT enables administrators of eGOV platform at public agencies to create, retrieve, update and delete XML documents via a user friendly Graphical User Interface, following a sequence of predefined steps. Advanced XML techniques (i.e. indexing mechanisms based on Xpath expressions) are applied in order to enhance the efficient manipulation of GovML documents. SDT is a multilingual application, currently supporting English, German, French and Greek. The SDT tool, which manipulates GovML data vocabularies, is illustrated in Figure 8.
By using SDT, the administrator fills out the elements of the new GovML document. Their content derives from the related governmental law, rules at a national level and the regulations of the public authority (in case of a specific GovML). In this example, it is assumed that the administrator produces a specific service description. The element identifier aims to uniquely identify the GovML for internal management reasons (i.e. storage, retrieval etc). It is automatically filled in by the SDT. Language is an element of type xml:lang, illustrating the language of the document. It is also filled in by the system implicitly, after the administrator’s language choice from the SDT. Title and description describe the title and a short description of the provided public service respectively. Eligibility element explains which citizens have the right to consume the public service i.e. who can apply for issuing a birth certificate. The required documents element provides the list of transcripts that are considered prerequisite for the provision of the service; in our example these could be passport or identity card. Procedure describes analytically the steps that the consumer has to follow, and periodicity illustrates how often the service is available. In this scenario it could have the value “every working day”.

The new document is stored in the local repository of the public organisation, which contains all GovML documents of this organization.

The final step is to transform the document to a specific display format e.g. HTML or WML. This can be achieved with the utilisation of an appropriate XSLT script on the public authority side. The transformation mechanism is transparent to users, who are given the opportunity to access the desired information in multiple formats, for

**Figure 7. SIR RDF engine gui**
example HTML through a web browser or WML through a mobile phone supporting the WAP protocol.

![Screen shot of SDT tool](image.png)

**Figure 8.** SDT tool

### 3.3.5 Conclusion

Based on the analysis of the projects that are related to the usage of ontologies in the E-Government we came to the following conclusion:

- SmartGov, ICTE-PAN and E-Power projects use ontologies for representing the profile of a service;
- Lifecycle aspects were considered in the SmartGov and ICTE-PAN projects;
- Ontologies for the legal documents have been developed in the E-Power project;
- In the E-Power and in the eGov projects the E-Government metadata standards were defined. These standards can be extended into ontologies for describing the profile of a service.

Therefore, all of these projects provide significant inputs for the ontoGov project. However, we will extend their results by taking into account the whole lifecycle of the E-Government services. It means that we will use ontologies not only for the service configuration but also for the service reconfiguration.
4 Web Services

4.1 Definition

There are several definitions of Web Services. According to the IBM web service tutorial [14], “web services are the new breed of web application. They are self-contained, self-describing, modular applications that can be published, located, and invoked across the web. Web services perform functions, which can be anything from simple requests to complicated business processes. Once a Web service is deployed, other applications (and other Web services) can discover and invoke the deployed service”.

In [4], web services constitute software modules that “describe a collection of operations that are network-accessible through standardized XML messaging”. Gartner\(^{20}\) defines web services as “loosely coupled software components that interact with one another dynamically via standard Internet technologies. “

Indeed, web services enable application development and integration over the Web by supporting program-to-program interactions. A web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.

Web services are most commonly defined using the Web Services Definition Language (WSDL). That definition is stored in repositories. The most common registry is the Universal Description, Discovery, and Integration (UDDI) registry. It is intended to eventually serve as a means of "discovering" Web Services described using WSDL. The idea is that the UDDI registry can be searched in various ways to obtain contact information and the Web Services available for various organizations. The most common form of messaging to communicate with a repository is SOAP. SOAP provides the envelope for sending messages over the Internet/Internet. SOAP commonly uses HTTP, but other protocols such as Simple Mail Transfer Protocol (SMTP) may by used. SOAP can be used to exchange complete documents or to call a remote procedure.

Figure 9 depicts the conceptual architecture of a system that supports web services. Entities in the basic architecture can take on one or all of the following roles:

- **Service requester** - requests the execution of a web service;
- **Service provider** - processes a web service request;
- **Service registry** - registry through which a web service description is published and made discoverable.

A web service requester interacts via XML based messages on a transport protocol with a web service registry or a web service provider. The dynamic binding to the

\(^{20}\) http://www.gartner.com/
consumed web services is done using the information from the web service registry. A web service provider is the provider of the web service. It publishes information about the web service to the registry and offers a programmatic access the consumer can bind to. The web service registry contains information about location and offered web services from all web service providers.

The infrastructures for Web services are also readily available through application servers like IBM’s WebSphere, Microsoft’s .NET Framework or BEA’s Weblogic. All of these application servers provide support for SOAP, WSDL and UDDI connectivity.

![Figure 9. Web Services](image)

4.2 Semantic Web Services

Web services are designed to provide interoperability between diverse applications. Composing services dynamically to create new functionality is necessary when the required task cannot be realized directly by the existing services. It is an ongoing research activity to automate this process, but accomplishing this goal with a human controller as the decision mechanism can be achieved. The main problem is the gap between the concepts people use and the data computers interpret. This barrier can be overcome using Semantic Web technologies.

Semantic Web Services are web services with a formal description i.e. semantics. The semantic description of Web services allows:

- to better advertise and subsequently discover Web services and
- to supply a better solution for the selection, composition and interoperation of Web services.

The core idea of **Semantic Web Services** is to combine Web Services and Semantic Web worlds and their potential benefits. The benefits of the integration include increased visibility of Web services, because open ontology frameworks allow for
semantically expressive advertising on the Web that may be found by Web crawlers. They include better usability because of more expressive Web service descriptions. They include a smooth evolution from Web services for human users such as targeted by current industry (quasi-)standards toward Web services for personalized machine agents that assist the user. The differences between “traditional” Web Services and Semantic Web Services are shown in Table 2.

Table 2. Dimensions of Web service features

<table>
<thead>
<tr>
<th>Dimension</th>
<th>Web Services features</th>
<th>Semantic Web Services features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service</td>
<td>Simple</td>
<td>Composed</td>
</tr>
<tr>
<td>Requestor</td>
<td>Human</td>
<td>Machine</td>
</tr>
<tr>
<td>Provider</td>
<td>Registration</td>
<td>No registration</td>
</tr>
<tr>
<td>Service description</td>
<td>Taxonomy</td>
<td>Ontology</td>
</tr>
<tr>
<td>Descriptive elements</td>
<td>Closed world</td>
<td>Open world</td>
</tr>
<tr>
<td>Data exchange</td>
<td>Syntactic-based</td>
<td>Semantics-based</td>
</tr>
</tbody>
</table>

In particular, the Semantic Web will allow giving richer descriptions of Web services (e.g., semi-structured data, types, inheritance, and semantic constraints). The key role of the broker may disappear, it may still be viable as a kind of search machine for Web services (with meta search engines on top), but it will lose its central role, because everyone may publish semantic descriptions and crawlers may find them. Personalized machine agents will take over the role of a service requestor from the human user. And, they may also do the composition for the human user.

4.3 E-Government Metadata Standards and Initiatives

4.3.1 Introduction

Semantic web services are closely related to metadata, which facilitate their advertisement and subsequent discovery by their requestors. In the public sector, metadata can be used for the discovery and retrieval of governmental information.

An increasing number of governments worldwide recognize the role of establishing a metadata standard as an integral ingredient of their interoperability framework towards realizing their E-Government strategy.

Section 4.3 presents an overview and comparison of E-Government metadata initiatives throughout the world. Although all initiatives presented in this section mainly deal with governmental resources (e.g. word, pdf documents, etc.), they can also serve as a basis for other, new initiatives which will aim at facilitating the discovery and execution of semantic web services.

A review of the Dublin Core and the work within the Dublin Core - Government Working Group is also presented. Moreover, a review of national metadata standards and a comparison to Dublin Core can be found thereafter. Finally, a metadata element
set that has been proposed by the European Committee for Standardization (CEN) is presented.

4.3.2 Definitions

The following definitions have been used in the following sections:

- **Element**: An Element is a property of a resource. Furthermore, “properties” are attributes of resources, i.e. characteristics that a resource may “have”, such as a Title, Publisher, or Subject;
- **Element Refinements**: An Element Refinement is a property of a resource that shares the meaning of a particular Element, but with narrower semantics. In DC practice, an Element Refinement refines just one parent DC property;
- **Application Profile (AP)**: An Application Profile is a declaration of which metadata terms an organization, information resource, application, or user community uses in its metadata [47].

4.3.3 Dublin Core

The Dublin Core Metadata Initiative has proposed a set of 16 elements as shown in Table 3.

<table>
<thead>
<tr>
<th><strong>Element</strong></th>
<th><strong>Definition</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Title</td>
<td>A name given to the resource</td>
</tr>
<tr>
<td>Creator</td>
<td>An entity primarily responsible for making the content of the resource</td>
</tr>
<tr>
<td>Subject</td>
<td>The topic of the content of the resource</td>
</tr>
<tr>
<td>Description</td>
<td>An account of the content of the resource</td>
</tr>
<tr>
<td>Publisher</td>
<td>An entity responsible for making the resource available</td>
</tr>
<tr>
<td>Contributor</td>
<td>An entity responsible for making contributions to the content of the resource</td>
</tr>
<tr>
<td>Date</td>
<td>A date associated with an event in the life cycle of the resource</td>
</tr>
<tr>
<td>Type</td>
<td>The nature or genre of the content of the resource</td>
</tr>
<tr>
<td>Format</td>
<td>The physical or digital manifestation of the resource</td>
</tr>
<tr>
<td>Identifier</td>
<td>An unambiguous reference to the resource within a given context</td>
</tr>
<tr>
<td>Source</td>
<td>A Reference to a resource from which the present resource is derived</td>
</tr>
<tr>
<td>Language</td>
<td>A language of the intellectual content of the resource</td>
</tr>
<tr>
<td>Relation</td>
<td>A reference to a related resource</td>
</tr>
<tr>
<td>Coverage</td>
<td>The extent or scope of the content of the resource</td>
</tr>
<tr>
<td>Rights</td>
<td>Information about rights held in and over the resource</td>
</tr>
<tr>
<td>Audience</td>
<td>A class of entity for whom the resource is intended or useful.</td>
</tr>
</tbody>
</table>
4.3.4 National Metadata Standards

The following DC-based national standards exist:

- UK E-Government Metadata Standard\(^{21}\) ver. 2.0, April 2003;
- Irish Public Service Metadata Standard\(^{22}\) ver. 1.0, August 2001;
- The Danish Government Metadata;
- The Finnish Government Metadata Standard\(^{23}\);
- The Iceland Government Metadata Standard;
- The Australian Government Locater Service (AGLS)\(^{24}\);
- The New Zealand Government Locater Service (NZGLS)\(^{25}\);
- The Canadian Government On-Line Metadata Standard\(^{26}\).

The additional metadata elements that are used in the investigated national standards and are not included in DC are summarized in Table 4.

It should be noted that two elements of the Icelandic standard (namely Quality and Standards) are not included in Table 4. This is due to the fact that the Icelandic standard was mainly focused on educational purposes and the relevance of these elements to E-Government is not adequately documented. Despite this fact, there are still nineteen elements in use by various governments that are not included in DC.

4.3.5 Towards a Universal E-Government Metadata Standard

The following DC-based international initiatives and projects exist:

- DC-Government (DC-Gov) Working Group\(^{27}\);
- MIReG project funded by the IDA initiative\(^{28}\);
- IST eGOV RTD project co-funded by the European Commission\(^{29}\);
- CEN MMI-DC Workshop project\(^{30}\) entitled “A Dublin Core E-Government profile and model”.

---

\(^{21}\) http://www.e-envoy.gov.uk/Resources/Guidelines/fs/en
\(^{22}\) http://www.gov.ie/webstandards/metastandards/index.html
\(^{23}\) http://www.intermin.fi/juhta/suosituksset/jhs143.htm
\(^{27}\) http://dublincore.org/groups/government/profile-200202.shtml
\(^{28}\) http://dublincore.org/groups/government/mireg-metadata-20010828.shtml
\(^{29}\) http://www.egov-project.org
\(^{30}\) http://www.cenorm.be/sh/mmi-dc
The first two projects are ongoing and have not yet finalized their work. In the rest of this section we describe the last two standards in more details.

### Table 4. Additional Metadata

<table>
<thead>
<tr>
<th>Element</th>
<th>National Standard</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accessibility</td>
<td>UK</td>
</tr>
<tr>
<td>Disposal</td>
<td>UK</td>
</tr>
<tr>
<td>Location</td>
<td>UK</td>
</tr>
<tr>
<td>Preservation</td>
<td>UK</td>
</tr>
<tr>
<td>Status</td>
<td>UK</td>
</tr>
<tr>
<td>Addressee</td>
<td>UK</td>
</tr>
<tr>
<td>Aggregation</td>
<td>UK</td>
</tr>
<tr>
<td>Digital signature</td>
<td>UK</td>
</tr>
<tr>
<td>Mandate</td>
<td>UK</td>
</tr>
<tr>
<td>Receiver</td>
<td>DK, FI</td>
</tr>
<tr>
<td>Process</td>
<td>DK</td>
</tr>
<tr>
<td>DocumentType</td>
<td>FI</td>
</tr>
<tr>
<td>Publicity</td>
<td>FI</td>
</tr>
<tr>
<td>Version</td>
<td>FI, IC</td>
</tr>
<tr>
<td>Environment</td>
<td>FI</td>
</tr>
<tr>
<td>Mandator</td>
<td>FI</td>
</tr>
<tr>
<td>Availability</td>
<td>FI, AU, NZ</td>
</tr>
<tr>
<td>Metadata</td>
<td>IC</td>
</tr>
<tr>
<td>Function</td>
<td>AU, NZ</td>
</tr>
</tbody>
</table>

#### 4.3.5.1 GovML – based on DublinCore metadata

The Governmental Markup Language (GovML) has been proposed by the IST eGOV project to define structures (or vocabularies) for governmental data (see section 4.3.5.1.1) and metadata (see section 4.3.5.1.3).

#### 4.3.5.1.1 GovML data vocabularies

Online information provided by public authorities does not comply with a specific structure. Furthermore, it is not characterized by a unified presentation style and principles. On the contrary, it is very common that each public authority applies its own format to the presentation of information, hence preventing users from a unique experience, whenever they interact with the public sector.
Apart from presentation of public information, another important issue in online one-stop government is interoperability. It is common in E-Government projects that several governmental data need to be exchanged among public authorities. Each authority however, usually stores governmental information in a proprietary data format and storing system, e.g. HTML, ASCII, RDBMS etc. Therefore data conversion must take place from the format adopted by one public authority to the format adopted by others, whenever exchange of data is needed. This data conversion can become a significant overhead to the Information Technology (IT) responsibilities of public authorities.

The main aim of GovML data vocabularies is to standardise (in XML format) the information describing public services and life events.

GovML proposes a common structure for describing public services, in order to ease production and management of governmental information. Moreover, interoperability with other agencies would be fostered.

The more structured the information is, the more easily a human and/or a machine can understand it and use it to make knowledgeable decisions.

The final GovML data structure consists of three sub-vocabularies, two for describing public services and one for life events.

Technically, each sub-vocabulary is characterized by a set of predefined XML elements. The three GovML vocabularies are analysed in the next paragraphs.

**Generic description data vocabulary for public services**

This vocabulary defines a common standard for the content of all public authorities at a national level. Such governmental content is created only once, at a national level. This type of content could be normally based upon a governmental law, so it can be adopted by all public agencies of a country. Examples of data elements are: title, procedure, required documents, etc.

**Specific description data vocabulary for public services**

This vocabulary caters for the creation of content related to a public service provided by a specific public authority. It can be considered as a specialization of the generic description vocabulary, because the values of some elements of this vocabulary depend on the public authority, which provides the public service. Some of its elements are: name and address of the public authority, public servant contact details, delivery channel of the service, etc.

It should be noted that the generic and specific data vocabularies for public services have many common elements.

**Data vocabulary for life events and business situations**

This vocabulary defines a set of elements necessary to describe any every day life event or business situation. Elements of this vocabulary are a subset of the generic description data vocabulary for public services. The three GovML data vocabularies are listed below.
### Table 5. GovML data vocabularies

<table>
<thead>
<tr>
<th>Seq</th>
<th>Public Services</th>
<th>Life Events</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Generic description</td>
<td>Specific description</td>
</tr>
<tr>
<td>1</td>
<td>identifier</td>
<td>identifier</td>
</tr>
<tr>
<td>2</td>
<td>language</td>
<td>language</td>
</tr>
<tr>
<td>3</td>
<td>title</td>
<td>title</td>
</tr>
<tr>
<td>4</td>
<td>description</td>
<td>description</td>
</tr>
<tr>
<td>5</td>
<td>attention</td>
<td>attention</td>
</tr>
<tr>
<td>6</td>
<td>faq-list</td>
<td>faq-list</td>
</tr>
<tr>
<td>7</td>
<td>eligibility</td>
<td>eligibility</td>
</tr>
<tr>
<td>8</td>
<td>required-documents</td>
<td>required-documents</td>
</tr>
<tr>
<td>9</td>
<td>procedure</td>
<td>procedure</td>
</tr>
<tr>
<td>10</td>
<td>periodicity</td>
<td>periodicity</td>
</tr>
<tr>
<td>11</td>
<td>time-to deliver</td>
<td>time-to deliver</td>
</tr>
<tr>
<td>12</td>
<td>cost-Info</td>
<td>cost-Info</td>
</tr>
<tr>
<td>13</td>
<td>service-hours</td>
<td>service-hours</td>
</tr>
<tr>
<td>14</td>
<td>employee-hints</td>
<td>employee-hints</td>
</tr>
<tr>
<td>15</td>
<td>citizen-hints</td>
<td>citizen-hints</td>
</tr>
<tr>
<td>16</td>
<td>related-services</td>
<td>public-authority-name</td>
</tr>
<tr>
<td>17</td>
<td>audience</td>
<td>Public authority department</td>
</tr>
<tr>
<td>18</td>
<td>public-authority-type</td>
<td>e-documents</td>
</tr>
<tr>
<td>19</td>
<td>law</td>
<td>delivery-channel</td>
</tr>
<tr>
<td>20</td>
<td>result</td>
<td>cost</td>
</tr>
<tr>
<td>21</td>
<td></td>
<td>contact-details</td>
</tr>
<tr>
<td>22</td>
<td>service-code</td>
<td>automation-level</td>
</tr>
<tr>
<td>23</td>
<td></td>
<td>public-authority-address</td>
</tr>
<tr>
<td>24</td>
<td></td>
<td>state</td>
</tr>
<tr>
<td>25</td>
<td></td>
<td>service-name</td>
</tr>
<tr>
<td>26</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### 4.3.5.1.2 Properties of GovML data vocabularies

### Table 6. GovML General description data vocabulary

<table>
<thead>
<tr>
<th>Seq</th>
<th>Element</th>
<th>Brief Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>identifier</td>
<td>Unique identifier of the document</td>
</tr>
<tr>
<td>2</td>
<td>language</td>
<td>Language of the GovML Document</td>
</tr>
<tr>
<td>3</td>
<td>title</td>
<td>Title of the life-event or public service</td>
</tr>
<tr>
<td>Seq</td>
<td>Element</td>
<td>Brief Description</td>
</tr>
<tr>
<td>-----</td>
<td>------------------</td>
<td>------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>1</td>
<td>identifier</td>
<td>Unique identifier of the document</td>
</tr>
<tr>
<td>2</td>
<td>language</td>
<td>Language of the GovML Document</td>
</tr>
<tr>
<td>3</td>
<td>title</td>
<td>Title of the life-event</td>
</tr>
<tr>
<td>4</td>
<td>description</td>
<td>Description of the life-event</td>
</tr>
<tr>
<td>5</td>
<td>attention</td>
<td>What the citizen / business should pay attention at</td>
</tr>
<tr>
<td>6</td>
<td>faq-list</td>
<td>Questions asked frequently by citizens, businesses along with their answers from</td>
</tr>
<tr>
<td></td>
<td></td>
<td>the respective public authorities</td>
</tr>
<tr>
<td>7</td>
<td>related-services</td>
<td>Public services related to the life event</td>
</tr>
<tr>
<td>8</td>
<td>law</td>
<td>Identification of the law related to the public service</td>
</tr>
</tbody>
</table>

**Table 7. GovML data vocabulary for life events**

<table>
<thead>
<tr>
<th>Seq</th>
<th>Element</th>
<th>Brief Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>identifier</td>
<td>Unique identifier of the document</td>
</tr>
<tr>
<td>2</td>
<td>language</td>
<td>Language of the GovML Document</td>
</tr>
<tr>
<td>3</td>
<td>title</td>
<td>Title of the life-event</td>
</tr>
<tr>
<td>4</td>
<td>description</td>
<td>Description of the life-event or public service described</td>
</tr>
<tr>
<td>5</td>
<td>attention</td>
<td>What the citizen / business should pay attention at</td>
</tr>
<tr>
<td>6</td>
<td>faq-list</td>
<td>Questions asked frequently by citizens, businesses along with their answers from</td>
</tr>
<tr>
<td></td>
<td></td>
<td>the respective public authorities</td>
</tr>
<tr>
<td>7</td>
<td>related-services</td>
<td>Public services related to the life event</td>
</tr>
<tr>
<td>8</td>
<td>law</td>
<td>Identification of the law related to the public service</td>
</tr>
</tbody>
</table>

**Table 8. GovML Specific description data vocabulary**

<table>
<thead>
<tr>
<th>Seq</th>
<th>Element</th>
<th>Brief Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>identifier</td>
<td>Unique identifier of the document</td>
</tr>
<tr>
<td>2</td>
<td>language</td>
<td>Language of the GovML Document</td>
</tr>
<tr>
<td>3</td>
<td>title</td>
<td>Title of the life-event</td>
</tr>
<tr>
<td>4</td>
<td>description</td>
<td>Description of the life-event or public service described</td>
</tr>
<tr>
<td>5</td>
<td>attention</td>
<td>What the citizen / business should pay attention at</td>
</tr>
<tr>
<td>Seq</td>
<td>Element</td>
<td>Brief Description</td>
</tr>
<tr>
<td>-----</td>
<td>---------------</td>
<td>------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>6</td>
<td>faq-list</td>
<td>Questions asked frequently by citizens, businesses along with their answers</td>
</tr>
<tr>
<td></td>
<td></td>
<td>from the respective public authorities</td>
</tr>
<tr>
<td>7</td>
<td>eligibility</td>
<td>Ability of a citizen / business to be provided with a public service</td>
</tr>
<tr>
<td>8</td>
<td>required-documents</td>
<td>Documents required for a public service</td>
</tr>
<tr>
<td>9</td>
<td>procedure</td>
<td>Procedure that has to be followed</td>
</tr>
<tr>
<td>10</td>
<td>periodicity</td>
<td>How often is the service provided</td>
</tr>
<tr>
<td>11</td>
<td>time-to deliver</td>
<td>How long it takes to deliver the output of a service</td>
</tr>
<tr>
<td>12</td>
<td>cost-Info</td>
<td>Information about the cost of the public service</td>
</tr>
<tr>
<td>13</td>
<td>service-hours</td>
<td>Hours that the service is provided</td>
</tr>
<tr>
<td>14</td>
<td>employee-hints</td>
<td>Hints for the employee of the public authority</td>
</tr>
<tr>
<td>15</td>
<td>citizen-hints</td>
<td>Hints for the citizen / business of the public authority</td>
</tr>
<tr>
<td>16</td>
<td>public-authority-name</td>
<td>Name of the public authority</td>
</tr>
<tr>
<td>17</td>
<td>public authority</td>
<td>Department of the public authority</td>
</tr>
<tr>
<td></td>
<td>department</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>e-documents</td>
<td>Documents in electronic format</td>
</tr>
<tr>
<td>19</td>
<td>delivery-channel</td>
<td>Channel for the delivery of the service</td>
</tr>
<tr>
<td>20</td>
<td>cost</td>
<td>Numeric value of the cost of the service</td>
</tr>
<tr>
<td>21</td>
<td>contact-details</td>
<td>Details of the public servant to contact</td>
</tr>
<tr>
<td>22</td>
<td>service-code</td>
<td>Legal code of a service</td>
</tr>
<tr>
<td>23</td>
<td>automation-level</td>
<td>Automation level of the public service</td>
</tr>
<tr>
<td>24</td>
<td>public-authority-address</td>
<td>Address of the Public authority</td>
</tr>
<tr>
<td>25</td>
<td>state</td>
<td>State of the public service</td>
</tr>
<tr>
<td>26</td>
<td>service-name</td>
<td>Information related to implementation</td>
</tr>
</tbody>
</table>

### 4.3.5.1.3 GovML metadata vocabularies

Metadata used in terms of eGOV project enhanced and facilitated search and retrieval of Governmental resources across the Internet.

The core of the GovML metadata vocabulary is DC Element set and eGMS. Moreover Prism (http://www.prismstandard.org/) has been utilised too.

Mixing and matching of elements from different sets (e.g. DC, eGMS and PRISM) was achieved, using mechanisms available in RDF and XML.

### 4.3.5.1.4 Properties of GovML metadata vocabularies

Table 9. GovML metadata element set

<table>
<thead>
<tr>
<th>Seq</th>
<th>Element</th>
<th>Brief Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Title</td>
<td>A short name given to the resource</td>
</tr>
<tr>
<td>2</td>
<td>Creator</td>
<td>Creator of the content of the resource</td>
</tr>
<tr>
<td>3</td>
<td>Subject</td>
<td>Topic of the resource</td>
</tr>
<tr>
<td>Seq</td>
<td>Element</td>
<td>Brief Description</td>
</tr>
<tr>
<td>-----</td>
<td>--------------------</td>
<td>-------------------------------------------------------------</td>
</tr>
<tr>
<td>4</td>
<td>Publisher</td>
<td>Entity making the resource available</td>
</tr>
<tr>
<td>5</td>
<td>Date</td>
<td>Date of the creation / availability of the resource</td>
</tr>
<tr>
<td>6</td>
<td>Type</td>
<td>Nature of the content of the resource</td>
</tr>
<tr>
<td>7</td>
<td>Format</td>
<td>Format of the digital content of the resource</td>
</tr>
<tr>
<td>8</td>
<td>Language</td>
<td>Language of the content of the resource</td>
</tr>
<tr>
<td>9</td>
<td>Relation</td>
<td>A reference to a related resource</td>
</tr>
<tr>
<td>10</td>
<td>Coverage</td>
<td>The extent in terms of spatial location of the content of the resource</td>
</tr>
<tr>
<td>11</td>
<td>Audience</td>
<td>Group of people, businesses or public authorities the resource focuses on.</td>
</tr>
<tr>
<td>12</td>
<td>Has Translation</td>
<td>Reference to a resource in a translated version</td>
</tr>
</tbody>
</table>

### 4.3.5.1.5 Example of a GovML document

In Figure 10 an example of a GovML document describing the life event: “Getting married” is given.

```xml
<?xml version="1.0" encoding="UTF-8"?>
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.e-gov-project.org/GovMLSchema/ 1le:///C:/temp/GovMLSchema.xsd"
<identifier>ABC12345</identifier>
<languages>EN</language>
<title>Getting married</title>
<description>This life event concerns only adults</description>
<faq-list>
  <item>
    <question>Is there a possibility to get married online?</question>
    <answer>Yes. Visit the national governmental portal</answer>
  </item>
</faq-list>
<related-services>
  <item>
    <title>Issuing a birth certificate</title>
    <url>http://www.e-govproject.org/birth</url>
  </item>
  <item>
    <title>Online payment</title>
    <url>http://www.e-govproject.org/online_payment</url>
  </item>
</related-services>
<law>Law with number FRX:234</law>
</govml:GovML>
```

**Figure 10. Example of a GovML document**
4.3.5.2 A Dublin Core E-Government profile and model

The last project was conducted within the MultiMedia Information – Dublin Core (MMI-DC) Workshop of the European Committee of Standardization (CEN). The project started on September 2002, ended on September 2003, and resulted in two CEN Working Agreements (CWA), namely CWA 14860 “Dublin Core E-Government Application Profiles” [48] and CWA 14859 “Guidance on the Use of Metadata in E-Government” [49].

This section only presents the work relevant to the establishment of a DC-based E-Government Application Profile by harmonizing existing national initiatives, mainly in Member States.

It should be noted that a follow up project is currently running by CEN MMI-DC Workshop. The project started on January 2004, will finish on January 2005 and is entitled “Further development, guidance and promotion related to the EU E-Government Metadata Framework”. This project will culminate in two more CWA, namely “EU E-Government Metadata Framework” and “Guidance for the deployment of the EU E-Government Metadata Framework”.

4.3.5.2.1 General Rules

The establishment and maintenance of a metadata Application Profile (AP) for E-Government should be based on a set of rules. The proposed rules are:

1. **General Rules**
   
   1.1. The application profile will be based on Dublin Core.
   
   1.2. Additions to Dublin Core will be only considered if they:
       - have clear purpose
       - provide clear value and
       - do not significantly overlap with existing terms
   
   1.3. The application profile should allow harmonisation by allowing Member States to use their own labels for metadata terms (i.e. elements and refinements) but always using the centrally agreed name.
   
   1.4. All metadata terms in the application profile are considered as optional thus allowing Member States to determine the obligation of each metadata term according to their needs.

2. **First version and Updates**

   2.1. The first version of the E-Government Application Profile will be based on the metadata standards in Member States that are based on Dublin Core (i.e. UK, Denmark, Finland, Iceland and Ireland) while other relevant standards and initiatives will be also taken into account.

   2.2. All official changes of Dublin Core should be adopted immediately.

   2.3. For considering a new term, suitable documentation should be provided to indicate its purpose and value. For example, the documentation could be
provided in the form proposed by the methodology reported in CWA “Guidance on the Use of Metadata in E-Government”.

4.3.5.2.2 CEN Application Profile v.1.0

The following namespaces are assumed:

- \texttt{dc}: this contains version 1.1 of the Dublin Core Element Set
- \texttt{uk}: this contains version 2.0 of the e-GMS (http://www.govtalk.gov.uk/terms/)
- \texttt{dk}: this contains the Danish metadata standard
- \texttt{fi}: this contains the Finnish metadata standard
- \texttt{ic}: this contains the Icelandic metadata standard
- \texttt{cen}: this contains new terms of version 1.0 (http://www.cenorm.be/metadata/egov/terms)

The proposed CEN E-Government metadata AP consists of the following terms (besides those element refinements already in DC and national standards) as shown in Table 10.

Compared to DC, this AP includes only seven new elements: Accessibility, Disposal, Location, Preservation, Status, Mandate and MetaMetadata. Is summary, from the initial 19 extra elements in use in national standards, 2 are currently eliminated, 7 are retained as elements while the remaining 10 are either included as refinements or mapped to those elements retained.

It should be noted that no new metadata term has been included in this AP. The work included only reviewing national metadata standards and harmonizing them towards a common metadata standard.

4.3.5.2.3 Mapping to Existing Standards

4.3.6 presents the mapping of existing standards to the proposed CEN E-Government Application Profile (AP) version 1.0. The standards considered are those of UK, Denmark, Finland, Iceland, Ireland, Canada, Australia, New Zealand and the one proposed by the IDA MIREG project. This table only contains elements that do not map obviously i.e. elements with a name and definition that is not identical to the name and definition in the CEN E-Government AP version 1.0.

There are two issues to be noted. Firstly, although nine different standards were included in this mapping exercise, only 16 elements needed to be mapped. This suggests that although the proposed CEN E-Government AP is relatively small (only the standard DC terms plus seven new) it still includes the vast majority of terms in the national standards.

Secondly, for the 12 out of 16 elements the mapping is one to one thus it is assured that national standards can be easily adapted to comply with the new AP. From the remaining 4 elements, the 3 are not mapped while 1 is mapped into two refinements. It is noted that 2 out of the 3 elements that were not mapped are referring to Digital Signatures; an indication that these should be examined in a next version of the AP. The remaining element, namely “Function”, refers to the “business function of the
agency to which the resource or service relates”. This seems a valuable element in the content of public sector resources. Indeed, citizens are often concerned on the authority that owns the resource they access since this is related to their trust and confidence on the use of that resource.

Table 10. CEN E-Government metadata

<table>
<thead>
<tr>
<th>Element</th>
<th>Element Refinement(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>dc: Title</td>
<td></td>
</tr>
<tr>
<td>dc: Creator</td>
<td></td>
</tr>
<tr>
<td>dc: Subject</td>
<td></td>
</tr>
<tr>
<td>dc: Description</td>
<td></td>
</tr>
<tr>
<td>dc: Publisher</td>
<td></td>
</tr>
<tr>
<td>dc: Contributor</td>
<td></td>
</tr>
<tr>
<td>dc: Date</td>
<td></td>
</tr>
<tr>
<td>dc: Type</td>
<td>CEN: Aggregation, DocumentType</td>
</tr>
<tr>
<td>dc: Format</td>
<td></td>
</tr>
<tr>
<td>dc: Identifier</td>
<td></td>
</tr>
<tr>
<td>dc: Source</td>
<td></td>
</tr>
<tr>
<td>dc: Language</td>
<td></td>
</tr>
<tr>
<td>dc: Relation</td>
<td></td>
</tr>
<tr>
<td>dc: Coverage</td>
<td></td>
</tr>
<tr>
<td>dc: Rights</td>
<td>CEN: Publicity</td>
</tr>
<tr>
<td>dc: Audience</td>
<td>CEN: Receiver</td>
</tr>
<tr>
<td>UK: Accessibility</td>
<td></td>
</tr>
<tr>
<td>UK: Disposal</td>
<td></td>
</tr>
<tr>
<td>UK: Location</td>
<td></td>
</tr>
<tr>
<td>UK: Preservation</td>
<td></td>
</tr>
<tr>
<td>UK: Status</td>
<td>CEN: Version</td>
</tr>
<tr>
<td>UK: Mandate</td>
<td>CEN: Mandator</td>
</tr>
<tr>
<td>CEN: MetaMetadata</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CEN: Creator, dateCreated, dateModified</td>
</tr>
</tbody>
</table>

4.3.6 Conclusions and future work

Metadata may be used amongst others for the discovery and retrieval of governmental resources. As a result, an increasing number of governments worldwide establishes and promotes the use of metadata standards as an essential part of their E-Government
strategies. Many of these standards are based on the Dublin Core (DC) metadata initiative.

This section presented a number of metadata standards worldwide that are based on Dublin Core. It has been noted that the approach these governments are using is to adopt DC and further extent it according to their needs and requirements. As the needs and requirements across these governments are similar, it comes as no surprise that these standards are similar. Despite the similarities however, there are still differences between standards that in the future may harm interoperability between different countries. For this purpose, a number of initiatives and projects are working towards establishing and promoting a universal metadata standard.

Table 11. The mapping of existing standards to CEN E-Government Application Profile (AP)

<table>
<thead>
<tr>
<th>Standard</th>
<th>Metadata Term</th>
<th>CEN Metadata Term</th>
</tr>
</thead>
<tbody>
<tr>
<td>UK</td>
<td>Addressee</td>
<td>Audience.Receiver</td>
</tr>
<tr>
<td>UK</td>
<td>Aggregation</td>
<td>Type.Aggregation</td>
</tr>
<tr>
<td>UK</td>
<td>Digital Signature</td>
<td>-</td>
</tr>
<tr>
<td>DK, FI</td>
<td>Receiver</td>
<td>Audience.Receiver</td>
</tr>
<tr>
<td>DK</td>
<td>Process</td>
<td>Status</td>
</tr>
<tr>
<td>FI</td>
<td>DocumentType</td>
<td>Type.DocumentType</td>
</tr>
<tr>
<td>FI</td>
<td>Publicity</td>
<td>Rights.Publicity</td>
</tr>
<tr>
<td>FI</td>
<td>Publicity.securityClass</td>
<td>Rights.RightsSecurityClass</td>
</tr>
<tr>
<td>FI, IC, MIReG</td>
<td>Version</td>
<td>Status.Version</td>
</tr>
<tr>
<td>FI</td>
<td>Environment</td>
<td>Preservation.Environment</td>
</tr>
<tr>
<td>FI</td>
<td>Mandator</td>
<td>Mandate.Mandator</td>
</tr>
<tr>
<td>FI, AU, NZ</td>
<td>Availability</td>
<td>Rights.Publicity, Rights.Price</td>
</tr>
<tr>
<td>IC</td>
<td>Metadata</td>
<td>MetaMetadata</td>
</tr>
<tr>
<td>AU, NZ</td>
<td>Function</td>
<td>-</td>
</tr>
<tr>
<td>MIReG</td>
<td>Circulation List</td>
<td>Audience.Receiver</td>
</tr>
<tr>
<td>MIReG</td>
<td>Electronic Signature</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Certificate</td>
<td>-</td>
</tr>
</tbody>
</table>

4.4 Current Languages for service descriptions

In this section we first take a look at existing work. Currently, several Web Service-related languages are being discussed. They range from supporting sessions and transactions, quality of service issues, to publishing and subscribing to events. Considering the issue of actually invoking a Web Service in a more flexible way, two approaches are closely related, which we discuss in the following two subsections.
- WSDL (Web Service Description Language) – describes the syntactic information of a service (the operations it supports, the transport and messaging protocols on which it supports those operations, the network endpoint of the Web service);
- Ontology-based Web Service Description Languages – describe the syntactic as well as the semantic information. OWL-S and WSMO are the most salient initiatives to describe semantic web services.

4.4.1 WSDL

Web services need to be described. A service requestor needs to analyse a service for his requirements. Moreover, a web service needs to provide the following information:

- the operations it supports,
- the transport and messaging protocols on which it supports those operations,
- the network endpoint of the Web service.

WSDL stands for Web Services Description Language. It is an XML-based language for describing Web services. It specifies the location of the service and the operations (or methods) the service exposes. The main structure of a WSDL document that contains set of definitions to define a web service looks like it is shown in Figure 11.

![Web Service Description (WSDL)](image)

Figure 11. Web Service Description (WSDL)

The `<types>` element defines the data types that are used by the web service. For maximum platform neutrality, WSDL uses XML Schema syntax to define data types.

The `<message>` element defines the data elements of an operation. Each message can consist of one or more parts. The parts can be compared to the parameters of a function call in a traditional programming language.

The `<portType>` element is the most important WSDL element. It defines a web service, the operations that can be performed, and the messages that are involved. The
<portType> element can be compared to a function library (or a module, or a class) in a traditional programming language. WSDL defines four operation types:

- One-way - The operation can receive a message but will not return a response
- Request-response - The operation can receive a request and will return a response
- Solicit-response - The operation can send a request and will wait for a response
- Notification - The operation can send a message but will not wait for a response

The <binding> element defines the message format and protocol details for each port. It has two attributes - the name attribute and the type attribute. The name attribute (one can use any name he/she wants) defines the name of the binding, and the type attribute points to the port for the binding.

The simplified fraction of a WSDL document is shown in Figure 12. In this example the portType element defines "glossaryTerms" as the name of a port, and "getTerm" as the name of an operation. The "getTerm" operation has an input message called "getTermRequest" and an output message called "getTermResponse". The message element defines the parts of each message and the associated data types. Compared to traditional programming, glossaryTerms is a function library, "getTerm" is a function with "getTermRequest" as the input parameter and getTermResponse as the return parameter. The type attribute points in this case to the "glossaryTerms" port.

One important aspect is the composition of processes. Composition is the task of combining and linking existing Web Services and other components to create new processes. It adds value to the collection of services, by orchestrating them according to the requirement of the problem. However, WSDL allows us to capture the various methods but does not support constraints among those methods. Either too many methods will always be enabled or too few. However, WSDL’s functionality is required to specify the methods supported by a service. Recently, the Web Services Flow Language (WSFL) was proposed to describe compositions of services in the form of a workflow [5]. WSFL specifications tell us how different services ought to be invoked, e.g. in terms of ordering and parallelism among them, but they don’t tell us whether a particular service that is bound in the workflow will in fact deliver the right interactions. XLANG [6] fits in Microsoft’s BizTalk Server architecture. It describes the behaviour of a single Web service. XLANG as it stands today provides a notation similar in spirit to workflow languages. More information about languages for the composition of web services is given in section 5.

The main disadvantage of the WSDL language is that it describes only the syntactic information of a service. The elements of document types must be populated with correct values so that they are semantically correct and are interpreted correctly by the service requesters and providers. This requires the definition of a vocabulary that enumerates or describes valid element values. For example, one vocabulary can contain a list of product names or products that can be ordered from a manufacturer. Further examples are unit of measures as well as country codes. In the next section we elaborate on ontologies for describing web services that describe the syntactic as well as the semantic information.
Figure 12. A WSDL example

4.4.2 Ontology-based Web Service Description Languages

4.4.2.1 OWL-S

The Semantic Web [2] is an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation. This is realized by marking up Web content, its properties, and its relations, in a reasonably expressive markup language with a well-defined semantics. The Web Ontology Language (OWL) (see section 3.1.3) is a forthcoming W3C specification for such a language. OWL-S supersedes and extends the DAML-S proposal [15].

OWL-S\footnote{http://www.daml.org/services/owl-s/1.0/} is an OWL-based Web service ontology, which establishes a framework within which the web services may be described in this semantic web context. It supplies Web service providers with a core set of markup language constructs for describing the properties and capabilities of their Web services in unambiguous, computer-interpretable form. OWL-S markup of Web services will facilitate the automation of Web service tasks, including automated Web service discovery, execution, composition and interoperation.
OWL-S partitions a semantic description of a web service into three components: the service profile, process model and grounding. This is shown in Figure 13. The ServiceProfile describes what the service does by specifying the input and output types, preconditions and effects. The Process Model describes how the service works; each service is either an AtomicProcess that is executed directly or a CompositeProcess that is a combination of other subprocesses. The Grounding contains the details of how an agent can access a service by usually specifying a binding to a WSDL operation.

![Figure 13. Top level of the service ontology](image)

The class SERVICE provides an organizational point of reference for declaring Web services; one instance of SERVICE will exist for each distinct published service. The properties presents, describedBy, and supports are properties of SERVICE. The classes SERVICEPROFILE, SERVICEMODEL, and SERVICEGROUNDING are the respective ranges of those properties. Each instance of SERVICE will present a descendant class of SERVICEPROFILE, be describedBy a descendant class of SERVICEMODEL, and support a descendant class of SERVICEGROUNDING. The details of profiles, models, and groundings may vary widely from one type of service to another—that is, from one descendant class of SERVICE to another. But each of these three classes provides an essential type of information about the service.

Returning to the passport issuance example elaborated in section 2, the following description in the OWL-S would be created:

```xml
<owl:Class rdf:ID="PassportIssuance">
  <rdf:subClassOf rdf:resource="OWLS#Service"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#presents"/>
      <owl:hasValue rdf:resource="#PassportIssuanceProfile"/>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

Page 55 of 162
Instead of describing the service on a conceptual level one can also use an instance of the OWL-S concept “service” like in the example below:

```
<service:Service rdf:ID="PassportIssuance_ServiceInstance">
    <service:presents rdf:resource="&PassportIssuanceProfile;&PassportIssuance_ProfileInstance"/>
    <service:describedBy rdf:resource="&PassportIssuanceProcess;&PassportIssuance_ProcessModelInstance"/>
    <service:supports rdf:resource="&PassportIssuanceGrounding;&PassportIssuance_GroundingInstance"/>
</service:Service>
```

### 4.4.2.1 Service Profile

The service profile tells “what the service does”; that is, it gives the types of information needed by a service-seeking agent to determine whether the service meets its needs. In the rest of this subsection we explain properties defined in this ontology through the passport issuance service that is elaborated in section 2.

The Service Profile always contains references to the Service and the main process(es) of the ServiceModel like stated below:

```
<profile:PassportIssuance_Profile rdf:ID="PassportIssuance_ServiceInstance">
    <service:presentedBy rdf:resource="&PassportIssuanceService;"/>
</profile:PassportIssuance_Profile>
```
Some properties of the profile provide human-readable information that is unlikely to be automatically processed. These properties include serviceName, textDescription and contactInformation.

An essential component of the profile is the specification of what functionality the service provides and the specification of the conditions that must be satisfied for a successful result. In addition, the profile specifies what conditions result from the service, including the expected and unexpected results of the service activity. The OWL-S Profile represents two aspects of the functionality of the service: the information transformation (represented by inputs and outputs) and the state change produced by the execution of the service (represented by preconditions and effects). For example, to complete the passport issuance, a passport issuance service requires as input the ID number of a person, but also the precondition that he/she has corresponding citizenship. The result of the service is the output of a passport, and as effect the physical transfer of the passport from the office to the address of the person who required a passport. The following statements specify the statements mentioned above in OWL-S:

```xml
<profile:hasInput rdf:resource="&PassportIssuanceProcess;#Person_IDNumber"/>
<profile:hasPrecondition rdf:resource="&PassportIssuanceProcess;#Citizenship"/>
<profile:hasOutput rdf:resource="&PassportIssuanceProcess;#Passport_Document"/>
<profile:hasEffect rdf:resource="&PassportIssuanceProcess;#PassportShippedEffect"/>
```
Further, there are attributes that include the quality guarantees that are provided by the service, possible classification of the service, and additional parameters that the service may want to specify.

```
<profile:QualityRating rdf:ID="PassportIssuance_Rating">
    <profile:ratingName> Some Rating </profile:ratingName>
    <profile:rating rdf:resource="#GoodRating"/>
</profile:QualityRating>
```

### 4.4.2.1.2 Process Model

The service model tells “how the service works”; that is, it describes what happens when the service is carried out. For nontrivial services (those composed of several steps over time), this description may be used by a service-seeking agent in at least four different ways: (1) to perform a more in-depth analysis of whether the service meets its needs; (2) to compose service descriptions from multiple services to perform a specific task; (3) during the course of the service enactment, to coordinate the activities of the different participants; and (4) to monitor the execution of the service.

First of all the Process Model specifies all data types needed for the description of a service, for instance as input or output parameters. In order to do so one can use all mechanisms provided by OWL. To specify a concept of a credit card type one can state:

```
<owl:Class rdf:ID="CreditCardType">
    <owl:oneOf rdf:parseType="Collection">
        <CreditCardType rdf:ID="MasterCard"/>
        <CreditCardType rdf:ID="VISA"/>
        <CreditCardType rdf:ID="AmericanExpress"/>
    </owl:oneOf>
</owl:Class>
```

The main task a process model has to accomplish is the description of the processes themselves. The first task to specify has to be the one denoted in the Service Model within the `<service:describedBy>` property. The following describes the main process advertised in the service model and the process being responsible for the implementation of the process itself:

```
<process:ProcessModel rdf:ID="PassportIssuance_ProcessModelInstance">
    <service:describes rdf:resource="&PassportIssuanceService; PassportIssuance_ServiceInstance"/>
    <process:hasProcess rdf:resource="#PassportIssuance_Process"/>
</process:ProcessModel>
```
The statement above introduces a process called PassportIssuance_Process. These process is in general a composite process containing all steps needed to provide the functionality denoted at the service model.

Regarding our passport example shown in Figure 2 for the first two steps (Application and Verification) the concept description of the process might be:

```
<owl:Class rdf:ID="PassportIssuance_Process">
  <owl:subClassOf rdf:resource="&process;#CompositeProcess"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="&process;#composedOf"/>
      <owl:toClass>
        <owl:Class>
          <owl:intersectionOf rdf:parseType="owl:collection">
            <owl:Class rdf:about="&process;#Sequence"/>
            <owl:Restriction>
              <owl:onProperty rdf:resource="&process;#components"/>
              <owl:toClass>
                <owl:Class>
                  <process:listOfInstancesOf rdf:parseType="owl:collection">
                    <owl:Class rdf:about="#Application"/>
                    <owl:Class rdf:about="#Verification"/>
                  </process:listOfInstancesOf>
                </owl:Class>
              </owl:toClass>
            </owl:Restriction>
          </owl:intersectionOf>
        </owl:Class>
      </owl:toClass>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

This quite complex statement just describes that the process PassportIssuance_Process is a sequence of the processes Application and Verification meaning that the Verification process can be started after the completion of the Application process at the earliest.

Like mentioned before a process has input and output parameters, conditions and effects. This will be shortly demonstrated considering the atomic Application process. To simplify matters let’s assume that there are just one input document to prove the residence of the applicant – named ResidenceDocument - and one output document (VerificationDocument) containing all the applicant details collected in the Application process, e.g. the name, the date of birth, the address of the applicant etc.

```
<owl:Class rdf:ID=""Application">
  <owl:subClassOf rdf:resource="&process;#AtomicProcess"/>
</owl:Class>
```
The concepts ResidenceDocument and VerificationDocument may be modelled as subconcepts of a general concept Document.

The output parameter output_verificationDocument of the Application process is also the input parameter of the Verification process, since the Verification process has the task to verify the applicant’s details collected in the Application process. In order to map output parameters of one process to input parameters of another OWL-S provides a simple mechanism:

```
<process:sameValues rdf:parseType="Collection">
  <process:ValueOf
      process:atProcess="#Application"
      process:theParameter="#output_verificationDocument"/>
  <process:ValueOf
      process:atProcess="#Verification"
      process:theParameter="#input_verificationDocument"/>
</process:sameValues>
```

The parameter input_verificationDocument denotes here the input parameter of the Verification process.

One major problem with OWL-S is the lack of a standardized way to express conditions. Conditions in OWL-S are used to describe preconditions of a process execution or conditional outputs and effects, which allow different outputs/effects depending on the execution of the process. Furthermore, conditions are intended to describe the connection of consecutive processes.

Suppose having a process A with a boolean output, saying either true or false. Depending on the value of the output parameter we want to start either process B or process C. OWL-S provides an IF-THEN-ELSE mechanism for that purpose based on conditions.

People at the OWL-S conspiracy are working on the expression of conditions but till now there is no common sense how it might be done. One suggestion is the use of the
Semantic Web Rule Language (SWRL)\textsuperscript{32}, a combination of OWL and RuleML\textsuperscript{33} and reference mechanisms to exchange the values of the conditions among the processes. An import question for the OntoGov project might be how long it will take till a standardized mechanism in OWL-S will be provided.

4.4.2.1.3 Grounding

A service grounding (“grounding” for short) specifies the details of how an agent can access a service. Typically grounding will specify a communication protocol, message formats, and other service-specific details such as port numbers used in contacting the service. In addition, the grounding must specify, for each abstract type specified in the SERVICEMODEL, an unambiguous way of exchanging data elements of that type with the service (that is, the serialization techniques employed).

4.4.2.2 WSMO

WSMO-Standard\textsuperscript{34} is a meta-ontology for describing aspects of semantic web services, so that these web services can be automatically discovered and composed to satisfy some user’s goal, mediating between all kinds of possible mismatches. WSMO-Standard provides modelling constructs to describe web services, the ontologies they use, goals a user may have and different mediators that can solve integration problems.

WSMO-Standard allows different levels of modelling: firstly, it is possible to extend WSMO-Standard by adding one’s own constructs; secondly, one can decide to model elements with more, or less, detail. WSMO-Lite limits the elements that can be described, and does so for a very simple reason. The goal of WSMO-Lite is to provide a subset of WSMO-Standard that is meaningful (in the context of web service integration), but, for which, an execution environment is easily implementable. This execution environment would satisfy the goals of users by invoking web services; it would use WSMO-compliant definitions of these goals, these web services and the ontologies they are using. Having WSMO-Lite is very important, as it allows WSMO to 'bootstrap' itself into world-wide use: it allows easy implementation of a simple subset of WSMO, showing the viability and potential of WSMO, and thus attracting interested parties. Once such execution environments have been shown to work, the full potential of WSMO can be unleashed, and the execution environments could be extended to support WSMO-Standard.

The WSMO-Lite defines a web service in the following way:

\begin{verbatim}
webService[
  nonFunctionalProperties => nonFunctionalProperties
  usedMediator =>>> ooMediator
]
\end{verbatim}

\textsuperscript{32} http://www.daml.org/rules/proposal/
\textsuperscript{33} http://www.dfki.uni-kl.de/ruleml/
\textsuperscript{34} http://www.wsmo.org/
4.4.2.2.1 Non-functional properties

The non-functional properties of a web service have the following form:

(i) Title - a name given to the element;

(ii) Identifier - An unambiguous reference to the element within a given context. Recommended best practice is to identify the element by means of a string or number conforming to a formal identification system. Formal identification systems include but are not limited to the Uniform Resource Identifier (URI) (including the Uniform Resource Locator (URL)), the Digital Object Identifier (DOI) and the International Standard Book Number (ISBN).

4.4.2.2 Used Mediators

By importing ontologies, a web service can make use of concepts and relations defined elsewhere. A web service can import ontologies using ontology mediators. Building an ontology for some particular problem domain can be a rather cumbersome and complex task. One standard way to deal with the complexity is modularization. Imported ontologies allow a modular approach for ontology design. By importing other ontologies, one can make use of concepts and relations defined elsewhere. Nevertheless, when importing an arbitrary ontology, most likely some steps for aligning, merging and transforming imported ontologies have to be performed. For this reason and in line with the basic design principles underlying the WSMF, we use ontology mediators for importing ontologies.

4.4.2.2.3 Capability

The capability of a web service is described as:

```
capability[  
  nonFunctionalProperties=>nonFunctionalProperties  
  preconditions =>> simpleLogicalExpression  
  postconditions =>> simpleLogicalExpression  
  assumptions =>> simpleLogicalExpression  
  effects =>> simpleLogicalExpression ]
```

The non functional properties of a capability consist of the core properties where, in this case, an element in the core properties is equivalent to a capability. Pre-conditions describe what a web service expects for enabling it to provide its service. They define conditions over the input. Post-conditions describe what a web service returns in response to its input. They define the relation between the input and the output. Assumptions are similar to pre-conditions, however, also reference aspects of the state of the world beyond the actual input. Effects describe the state of the world after the execution of the service.
4.4.2.2.4 Interface

The properties of an interface in WSMO-Lite are non functional properties, errors and messages exchange and grounding. The idea is to allow only simple conversation (message passing without composition):

```
interface[
  nonFunctionalProperties => nonFunctionalProperties
  messageExchange => messageExchangePattern ]
```

Message exchange consists of a Message Exchange Patterns (MEP). MEPs model stimuli-response patterns; they define the sequence and the cardinality of the multiple messages exchanged.

4.4.2.3 Discussion

In this section we examined whether it is possible to use any of the existing ontologies for modelling web services for achieving the objectives of the OntoGov project.

OWL-S and WSMO are the most salient initiatives to describe semantic web services. They aim at describing the various aspects of services in order to enable the automation of Web Services discovery, composition, interoperability and invocation.

Both of the proposed approaches focus mostly on the service profile in order to support better discovery of services. Location of services is based on the semantic match between a declarative description of the service being sought, and a description of the service being offered. However, for the E-Government domain the most important part of the service description is the process model. The WSMO ontology does not contain the process model. On the other hand, the OWL-S ontology includes the process ontology. However, this ontology cannot be reused due to following reasons:

- Problems with A-Box reasoning: E-Government services can be described at the conceptual level (to support reusability) as well as at the instance level. For example, the generic service for the passport issuance has to be described at the conceptual level. On the other hand, the natural way of modelling each specialization of this service (e.g. service describing the passport issuance for the children) is using instances. However, since only reasoning in T-Box is supported, the discovery of services described using instances can be problematic;

- Formalism for expressing conditions is not defined – There is no official specification how conditions (i.e. rules and logical expressions) should be expressed in OWL-S. All examples contain the conditions that are only stubs. The modelling of services in the E-Government domain would not work without the introduction of conditions. They are needed for the preconditions, postconditions as well as for the IF-THEN control construct;

35 In the latest OWL-S version several extensions for incorporating rules in OWL-S are mentioned. The authors mention DRS and SWRL as potential candidates for a formal language for conditions in the context of OWL-S [60].
- Meta-modelling - In OWL Lite and OWL DL an individual (i.e. instance) can never be at the same time a class: classes and individuals form disjoint domains (as do properties and data values). OWL Full allows the freedom of RDF Schema: a class may act as an instance of another (meta)class. This relaxation violates the constraints of Description Logic reasoners. The E-Government domain does not enforce a strict separation of classes and individuals. On contrary, it enforces the usage of an entity as a concept and as an instance in the same time as needed;

- Use-cases - The next problem is there is no realistic use case that is modelled and tested. All use cases are too simple. They do not take into account necessity of rules and logic in OWL-S. Only the experience from the modelling real services can confirm the quality and the usability of the OWL-S ontology. For example, most of the composite services contain only of a sequence of other services. In the E-Government domain there is an explicit need for more complex composition that includes the conditional connection between services, i.e. IF-THEN control construct.

Summarizing the points above, a semantic service description should be allowed to make more detailed logical statements about the service, its parameters, its results, and its side effects. The terminology to be used must definitely be a domain ontology, which is linked to local data structures. Therefore, none of the existing ontologies for the semantic web service description can fulfil all the E-Government specific requirements and offer all the required capabilities. For this reason, the optimal solution was to develop a new ontology that could fulfil all these requirements. Such an ontology is introduced in section 6.3.1.1.

---

36 The main reason for having the OWL DL sublanguage is that tool builders have developed powerful reasoning systems, which support ontologies constrained by the restrictions required for OWL DL.
5 Business process modelling

Everything that looks like a series of steps tends to be labelled a business process. A business process is the description of steps needed to carry out a business activity regardless of the systems involved. They provide a high level view of the steps involved and can be used to model, benchmark and document existing or future designs. An executable business process is a kind of business process whose lifecycle is controlled by one or a combination of business process management systems (BPMS). A business process is long running. Its execution is not limited to minutes or hours like the session of a web-based application, it rather spans days, months, or years. An executable business Process relies on specific interactions between users, systems, and business partners, which it ties together. This system provides all the facilities and services necessary for design and execution, and mediates the integration with its environment.

A business process activity (task) represents a short-lived interaction between users or, in certain cases, systems. A business process activity can be viewed as one step in an executable business process. A typical example is a user browsing a catalogue and filling a shopping cart. Once the user is finished, he or she pushes the checkout button, which in turn completes the activity. The proper information is passed to a business process management system as part of a completion message.

Business process modelling languages (such as BPML, XLANG or WSFL) are used to describe executable business processes. In this section we discuss how these languages meet the requirements of the E-Government services. The most important requirement is the capability of a language for composition. We distinguish two types of composition:

- Composition in the part-of sense, i.e. a larger part encapsulates services and exposes itself as a service;
- Composition in the sequencing sense, i.e. definition of the invocation order of services.

In the rest of this section we define business processes and analyse several business process modelling languages with respect to their capabilities to represent composition. Then we make the comparison between these languages and OWL-S ontology that is described in the section 4.4.2.1. Finally, we discuss tools for the business process modelling.

5.1 Business Process definition

There are plenty of definitions what Business Processes are. For the beginning, let us quote from Malhotra [18]: Davenport & Short [19] define a business process as "a set of logically related tasks performed to achieve a defined business outcome." A process is "a structured, measured set of activities designed to produce a specified output for a particular customer or market. It implies a strong emphasis on how work is done within an organization" [20]. In their view processes have two important characteristics:

(i) They have customers (internal or external),
(ii) They cross organizational boundaries, i.e., they occur across or between organizational subunits.

The Concise Oxford Dictionary definition is:

“…a course of action or a procedure…a series of stages in manufacture or some other operation…”

Butler Group [21] defines a business process as a series of tasks or activities to achieve a given purpose or goal, which can be completed either in sequence or in parallel, by people or systems, either inside or outside an organisation. The tasks are pre-defined, and the process can be repeated. A process can be continuous, in that it is an ongoing process such as inventory replenishment, or it can be discrete, with a definite start and finish, such as the development of a new product, or the discharge of a cured patient from a hospital.

Business Process Management (BPM) is the planned and coordinated use of suitable methodological and technological tools for planning, coordinating, optimising, controlling and monitoring of business process enactment in a given organization, taking into account the business environment of this organization and the interrelationships of the co-existing business processes. Often it aims at improved quality of products and services, more efficient performance of the organization (with respect to process throughput time, costs, ...) etc. Radical process optimisation efforts are normally labelled Business Process Reengineering (BPR).

Business Process Modelling is a central element of BPM. Figure 14 (taken from [22]) shows the basic steps of BPM activities with the result of Process Modelling, the Process Definition, at its heart.

Process modelling objectives can be classified into five categories [23]:

1. Facilitate human understanding and communication
   - Represent process in a form understandable by humans
• Enable communication about and agreement on processes
• Formalize the process so that people can work together more effectively
• Provide sufficient information to allow an individual or team to perform the intended process
• Form a basis for training the intended process

(2) Support process improvement
• Identify all the necessary components of a high-yield development of maintenance process
• Compare alternative processes
• Estimate the impact of potential changes without first putting them into actual practice
• Assist in the selection of incorporation of technology into a process
• Support managed evolution of a process

(3) Support process management
• Monitor, manage and coordinate the process
• Provide a basis for process measurement, such as definition of measurement points within the context of a specific process

(4) Automate process guidance
• Provide guidance, suggestions, and reference material to facilitate human performance of the intended process
• Retain reusable process representations in a repository

(5) Automate process execution support
• Automate sub-processes of a process
• Support cooperative work among individuals and teams by automating the process
• Automatically collect measurement data reflecting actual experience with a process
• Enforce rules to ensure process integrity

Arguably, more relevant for OntoGov are items (1), (4), and (5) of the list above. Section 5.2 focuses on languages for modelling real life business processes for human understanding (1). In section 5.3 we address goals (4) and (5) by introducing protocols for service coordination and languages for service composition. Consequently, we will not examine approaches and techniques that have been developed primarily for business management and improvement (items 2 and 3).

5.2 Business Process Modelling Languages

The purpose of the business process modelling language is to provide a means to represent a process in a formalised way which is understandable by humans. This section introduces exemplarily two common concepts out of the amount of available languages.
5.2.1 BPML

Business Process Modelling Language (BPML) is a meta language for the modelling of business processes. Its open standards were defined by the Business Process Modelling Initiative (www.bpmi.org). It provides an abstract execution model for collaborative and business processes, and is based on the concept of a transactional finite state machine. In BPML, e-business processes are considered to be constructed from a common public interface with as many private implementations as there are process participants. This means that the public interface of BPML processes can be distributed as electronic ebXML business processes or RossetaNet Partner interface Processes, regardless of their implementations.

BPML processes can be described in a specific business process modelling language, which is layered on top of the extensible BPMLXML schema. Business processes are represented as the interleaving of control flow, data flow, and event flow, while orthogonal design capabilities are added for business rules, security roles, and transaction contexts. Explicit support is offered for synchronous and asynchronous distributed transactions, enabling it to be used as an execution model for embedding existing applications within e-business processes as process components.

The following listing shows a specific path of the example process “passport issuing”, encoded in the BPML. It refers to Figure 24, which is a more detailed elaboration of the process described in section 2 of this document.

The activities, which are not touched by this path, are hidden in the BPML code (+).

```
<?xml version="1.0" encoding="UTF-8" ?>
<package>
  <process name="issue_of_passport">
    <sequence>
      <action name="apply_for_passport"/>
      <switch>
        <case>
          <condition>requirements_fulfilled==yes</condition>
          <switch>
            <case>
              <action name="check_foreign_implications"/>
            </case>
          </switch>
        </case>
        <case>
          <condition>true</condition>
          <action name="check_foreign_implications"/>
        </case>
        <switch>
          <case>
            <condition>conditions_satisfied==yes</condition>
            <sequence>
              <action name="prepare_passport"/>
              <action name="send_passport_to_applicant"/>
            </sequence>
            </case>
          </switch>
        </sequence>
    </switch>
  </process>
</package>
```
5.2.2 IDEF Techniques (IDEF0, IDEF3)

The IDEF family of modelling techniques was developed as a set of notational formalisms for representing and modelling process and data structures in an integrated fashion. The IDEF suite consists of a number of independent techniques, the most well known being IDEF0 (Function Modelling), IDEF1x (Data Modelling), and IDEF3 (Process Description Capture). In this section we will describe IDEF0 and IDEF3 since they relate primarily to business process modelling.

The IDEF0 method is designed to model the decisions, actions, and activities of an organisation or other system and, as such, it is targeted mostly towards the functional modelling perspective. As a communication tool, IDEF0 aims at enhanced domain expert involvement and consensus decision-making through simplified graphical devices. Perhaps the main strength of IDEF0 is its simplicity, as it uses only one notational construct, called the ICOM (Input-Control-Output-Mechanism, see Figure 15). IDEF0 supports process modelling by progressively decomposing higher-level ICOMs into more detailed models that depict the hierarchical decomposition of activities.

Despite its advantages, IDEF0 presents a number of limitations that may render the technique unsuitable for process analysis. More specifically, IDEF0 models are static diagrams with no explicit or even implicit representation of time. Even the sequence of ICOMs is not meant to depict the temporal relations between activities. As such, IDEF0 models cannot represent the behavioural or informational modelling perspectives.

To overcome some of the limitations of IDEF0 models, IDEF3 has been developed. IDEF3 describes processes as ordered sequences of events or activities. As such, IDEF3 is a scenario-driven process flow modelling technique, based on the direct capture of precedence and causality relations between situations and events. The goal of an IDEF3 model is to provide a structured method for expressing the domain experts’ knowledge about how a particular system or organisation works (as opposed to IDEF0, which is mainly concerned with what activities the organisation performs).

IDEF3 makes use of two complementary diagrammatic representations of process models. Process Flow Diagrams (Figure 16) depict the flow of activities within a
process, while Object State Transition Diagrams (Figure 17) represent the different states of entities as they flow through the process.

![Figure 16. IDEF3 Example (Process Flow Diagram)](image1)

![Figure 17. IDEF3 Example (Object State Transition Diagram)](image2)

5.3 Service Orchestration Languages and Protocols

This section introduces emerging languages and protocols for interoperation between services. In the first part it refers to BPEL4WS which is a formal specification for business processes using web services. The usage of BPEL4WS is demonstrated in a detailed example from the E-Government domain. The second part deals with the concurrent WSCI specification and the last part contains a description of ebXML and RosettaNet.

5.3.1 BPEL4WS

In July 2002, BEA, IBM, and Microsoft released a trio of specifications designed to support business transactions over Web services. BPEL4WS represents the uniting of two previously competing standards: Web Services Flow Language (WSFL) from IBM, and Microsoft's XML business process language in BizTalk Server (XLANG).
These specifications, BPEL4WS, WS-Transaction, and WS-Coordination, form the bedrock for reliably choreographing Web services-based applications, providing business process management, transactional integrity, and generic coordination facilities, respectively.

The BPEL4WS model is built on a number of layers, each one building on the facilities of the previous. Figure 18 shows the fundamental components of the BPEL4WS architecture, which consists of the following:

![Figure 18. BPEL4WS Logical View](image)

1. A means of capturing enterprise interdependencies with partners and associated service links

To create workflows that span enterprises, we must understand how those enterprises are related. BPEL4WS provides a means of capturing the roles played by business partners in a Web services-based workflow through service linking, partners, and service references.

Figure 19 shows the relationship between service links, partners, and service references. Service links are the most abstract relationship supported in BPEL4WS, and link two parties by specifying the roles of each party and the (abstract) interface that each provides. serviceLinkType definitions can either be part of a service's WSDL interface, or defined separately and referenced by the WSDL. Embedding this definition directly in a WSDL description leverages WSDL's extensibility mechanism, allowing serviceLinkType elements to become a direct child of the wsdl:definitions element.

The actual content of a serviceLinkType is straightforward. It usually defines a link between two services, qualified by the targetNamespace of the WSDL document; and then exposes that relationship as two roles. In some cases, a serviceLinkType may specify a single role, which indicates that the workflow is willing to bind to any other service, without placing any requirements on that service.

A BPEL4WS partner refines a serviceLinkType declaration by defining the roles played by actual partners at the endpoints of the relationship. A partner is declared within the workflow script because it forms part of the behavior of that workflow. Partnerships only make sense within the scope of the workflow where business partners interact. Inside the partners element we have individual partner declarations that specify the role of our enterprise and its partners on a per-serviceLinkType basis.
The final step in cementing our business interrelationships is to specify the network locations of our partners so that we can discover and consume their Web services.

![Diagram of abstract and concrete enterprise relationships]

**Figure 19. Abstract and concrete enterprise relationships**

2. **Message correlation layer that ties together messages and specific workflow instances**

Once we've captured the relationships between our enterprise and its partners, we can begin to exchange messages through the conduits defined by those relationships. Whether we are dealing with an invoice or a dispatch note, there is often a field or set of fields within that note that can be used to unambiguously differentiate that note from piles of other similar looking ones. For instance, an invoice number is usually used in correspondence rather than the date and address of the sender of the invoice since it is both simpler and more likely to resolve to a unique result. This notion of "distinguished" data is supported through message properties. Put simply, a message property is a unique name (within the workflow) that has a specific type from XML Schema (e.g., xs:positive Integer) and whose name has significance to the workflow.

Having a friendly name and type information for our property is akin to having object references in traditional programming languages. However, just like object references need to be bound to objects before we can use them, we need to bind properties to values before workflow scripts can access those values. In BPEL4WS we have a way of binding typed friendly names to values that we can use within our workflows - *property aliases*. A property alias binds the value of a property to the value of an element in a message using an XPath query. For instance, we may be interested in the invoice number from a purchase order and want to expose that value to the workflow.

Once properties have been defined, they can be used to correlate messages. Using a property like an invoice number allows the underlying BPEL4WS implementation to route messages to particular workflow instances at the application level without relying on sophisticated conversational transports to manage that mapping. As we shall see, the BPEL4WS communication activities encapsulate this behavior to further simplify matters.

3. **State management features to maintain, update, and interrogate parts of process state as a workflow progresses**
In dealing with Web services-based workflows we encounter a significant philosophical difference between the two technologies: workflows are inherently stateful applications, whereas Web services are inherently stateless. Of course many Web services do actually maintain state between invocations, but do so in a proprietary manner in databases, files, statically allocated program variables, and so on, all of which requires programmer effort and is likely to be inaccessible to the business analyst. BPEL4WS has abstracted these proprietary approaches and replaced them with a generic state management facility based on containers.

A BPEL4WS container is a typed data structure that stores messages associated with a workflow instance. The underlying notion of containers is that in a workflow the state of the application is simply a function of the messages that have been exchanged. Containers begin their lives uninitialized, and are populated over time by the arrival of messages, or computations being executed that populate them.

4. Scopes where individual activities (workflow stages) are composed to form actual algorithmic workflows

Communication Activities

BPEL4WS defines a set of communication activities that deal with the sending and receiving of messages so that a workflow process instance can communicate with partners' Web services. BPEL4WS provides three activities - invoke, receive, and reply - each of which handles a different type of interaction between partners in a workflow.

The invoke activity allows a workflow instance to call a synchronous or asynchronous operation on a remote Web service. An asynchronous one-way operation is the simplest form of invoke since it only requires a single input container to send messages.

Web service operations are exposed to the outside world by a receive activity. The receive activity is the workflow entity that a WSDL operation maps onto. A receive activity is blocking, which does not allow the workflow graph it precedes to progress until the messages it requires have been received. Such activities may be used to trigger the creation of a new workflow instance in response to an incoming message, or they may deal with the message delivery to an existing instance.

A reply activity sends synchronous responses to messages received through a receive activity. Correlation between a receive and a reply is handled by the underlying BPEL4WS implementation.

Activities

In order to execute a process, we need a means of describing its behaviour. We have to understand the features that the workflow language provides to manipulate data, iterate, call external functions, and so on; and how to compose these primitives into meaningful workflows. To support this, the BPEL4WS specification defines a number of fundamental activities that are the basic building blocks of the workflow. It is beyond the scope of this article to look at every facet of each language construct defined by BPEL4WS, but we will give you an idea of what kinds of things are possible.
BPEL4WS control flow activities are responsible for serializing and parallelizing activities, choosing from alternative paths in a workflow, iterating commands, and so on.

Parallelizing activities that have no dependencies is achieved by enclosing the parallel activities within a flow element. For example, a customer's computing system, which initiated a hotel reservation, may also have been organizing flights and car rentals simultaneously. If we assume these activities are independent, we can execute them in parallel with a flow activity.

A *scope* is a means of explicitly packaging activities together so that they can share common error handling and compensation routines. The full structure for a scope is shown in Figure 20 and consists of a set of optional fault handlers, a single optional compensation handler, and the primary activity of the scope, which defines its behaviour.

![Figure 20. BPEL4WS Scope Structure](image)

### 5.3.1.1 Passport Issuance Example in BPEL4WS

BPEL4WS provides a construct to compose web services to business flows. In this section we create a web service oriented view of the passport issuance process, described in section 2. The visual representation in Figure 21 illustrates that the service can be considered as a composite process of the following web services:

- **Verifying:** For verifying the application, the data of the applicant is sent to the web service of the responsible authority (e.g. municipality). The back office decides in a formal check whether the requirements are fulfilled and the next service can be activated or the process should be stopped at this point with a
message to the applicant. Every request and the result of the verification are stored in the back office system.

- Producing: The next web service is held by the authority that is responsible for producing the passport. The web service is initiating a new job in the system, which is holding the work lists of the employees. A job contains two activities: printing and sending the passport.

- Approving: The last web service in this sequence is executed by the first authority that already has verified the application. It requests an acknowledgement of the responsible official and causes an entry in the database of the back office system. After the acknowledgement, the approval is sent to the applicant.

The visual representation is based on the notification in [42]. In order to provide a better overview of the functionality, a few elements are either not shown or simplified in the diagram.

![Figure 21. BPEL process for passport application](image-url)

This graphical representation refers to a textual BPEL4WS-description of the process in XML. The root of the BPEL document is the `<process>` element. According to the passport issuance example, the following description in BPEL would be needed:

```xml
<process name="PassportIssuance"
         targetNamespace="http://page.gov/ws-bp/passport"
         xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/
         xmlns:ns1="http://egov.org/wsd/def/bindings">
  
  …

</process>
```
The `<process>` element has an attribute “name” with the value “PassportIssuance”. Each Process collaborates with different partners. These Partners can be clients as well as web services that are providing their services to the whole process. Partners are defined by a `<partnerLink>`, a corresponding `<partnerLinkType>` and roles.

```xml
<partnerLinks>
  <partnerLink name="applicant"
    partnerLinkType="ns1:applicantLT"
    myRole="applicant"/>
  <partnerLink name="verifying"
    partnerLinkType="ns2:invoicingLT"
    myRole="verificationRequester"
    partnerRole="verificationService"/>
  ...
</partnerLinks>
```

Note that the construct `<partnerLinkType>` and the roles are referring to definitions in the WSDL document. The structure of the WSDL document is discussed in section 4.4.1.

The example process uses the following constructs for the flow definition:

- sequence
- receive
- invoke
- switch
- reply

The `<sequence>` construct is used to define a collection of activities to be performed sequentially in the order they are listed. In the passport example all activities are embedded by a `<sequence>` element. The first listed activity has the ability to receive information. The `<receive>` construct as well as the `<reply>` construct are often used for communication with the client. The following listing shows the first activity (receive) of a sequence in the passport example.

```xml
<sequence>
  <receive partnerLink="applicant"
    portType="ns1:applicationService"
    operation="applicationRequest"
    variable="applicationDataRequest">
    ...
  </receive>
</sequence>
```
<invoke> is used to call a web service in order to receive a specific information. In the example we use <invoke> to access the web services “verifying”, “producing” and “approving”. The <invoke> element has the following structure:

```
<sequence>
  <invoke partnerLink="approving"
    portType="ns4:approvingApplication"
    operation="sendApprove"
    inputVariable="applicationInfo">
  </invoke>

  ...
</sequence>
```

The <switch> construct allows choosing a specific branch of activities. Each branch has a condition. Occasionally there is a branch with the condition “otherwise”. This branch will be executed, if no other branch matches the given conditions.

In our example we use this construct after the web service “verifying” has been executed to decide whether the process goes on with the production of the passport and the approval or it stops and returns a message to the applicant.

```
<sequence>
  <switch>
    <case condition= "ns2:getVerification(verified) = true">
      <sequence>…</sequence>
    </case>
    <otherwise>
      <sequence>…</sequence>
    </otherwise>
  </switch>

  ...
</sequence>
```

Finally we used a <reply> construct in our example. This element allows the business process to reply to a message that was received through a <receive> element. The combination of a <receive> and a <reply> forms a request-response operation.

```
<sequence>
  ...
  <reply partnerLink="applicant">
    portType="ns1:applicationService"
    operation="applicationRequest"
  </reply>
</sequence>
```
This passport issuance example illustrates the use of some BPEL constructs, but it does not cover the whole specification.

Further constructs, such as variables, fault handlers or other activity types (assign, throw, terminate, etc.) are missing in this description. The specification with detailed information to all elements of BPEL4WS is provided in [43].

5.3.2 WSCI

The Web Service Choreography Interface (WSCI) was announced in August 2002 as a W3C note, promoted by Sun, SAP, BEA and Intalio [44]. The specification defines an XML-based language for web service interoperability. WSCI works with the Web Service Description Language (WSDL) and other service definition languages. Therefore it can be viewed as a layer on top of the existing web service stack (UDDI, WSDL, SOAP). Particularly it addresses the problem that current web service definitions do neither define mechanisms for choreography nor other relationships between atomic services.

WSCI supports message correlation, sequencing rules, exception handling, transactions, and dynamic collaboration. The specification does not address to define processes such as b2b services between different companies. A single WSCI description only defines the participation of one partner in a message exchange. Thus, a process would include a set of WSCI interfaces, one for each partner in the interaction. The specification contains Information on how to interact with a certain service. Moreover it enables anticipation of a client by describing the expecting behaviour of the web service at any point of the process. In other words, WSCI defines the interface of a web service dynamically in relation to a particular process. Thus it enables the developer to abstract from the implementation and to focus on the role of the web service.

The following simple example of a WSCI interface shows a purchasing process which contains two sequential activities, “Receive Order” and “Confirm”. Each activity maps to a WSDL portType, and a correlation is established between the two steps. Note that this is the WSCI document from the perspective of the Agent. There would also be WSCI files for the buyer and the supplier in the process.

```
<process name="Purchase" instantiation="message">
  <sequence>
    <action name="ReceiveOrder" role="Agent" operation="tns:Order">
    </action>
    <action name="Confirm" role="Agent" operation="tns:Confirm">
      <correlate correlation="tns:ordered"/>
      <call process="tns:Purchase"/>
    </action>
  </sequence>
</process>
```

WSCI defines an <action> tag for specifying a basic request or response message. Each activity specifies the WSDL operation involved and the role being played by the participant. External services can then be invoked through the <call> tag. A wide
variety of structured activities are supported, including sequential and parallel processing and condition looping. WSCI also introduces an <all> activity, used to indicate that the specific actions have to be performed, but not in any particular order. The example and the explanation are further described in [44].

BPEL4WS and WSCI address quite the same scope and do meet the technical requirements for a persistent web service interoperation. Both also have strong industrial partners which are pushing the standardization process of their own specification. According to [45] it seems that the industry appears to be embracing the BPEL4WS initiative. As in [46] described, it is still an open issue, how WSCI relates to competing specifications such as BPEL4WS.

Also possible is a merging approach, with BPEL4WS taking the lead in the efforts. Vendor backing and the development of tools will also support the adoption, taken by the software industry.

5.3.3 ebXML

The objective of Electronic Business XML (ebXML) is to provide an open XML-based infrastructure, enabling the global use of electronic business information in a interoperable, secure, and consistent manner by all parties. The United Nations Center for Trade Facilitation and Electronic Business and the Organization for the Advancement of Structural Information Standards sponsor ebXML. It is a global electronic business standard, which defines a framework for electronic business, enabling businesses to find each other and conduct business over the Internet, based on XML messages.

This business is carried out in the context of standard business processes that are governed by standard or mutually negotiated partner agreement. In addition to conducting trading relationships, businesses can also communicate data in common terms, and define and register business processes.

EbXML is repository-based. The repository contains industry defined Business Processes and Scenarios that are commonly applicable to most business transactions. Companies can choose to extend these processes and also add scenarios of their own. The repository also contains profiles for businesses that have already registered themselves for performing ebXML transactions with other trading partners. There are three steps to be carried out by any company wishing to trade using the ebXML standards. These are: request information, implement ebXML system, and publish business profile.

5.3.4 RosettaNet

RosettaNet is a self funded, non-profit organization, comprising a consortium of more than 400 major IT, Electronic Components and Semiconductor Manufacturing companies, who are working to create and implement industry-wide, open e-business process standards. The organisation was founded in June 1998, by 40 leading IT companies, its standards form a common e-business language, aligning processes between supply chain partners on a global basis. The organization is named after the Rosetta Stone, which was carved with the same message in three languages and led to an understanding of hieroglyphics. RosettaNet believes that it is breaking language
barriers in creating a common language for the electronic sharing of business information.

RosettaNet’s standards have been developed to provide an open- non-proprietary solution that comprises data dictionaries, an implementation framework, and business message schemas and process specifications, which standardizes e-business, the standards are available free of charge form RosettaNet’s Web site at www.rosettanet.org, and include the RosettaNet Business Dictionary, RosettaNet Technical Dictionary, and RosettaNet Implementation Framework (RNIF).

RosettaNet Dictionaries provide a global terminology designed to reduce confusion in the procurement process, due to each company’s uniquely defined terminology. The Business Dictionary provides properties for defining business transactions between partners, and the Technical Dictionary offers properties for defining products and services. The RNIF uses XML to specify information exchange between trading partner servers, and encompasses the transport, routing, packaging, security, signals, and agreements.

5.3.5 Comparison

In this section we compare the process model of the OWL-S ontology language with the business process modelling languages.

The Semantic Web community aims at providing machine understandable meaning to resources on the Web [9]. Consequently, the lack of semantics in solutions like BPEL4WS shows that one should also consider the respective language out of the Semantic Web area. The Web Ontology Languages for Services (OWL-S) (see section 4.4.2) is the language of choice if the resources to be described are services [13].

The language defines a top-level ontology for services. This ontology aims at formalizing commonly used service concepts such as ”effect”, ”input”, or ”process”. The three main components are the services profile, i.e. what the service does, the services model, i.e. how it works, and the grounding describing how to access the service.

We pick up the stock quote example presented in [8]. The parameter and return values are described using the terminology ”ticker” and ”quote” from a financial ontology. In addition to this, a precondition and effect are specified in that the user’s account must be valid and will be charged by the operation.

The main operation supported by this kind of tagging is the so-called semantic matching which allows a more specific concept to be associated with a more general one via the subsumption hierarchy. This is used to locate a service by matching the actual and requested service types. Semantic matching is also applied when comparing the service’s capabilities. A desired output, for instance a ”quote”, is satisfied by a service yielding a ”detailed:quote”, if a quote subsumes the detailed quote concept.

Critique of OWL-S

OWL-S is definitely striving for a completely automated framework in which services can be composed and invoked by intelligent software agents on the fly. However,
there are some shortcomings. The semantic matching along concept hierarchies is not enough. The key issues described in the following points need to be addressed.

1. **Data-structure Mediation** - Clearly, the data-structures of client and server need to be aligned. This is a classical data integration problem in which domain ontologies are an essential tool. OWL-S is currently lacking this aspect.

2. **Rich Annotation** - If a domain ontology is used to mediate between client and server, such an ontology should also be used for describing parameters, return types, preconditions, and the effects of a service. This needs to be done, not just by referencing a certain concept, but also via some rule or constraint mechanism, which allows much more flexibility.

3. **Result Processing** - An important point missing is what to do with the service result. BPEL4WS allows defining the dataflow from one service to the next. Consequently, an engine processing such a workflow knows in which way an intermediate result is supposed to be used, i.e. as a parameter for the next service call. In a more automated environment, an agent would need some knowledge, not only about the ontological concept the result refers to, but what to actually do with it.

4. **Parameter Relationships** - Currently, OWL-S associates the parameters and return types of a service with concepts of an ontology. However, no knowledge about their relationships is provided. Consider the service int getPassportID (String ID). OWL-S would tag the parameter ID as a ID code and the return type as a passport ID value. This makes sense and holds much more knowledge than the primitive data types alone. However, the information that the passport refers to a person identified by the ID code is not represented. This is quite clear to a human programmer, though not necessarily to a software agent.

5. **Hidden Assumptions** - Besides the relation of result and parameter, quite often there are more hidden assumptions that are encoded in the method name. Consider the example above. A human will know, that the method yields the current temperature. Again, this needs to be made explicit for a software agent.

Summarizing the points above, OWL-S provides value in that it supplies a standard top-level service ontology and that various tasks such as service locations and service capability matching are supported by class subsumption axioms. However, this is not enough. A semantic service description should be allowed to make more detailed logical statements about the service, its parameters, its results, and its side effects. The terminology to be used must definitely be a domain ontology, which is linked to local data structures.

### 5.4 Tools for the Business Process Modelling

Business process modelling tools are widely used by companies and organisations to describe, maintain and optimise business flows. The Workflow Management Coalition (WFMC) has – among others – compiled a workflow reference model ([51]), a glossary ([52]) and a meta model for the interchange of process data ([53], see also [54] for an overview of process modelling standards).
There are a large number of tools on the market which are mostly in accordance with the WFMC standards – although they use different modelling approaches (see [55] for an overview of the functionalities of the various products). In this section two products are introduced in more details:

- ARIS from IDS Scheer has been chosen because it is one of the leading software products in business process modelling.
- ADONIS from BOC is well-known for its meta-modelling concept. This approach allows adapting objects and methods for specific organisations or application areas.

The OntoGov process modelling component aims to adopt concepts of user guidance from these products. Moreover OntoGov uses meta-models (meta-ontologies) to describe the main primitives and therefore has some relations to the concepts, used in ADONIS. In addition, ADONIS has a specific significance for OntoGov, because there already exists a meta-model for administration and government processes that reflects specific requirements of this application domain.

5.4.1 ARIS

ARIS - House of Business Engineering (see Figure 22) is an integrated concept describing and executing business processes. It distinguishes for views: organisation, data, function and process. For each view there exist different model types. Processes are modeled as extended event-driven process chains, integrating elements of the different views.

![Figure 22. ARIS views](image_url)
Furthermore, ARIS is a framework for developing real application software. In the following we outline the meaning of the different levels shown in Figure 23.

**Figure 23. Aris’ Levels**

**Level I: Process Design**

Level I describes business processes according to the routing. Therefore, the ARIS concept provides a method to cover every aspect of the business process. Methods for optimizing and guaranteeing the quality of processes are available, too.

**Level II: Process Management**

Level II plans and monitors every current business process from the “business process owner’s” point of view. Various methods of scheduling and capacity control and cost analysis are available. By monitoring the process, the process manager is aware of the status of each process instance.

**Level III: Process Workflow**

Level III transports the objects to be processed, such as customer orders with the corresponding documents or insurance claims in insurance companies, from one workplace to the next. The documents are then stored in “folders”. Workflow systems carry out the material handling in electronically stored documents.

**Level IV: Process Application**

Level IV processes the documents transported to the individual workplaces; that is where the functions of the business process are executed. At this level, computer-aided application systems – from simple word processing programs to complex standard software modules and internet applets – are used.

**Combination of the Four Levels and Framework**
Those four levels of ARIS – House of Business Engineering are connected interdependently. Level II (Process Management) delivers information about the profitability of current processes. That information forms the basis for a continuous adjustment and improvement of the business processes at Level I. The process workflow reports data regarding times, amounts and organizational allocation etc. to Process Management. The workflow systems at Level III start applications at Level IV. The framework is the fifth component, which includes all four levels. It combines architecture and application knowledge, from which concrete applications are configured.

5.4.2 ADONIS

ADONIS is a generic business process management toolkit that is based on a meta modelling concept. Thus, ADONIS can be adapted for various applications. There exist specific versions supporting various modelling methods, e.g.:

- ADONIS Standard - generic method for business process modelling and analysis;
- ADOegov - specialized method for E-Government applications;
- ADOlog - modelling of supply chains based on the SCOR standard;
- ADOit - planning, steering and evaluation of IT development, IT deployment and IT operation in a company; based on ITIL;
- ADOscore - creating and implementing balanced scorecards;
- ADOmed - mapping of operational and organisational structures in the health sector.

In addition to these program packages there exist libraries for various modelling techniques, e.g. event-driven process chains (well known from ARIS), UML, or entity relationship diagrams.

Every object in any model is further described by attributes. Important kinds of attributes are the inter-model references. Thus, for example, roles of the organisational model are associated to the activities of the process model. Each activity in the process model may be linked to resources and documents. The documents can be either working documents (like forms that have to be filled) or background information like regulation, laws or any kind of documentation or manual.

ADONIS offers four simulation algorithms. Time (cycle time, execution time, waiting time, idle time) and costs can be explicitly represented for each activity together with the number of process invocations and probabilities of variable values and decisions.

Model data and also the graphical representations can be exported into various formats. RTF and HTML are mainly for documentation and communication purposes. XML export allows to create any format. Together, meta modelling and XML export make it possible to use ADONIS as a modelling front end for OWL-S and any other process modelling language.

In the following the ADONIS Standard and ADOegov methods are described.
ADONIS Standard Meta-model

The ADONIS Standard Method is a generic modelling technique for displaying processes and organisations. The goals of this method are modelling, documentation, optimisation and cost-cutting. The process modelling language is based on flow diagrams.

Process Map

Process maps enable a quick overview over the processes in the corporation. Connected processes can either be displayed connected or organised in whole process hierarchies. With inter-references and links, details may be called with just a mouse click.

Process Models

Business process models are the core of all corporation modelling. They stand for the processes and organisational flows. A process is composed of activities and sub processes, needed to complete a distinct task. Steering elements (successors, decisions, parallelisms, terminating conditions, etc.) enable the modeller to map the control structure of the processes.

Working Environment Models

Working environment models display the structure of the corporation. After the process models, they are the most important model type with a lot of responsibilities. First of all, the company structure can be displayed as (hierarchic) organigrams. This serves mainly for documentation purposes. Within the departments and working groups, performers can be defined, having certain Roles, which in turn are associated with the activities of the process descriptions. This enables the analysis and simulation to calculate the optimal manpower, responsibilities and capacities.

Document Model

Document models are a pool for all documents accompanying a process or a working environment. Whether it is an input document like a form, a reference document like a text of a law, or an output document like a result list - document models administrate them easily and concise. Via hyperlinks, the real documents can be opened with just one klick.
Use Case Diagram

Use case diagrams are used to map the interactions of users with existing IT systems. The user contacts the system via an interface (e.g. an input dialogue), while the IT system has to fulfill certain requirements (e.g. deliver certain information). Furthermore, use case diagrams are used to explain wishes for future software solutions.

ADOegov Meta-model

In the following we exemplify some relevant features of the ADOegov meta-model using the example of passport issuing.

ADOegov has a number of predefined model types, the most important being:

- Strategy Model
- Process map
- Business process model
- Business process model with security
- Resource model
- Working environment
- Organisation chart
- Role model
- Class diagram

In the following, some meta models, that are especially relevant for OntoGov, are described in more details.

Strategic Model

The strategic modelling level specifies and defines products and services according to the life situation concept. Furthermore, in addition to the circumstances, all associated communication, money and product flows can be displayed.

Business Process Model

The ADOegov process meta model is an extension of the standard model with specific features for public administration processes. Figure 24 shows a model for the process of passport issuing.

The ADOegov process meta model provides control structures for decision (XOR split), concurrency (AND split) with synchronisation (AND join) and conditional path (OR split).

The main modelling objects are activities, subprocesses and resources. There are a number of attributes defined for each of these object types. Activities, for instance, have attributes describing the activities (name, description), the input and output resources, and information about cost and time restrictions (used for simulation purposes).

By inter-model relations each activity can be linked to roles, persons or departments in the organizational model.
Figure 24. ADOegov process model for Passport issuing

In addition to the standard process model, it is possible to assign security levels to each activity. Thus, a security model is generated automatically as a copy of the business process model.

There are different kinds of resources associated to some activities. These are instances of the object type “resource”; the graphical representation of resources depends on the value of the attribute “resource type”.

**Working Environment/Organisation Chart**

The organisational structure of authorities and departments can be modelled using two interlinked model types. The Organisation charts are connected to the products and services from the Strategy Model and define responsibilities. The Working Environment Models specify single bureaus down to single employees and their duties and roles. They are interlinked with the Business Process Models.

**Resource Model**

Resources and activities can be linked to objects in the resource model. The resource models represent resources like various types of documents (e.g. law, regulation, form, webpage, certificate, treaty), human experts, and IT resources. Figure 25 shows the resources that have been modeled for the process of passport issuing. There are two application forms – one paper form and an online form for E-Government - four checklists and an internal form for the communication between police station and passport authority.

For each document there is an attribute “source”. The value is the link where the document can be found. After export to HTML, this link is represented as an hyperlink to directly access the document (see Figure 26).
Figure 25. ADOegov resource model representing input and reference documents for passport issuing

Figure 26. ADOegov resource model with documents needed for passport application
With the resource resource model, every process or activity can be associated to the laws and regulations it is based upon. Thus, if a law or regulation changes, the affected services can be identified. Figure 27 shows a model with three laws and a regulation and a process instruction for one of the laws.

![Figure 27. ADOegov resource model for documents the process depends upon](image)

*Class Diagram*

Using the class diagramm, a kind of simple ontology can be represented. The class diagramm supports classes and instances as well as generalisation, association and composition relations (see Figure 28).

**Critique of ADONIS**

ADONIS offers a rich set of models for business processes, organisational charts, resources etc. Because of its meta-modelling approach, it has nearly unlimited expressiveness. It has been proved that it can be customized for various modelling methodologies. Its ADOamt version also provides class diagrams which is a kind of ontology language.

There are various application scenarios for ADONIS models:

- First, they are used for documentation purposes. Organisation manuals can be generated automatically by exporting models into HTML or RTF format. An important application is quality management, e.g. ISO certification, where process documentation is fundamental.

- Second, the ADONIS models form the basis for the implementation of information systems, in particular ERP systems or workflow-management systems.

- Third, business process optimization is supported by various analyses and simulation tools. ADONIS also supports activity based costing.
In this sense, ADONIS has a kind of operational semantics. However, there is no formal semantics of the various models.

Because of its meta-modelling capabilities, ADONIS could be adapted as a user interface for the OntoGov platform: using the meta-modeller the modelling objects and attributes can be customized to the requirements of the OntoGov ontologies. Since the ADONIS models can be exported to XML, it is rather easy to transform them into OntoGov domain and service ontologies and import them into the OMS.

Figure 28. A very simple ADOegov class diagram

5.5 Specific Requirements for E-Government Process Modelling

The process flow of E-Government processes can be modelled using the standard control structures and objects defined in any of the above-mentioned standards. However, there are specific requirements concerning security and retraceability of modelling foundations.

In most countries there exist very strict rules about the access and exchange of personal data. The process modelling language must allow to document security levels, to implement the rules for data access and to specify demands for secure connections. Identification and authentification of process participants is another
important topic to ensure data security. Thus, modelling how identification can be verified is a requirement for E-Government applications.

While in private organisations the decisions for process definitions are mainly based on time, cost and quality criteria, government processes must be in accordance with the existing law and regulations from different levels (state, region, and municipality). Therefore it is very important to document the laws and regulations the process is based upon – not only for the whole process but maybe also for specific activities.
6 OntoGov lifecycle framework

The increasing complexity of E-Government services demands a correspondingly larger effort for management. Today, many system management tasks, such as service re-configuration due to changes in the law, are often performed manually. This can be time consuming and error-prone. Moreover, it requires a growing number of highly skilled personnel, making E-Government systems costly. In this section, we show how the usage of semantic technologies for describing E-Government services can improve the management of changes. We have extended our previous work in ontology evolution [11], [17], [39], in order to take into account the specificities of ontologies that are used for description of semantic web services. Even though we use the E-Government domain as an example, the approach is general enough to be applied in other domains.

6.1 Introduction

E-Government is a way for governments to use the new technologies to provide people with more convenient access to government information and services, to improve the quality of the services and to provide greater opportunities to participate in the democratic institutions and processes [38]. In addition to providing new ways of working with citizens, enterprises, or other administrations, E-Government is also concerned with creating an integrated environment for the development, deployment and maintenance of online services [40]. In a fast changing world, this last requirement is especially important. Moreover, in the current economic situation, budgets are reduced and opportunities for gaining efficiency seem to be inevitable: the costs of control and maintenance have become the prime concern of public management. The emphasis in E-Government is thus shifting from implementation to cost efficient operations of service or data centres [34]. This effort includes the development of shared services centres that provide common services to local government organizations without affecting the autonomy of organizations and providing the flexibility to enhance and include additional functionality [31]. In such a distributed environment, the problem of efficient management of changes in E-Government has become even more critical.

The main focus of the current change management activities is the resolution of the so-called dynamic modification. It refers to the problem of managing running processes when unanticipated exceptions arise during a task execution, such as the appearance of some hazards in a system, or obtaining some unexpected results. These approaches ensure the consistent operation of a legacy system under unpredictable problems. However, in a dynamically changing political and economical environment, the regulations themselves have to be continually improved, in order to enable the efficient function of a modern society. Taking into account an enormous number of public services and dependencies between them [26], as well as the complexity of interpreting and implementing changes in government regulations, the process of reconfiguring the existing legacy systems (the so-called static modification) seems to be quite complex. Indeed, an efficient management system must provide primitives to allow the progressive refinement without rewriting it from scratch, and must guarantee that the new version of the service is syntactically and semantically correct.
[27]. However, an efficient management system for resolving static changes in an E-Government domain does not exist. In this section, we present such an approach.

The approach is based on enriching current mechanisms for implementing E-Government processes, i.e. web services, with semantic technologies, in order to support a more efficient management of changes. Indeed, the current languages for describing web services, WSDL\(^{37}\) and their composition on the level of business processes (BPEL4WS\(^{38}\)) lack semantic expressivity that is crucial for capturing service capabilities at abstract levels. We argue that business process flow specifications should be defined at abstract task levels, leaving open the details of specific service bindings and execution flows. This abstract level enables the definition of domain specific constraints that have to be taken into account during the (re)configuration of a process flow. In order to model this abstract representation of web services, we base our work on the OWL-S (see section 4.4.2) and WSMO ontologies (see section 4.4.2.2), which support the rich description of web services for Semantic Web. The family of semantic markup languages lays the foundation for semantic web services [35]. In this research, we extend these efforts, in order to support the efficient maintenance of semantic web services.

Since the descriptions of semantic web services are ontologies themselves, we base the web services change management on our previous work in distributed and dependent ontology evolution [11]. It enables us to develop a formal framework for coping with changes which includes the consistency of service descriptions, possible changes, as well as their resolution. Consequently, we can reason about the change management process, making it very flexible and efficient.

This section is organized as follows: in section 6.2, we give a short example from the E-Government domain explaining the requirements for a management system. This system is described in section 6.3. The problem is reduced to the evolution of the Meta Ontology that is used for the service description (section 6.3.1). We define the set of changes (section 6.3.1.2) and consistency constraints (section 6.3.1.3) that this ontology introduces. Finally, we propose procedures for propagation changes from changes in the business logic to description of services (section 6.3.2) and between services (section 6.3.3). Section 6.4 contains the description of the conceptual architecture of the OntoGov system. Before we conclude, we present an overview of related work.

### 6.2 Motivating Example

In order to make the description of the approach more understandable, we define here the basic structure of an E-Government system and give a motivating example that will be used through the section.

As shown in Figure 29, there are four basic roles played by actors in an E-Government system:

- politicians who define a law;

\(^{37}\) [http://www.w3.org/TR/wsdl](http://www.w3.org/TR/wsdl)

- public administrators (i.e. domain experts) who define processes for realizing a law;
- programmers who implement these processes and
- end-users (applicants) who use E-Government services.

Figure 29. Different views on the E-Government services

Domain experts have the key role. They possess a very good knowledge about the E-Government domain. This knowledge is needed for the design of a public service. It includes the legislation that a service is based on, the respective law, related decrees, directives, prerequisites etc. Based on the interpretation of a law, a domain expert describes a service as a sequence of activities that have to be done, which represents a business process. For example, the generic schema\(^\text{39}\) for the public service for issuing (renewal) a driving licence is realized through the following five activities: (i) application, (ii) verification/qualification, (iii) credential issuance, (iv) record management and (v) revenue collection.

In the application activity, all the necessary application data/documents are provided by an applicant. In the next activity, the provided information/documents are verified (e.g. validity and liquidity of a credit card) and are qualified by testing whether the applicant meets the qualification requirements. In the issuance activity either a permanent or temporary credential document (i.e. driving licence) is issued. The record management activity ensures the ongoing integrity of the driving licensing and control record. Finally, the required fee is charged from the applicant’s bank account. Each activity requires some inputs, produces some outputs. It can be executed only

\(^{39}\) Any process that accepts/enrolls applicants for a fee and then issues some sort of credential has the same generic structure – http://www.aamva.org/Documents/idsSeptember2003StatusReport2Attachment9.pdf
when its preconditions are fulfilled and it has postconditions that define the next activity in a conditional manner. In the case of the application activity of the driving licence service, inputs include a birthday certificate, the output is an application form, the precondition is that the applicant is older than 16 and the postcondition is that all fields in the application form are filled. Further, each activity can also be decomposed into several subactivities or can be specialized.

The crucial activity is the verification/qualification, since it reflects the constraints contained in the law. For example, it implements a rule that a person younger than 16 cannot apply for issuing the driving licence, whereas for motor cars (category B) the minimal age is 18. From the business process management point of view, the law can be treated as the business rule required to achieve goals of an organisation (defined by its business policy).

Due to the changes in the political goals of a government, changes in the environment, and changes in the needs of the people, or due to the possibility to organize regulations in a better way, the politicians might (i) make the revision of a law by accepting an amendment, (ii) enact a new law or (iii) even repeal a law. In the case of a new amendment, the domain expert must understand the changes in the law caused by the amendment; locate activities/services in which this law has to be implemented, and translate changes into the corresponding reconfiguration of the business process.

Let us continue the example with the driving licence. Recently, the German law that regulates issuing driving licences has been changed, so that foreigners from non-EU countries must have the German driving licence, although they have a domestic licence. Let us analyse which changes in the existing business process for issuing the driving licence will be caused by this change in the law. For each change, we discuss the role that an efficient change management system should play.

First of all, the domain expert should locate a business process and the corresponding activities that should be modified due to this change in a law. This is a time-consuming action if it is performed in a non-systematic way. Therefore, an efficient change management approach should inform the domain expert on these activities automatically. It means that each business activity must contain a reference to a chapter/paragraph/article/amendment of a law it implements. For example, the activity verification/qualification of the driving licence service is based on the Chapter 2, Paragraph “Mindestalter”\(^\text{40}\) in the Law “Bundesgesetz ueber den Fuehrerschein”.

After finding the service that has to be modified, the domain expert has to decide how to do that. She can specialise this service in the new one or she can adapt it to include new requirements. Let’s assume that the domain expert made a decision to generate a specific driving licence service for foreigners. This service should not be generated from scratch. Rather, it should be a specialization of an already existing driving licence service. The domain expert has to change the preconditions of this new service, since it is only for foreigners from non-EU countries. This automatically causes a change in the preconditions of the original service \(^\text{41}\), since the preconditions of two different services that provide the same functionality must be disjoint. Only in

\(^{40}\) Since we analyzed the German laws were considered as examples, we mention here the original titles in German.

\(^{41}\) The precondition of the original service was that a person is older than 18 years. Now, this precondition is extended by the condition that a person must be from EU.
this way will the run-time system know which service to execute. It is clear that when
the preconditions are semantically defined, the judgment about the inclusion relation
among them can be done automatically.

Further, the verification/qualification activity of the new service requires checking
whether a foreigner already has a domestic licence. Therefore, a new input for that
activity is necessary. Since each input has to be supplied, this change is propagated to
the previous activity, i.e. the application activity, which is responsible for the
interaction with an applicant. It means that that activity has to deliver (as its output)
the information about the domestic licence, the validity of which should be tested in
the verification activity. Consequently, the application activity of the new service
needs an additional input compared to the original service.

Obviously, different changes in a law have different consequences in the existing
services. We briefly discuss one more example. Recently, the German law that
regulates issuing driving licences has been changed, so that teenagers older than 17
can obtain a (temporary) licence for motor cars if they pass the exams and if they
drive with a person that is older than 25, has the driving licence for more than five
years, and has scored less than 20 negative points42 in the last five years. In that case,
the older person must have a licence for co-driving. This change in the law requires
changes in the postconditions of the verification/qualification activity: instead of
approval and non-approval of the licence, it can be temporarily approved. Further, the
credential issuance activity has to generate an additional output, since the new co-
driving licence should be printable, as well. An efficient change management system
should enable the domain expert to perform all these changes efficiently (e.g. to make
a minimal set of additional changes) and to ensure the overall consistency of the
reconfigured service automatically (e.g. to prohibit that an activity has two
contradictory preconditions).

In the rest of this section, we present a change management system that fulfils the
above mentioned requirements.

### 6.3 Our approach

Given the requirements described in section 6.2, we have developed an approach for
the change management for semantic web services. Note that even though we use the
E-Government domain as an example, the approach is general enough to be applied in
other domains. In order to emphasise this generality, in this section, we substitute the
E-Government vocabulary used in the previous section with the commonly-used
business process management terminology. Therefore, instead of the term law we use
a business rule, a public E-Government service is treated as a business process and a
manager plays the role of a domain expert.

Since we assume that services are described using ontologies (e.g. OWL-S or
WSMO), the management of changes requires the management of these semantic
descriptions. Therefore, our approach can be based on our previous work in ontology
evolution ([11], [39], [17]). Moreover, we have extended the work, in order to take
into account the specificity of semantic web services. The problems that have to be

---

42 Negative points are collected by participating in some traffic accidents.
resolved are shown in Figure 30. In section 6.3.1 we firstly define the extensions of our ontology evolution approach (cf. 1 in Figure 30). Then, in section 6.3.2 we discuss the way of bridging the gap between business rules and semantic web services implementing these rules (cf. 2 in Figure 30). Finally, in section 6.3.3 we define the procedures for the change propagation in semantic web services by defining the semantics of the required changes (cf. 3 in Figure 30).

![Diagram]

**Figure 30. Change Management in the E-Government domain**

### 6.3.1 Evolution of the Semantic Web Service Ontology

Ontology evolution can be defined as the timely adaptation of an ontology and a consistent propagation of changes to the dependent artefacts [17]. In this deliverable, we extend our approach for the ontology evolution toward handling the evolution of semantic web service ontologies. Since the evolution is driven by the set of changes that have to preserve the consistency, the approach requires:

(i) the explicit specification of changes that can be applied and

(ii) the consistency definition.

Both of them heavily depend on the underlying model and, thus, they vary from application to application. Therefore, we firstly introduce an ontology that is used for describing semantic web services (see section 6.3.1.1). Secondly, in section 6.3.1.2 we define more complex changes that can be applied to these descriptions. Finally, in section 6.3.1.3 we specify the consistency constraints that are derived from the semantics of this ontology.

---

Note that in the E-Government domain business rules represent the laws, since the laws define how to realize the E-Government services.
6.3.1.1 Ontologies used for modelling semantic web services

The first step that has to be clarified is the description of web services. We distinguish among the following ontologies:

- *Meta Ontology* that contains entities needed to describe services;
- *Domain Ontology* that contains domain specific knowledge;
- *Service Ontologies* that describe concrete services.

6.3.1.1.1 Meta Ontology

In defining the *Meta Ontology*, we based our work on the *OWL-S* and *WSMO* ontologies. We do not consider dynamic web services whose process flow can be composed on the fly. However, we allow the dynamic binding of web services during the execution. Therefore, we focus on static web services, whose composition is explicitly predefined by the business rules, as we showed in 6.2. In order to model the dependency between a business rule and the service implementing it and to take into account the other specificities of the E-Government services we introduce the *Meta Ontology*. This ontology contains entities needed to model the E-Government services.

Similarly to the OWL-S ontology, the *Meta Ontology* consists of two parts:

- the profile that is used for the service discovery and
- the process model that is used to describe the process flow.

Subsequently, we describe these ontologies in detail.

Profile Ontology

To define the *Profile Ontology* we extend the *OWL-S service profile ontology* in several ways. First, we extend the *OWL-S service profile ontology* with the property “*hasReferencedBusinessRule*” that establishes a reference between the service description and the business knowledge that is represented in the form of an ontology. This ontology is called *Business Rule* ontology and depends on the application domain. In the E-Government domain, this ontology contains the knowledge about laws, and is called the *Legal Ontology*. It is important mentioning that this ontology may be used as a well-defined vocabulary (semantics) for describing (annotating) both the content and the structure of legal documents [16]. However, for the problem we are aiming to resolve, it is necessary to model only the structure of legal documents, not their content. The *Legal Ontology* is further elaborated in next subsection.

The second extension of the service profile ontology comes from the business process modelling point of view. Indeed, in order to model the resources involved in a business process, we introduce additional entities such as the property “*requires*” and the concept “*Resource*” which can be either a person who is involved in the executing a service or an equipment (i.e. hardware or software) that performs a service automatically. In that way, we establish a bridge between the common language used by business people – in order to describe the business processes (i.e. web services) - and the ontology language used for describing web services.
Finally, the last extension of the OWL-S service profile ontology is achieved by taking into the consideration the standard metadata defined for the E-Government domain (see section 4.3), since ontologies may advance metadata solutions. Our goal was to model all information that exists in the standard including the implicit knowledge. Even though we use the CEN Application Profile v.1.0 metadata standard as an example, we note that similar strategies can be applied for other standards as well. The approach can be summarized as follows:

- the metadata standard is transformed into a set of the ontology properties that are explicitly included in the Meta Ontology;
- the Meta Ontology is extended with several concepts (e.g. the concept “Topic”) representing ranges of these properties with the goal to improve service discovery;
- “hidden” (hard-coded) knowledge embedded in the metadata standard is translated into a set of rules in the corresponding ontologies and is used in typical inferencing tasks.

We note that we did not incorporate the metadata Identifier, since each ontology entity has automatically an identifier attached to it. Modelling knowledge about E-Government services using properties (and not metadata) is very important for finding a corresponding service. From the user point of view, there is the problem which terms or keywords to use when searching for services. Simple keyword queries are valuable in situations where users have a clear idea of what they are searching for, and the information is well-defined. It doesn’t work for the E-Government domain, where the viewpoints and the knowledge levels of the service provider and service consumer may be completely different. Therefore, some mechanism for establishing a shared understanding is needed. Second, simple keyword searches cannot deal with synonyms (“Agent” and “Actor”), abbreviations (“World Wide Web” and “WWW”), different languages (“database” (English) and „Datenbank“ (German)) and even morphological variations (“Point-to-Point Network” and “Point to Point Network”), not to mention the context. This problem can be resolved by defining corresponding relations (e.g., synonym, abbreviation) in the ontology. Ontology relations are also used in the process of navigating through services (for example, it is reasonable to browse from the topic “Network” to the topic “Protocol”).

Moreover, since service creator, publisher and contributor are people, we established relationships between corresponding properties and the concept “HumanResource”. In this way the correctness of the concrete model is improved significantly, since only instances of the concept “HumanResource” can be used to specify the value of the property instance. Note that domains and ranges simply specify schema constraints that must be satisfied for the property to be instantiated. They don't infer new facts, but guide the user in constructing the ontology by determining what can be explicitly said at all.

Finally, axioms and rules are used to infer new knowledge. The possibility to derive information makes the model of a domain more concise, more accurate, and easier for maintenance. Obtaining and formalizing the non-explicit but available knowledge about the knowledge model ensures the advantages over other systems. For example,
the property “hasTopic” is defined as a transitive\(^{44}\) property. It means that if A "hasTopic“ B and B "hasTopic“ C, then it can be concluded that A “hasTopic“ C as well. Moreover, general rules may be exploited when searching for information. If one defines that some service named “S” is about topic “C”, there is no possibility (without programming or explicit specification) to find out that it is about the topics “A” and “B” as well due to “hasTopic” hierarchy.

A profile part of the Meta Ontology is shown in Figure 31. We note that the set of entities that are the range concepts of the “hasReferencedBusinessRule” property are entities of the Legal Ontology that is defined in the section 6.3.1.1.2.

![Figure 31. A profile part of the Meta Ontology](image)

**Process Ontology**

To describe the process flow we define the Process Ontology. We do not reuse the OWL-S process ontology due to the reasons elaborated in section 4.4.2.3. However, we try to combine the results of this work with the results of the business process community by taking into the consideration the specificities of the E-Government domain.

Similarly to the OWL-S process ontology, we distinguish between the services and the control constructs. Services can be either atomic or composite services. For each

---

\(^{44}\) A transitive axiom is a part of the standard set of axioms.
service we define the standard set of attributes such as name, description, etc. However, there are specific requirements concerning retraceability, realisation, security, costs etc. Therefore, we introduce the E-Government specific properties:

- each service can be associated to the laws it is based upon (“hasReferencedBusinessRule”). We note that it is very important to document the laws and regulations it is based upon - not only for the whole process but maybe also for specific activities;
- each service can be associated to the software component that implements it (“hasReferencedSoftware”). However, it is possible that the same description of the service is related to different implementations (e.g. in Switzerland this depends on the physical location). To inform the workflow engine about the software component that has to be invoked, it is necessary to model the decision attribute (“hasDecisionAttribute”);
- it is possible to assign security levels to each service (“hasSecurityLevel”);
- information about cost and time restrictions can be also specified (“hasCost”, “hasTimeRestriction”, etc.).

Similarly to the OWL-S process ontology, services have inputs (“hasInput”) and outputs (“hasOutput”). The concepts “Input” and “Output” are defined as subconcepts of the concept “Parameter”. Since some inputs have to be provided by the end-user the concept “User-defined Input” is defined as a specialisation of the concept “Input”. To define the equality between two parameters we introduce the symmetric property “isEqualTo”.

Since it is required that inputs/outputs are defined in the domain ontology, we introduce the additional concept “Reference” due to two reasons:

1. a property may be attached to several domain concepts;
2. a concept defined in the domain ontology may have many properties and only a subset of them is used as an input.

In order to specify the context of the usage of a property and to select a subset of them, we introduce the properties “hasConcept” and “hasProperty”, respectively. The range of these properties is the “KAON-Root” concept that is included in each KAON ontology. By using the meta-modelling facilities provided by the KAON management system, it is possible to reference any entity (i.e. a concept, a property or an instance) defined in the domain ontology. Furthermore, to name a parameter we define the attribute “hasName”.

The next difference in comparison to OWL-S process ontology is related to the conditions of a service. While OWL-S uses preconditions and effects to refer to the changes in the state of resources, we adopt the WSMO interpretation [30]. We use preconditions for defining what a service expects for enabling it to provide its service. Postconditions define what the service returns in response to its input. Indeed, they establish the relationship between inputs and outputs.
For a composite service we define the following additional properties: the property “hasFirst” indicating the first service in the process flow\(^\text{45}\) and the transitive property “consistsOf” indicating all services that a service includes. Further, a set of rules\(^\text{46}\) can be specified:

\[
\text{FORALL } X[\text{consistsOf} \Rightarrow Y] \gets \\
X: \text{Service}[\text{hasFirst} \Rightarrow Y] \text{ and } Y: \text{Service}.
\]

\[
\text{FORALL } X[\text{consistsOf} \Rightarrow Y] \gets \\
X: \text{Service}[\text{consistsOf} \Rightarrow Z] \text{ and } \\
Z: \text{Service}[\text{hasNextControlConstruct}^\text{47} \Rightarrow W] \text{ and } \\
W: \text{ControlConstruct}[\text{hasNextService} \Rightarrow Y] \text{ and } Y: \text{Service}.
\]

\[
\text{FORALL } X[\text{hasReferencedBusinessRule} \Rightarrow Y] \gets \\
X: \text{Service}[\text{consistsOf} \Rightarrow Z] \text{ and } \\
Z: \text{Service}[\text{hasReferencedBusinessRule} \Rightarrow Y].
\]

For example, the last rule can be interpreted in the following way: If a part of a service (either an atomic or a composite service) is related to the some part of the law, then the service itself is related to the same part of the law.

The process model provides the following control structures: sequence, split, join and if-then. We do not include while, repeat etc. which are defined in the OWL-S process ontology, since none of the E-Government use-cases we analysed requires them.

To connect the services and the control constructs we define the following properties:

- “hasNextControlConstruct” and “hasPreviousControlConstruct” whose domain concept is the concept “Service” and the range concept is the concept “ControlConstruct”;
- “hasNextService” and “hasPreviousService” whose domain concept is the concept “ControlConstruct” and the range concept is the concept “Service”;

The properties “hasNextControlConstruct” and “hasNextService” as well as the properties “hasPreviousControlConstruct” and “hasPreviousService” are inverse of each other.

Further, for the concept “Service” we define the properties “hasNext”, “hasPrevious” and “hasSpecialisation”. The range of these properties is also the concept “Service”. The values of these properties can be inferred based on the following rules:

\[
\text{FORALL } X[\text{hasNext} \Rightarrow Y] \gets \\
X: \text{Service}[\text{hasNextControlConstruct} \Rightarrow Z] \text{ and } Y: \text{Service} \text{ and } \\
Z: \text{ControlConstruct}[\text{hasNextService} \Rightarrow Y].
\]

\(^{45}\) This information can be considered as redundant since it can be always derived from a service model based on the rule that each service that does not have a previous service/control construct is a first service. However, it should be specified in order to optimise the run-time performance.

\(^{46}\) The rules are written in the F-Logic notation [64]. F-Logic is a deductive, object oriented database language which combines the declarative semantics and expressiveness of deductive database languages with the rich data modelling capabilities supported by the object oriented data model.

\(^{47}\) This property is defined below.
FORALL X[hasPrevious⇒Y] ←
X:Service[hasPreviousControlConstruct⇒Z] and Y:Service and
Z:ControlConstruct[hasPreviousService⇒Y].

FORALL X[hasSpecialisation⇒Y] ←
EXISTS Z Z:Service and Y:Service[consistsOf⇒Z] and not
X:Service[consistsOf⇒Z].

For the concept “IF-THEN” several additional properties are defined in order to
determine the next services based on the fulfillment of the condition.

A process part of the Meta Ontology is shown in Figure 32.

Figure 32. A process part of the Meta Ontology

6.3.1.1.2 Legal Ontology

In order to find a service that has to be updated after a change in a law, we need to
establish a reference between the service description and the law. One solution is to
model laws in the form of an ontology. This ontology is called the Legal Ontology.
The goal of the Legal Ontology is to model the structure of the legal documents. To
develop this ontology we have analysed the structure of legal documents in
Switzerland, Greece and Spain, since the goal of our project is to pilot the OntoGov
system, which is going to be developed, at our partners coming from these countries.
Figure 33 shows the Spanish law related to the citizens rights. A part of Switzerland's
constitution pointing out to the law related to the basic rights, civil rights and social
goals is depicted in Figure 34. The overall structure of the laws in Greece is shown in
Figure 35.
Based on the analysis of the legal documents in Spain (see Figure 33), we found a general structure of the laws in Spain. Each law consists of chapters; a chapter may have several articles, and an article may contain "points". A law can be identified by its identifier (e.g. 30/192); it has a name, the date of creation, etc. Further, a law can be defined to be general (for public administrations, municipalities, local administrations etc.) or it can be constrained to be applied in one type of PAs (e.g. municipality). Moreover, there is a short description of a low. This description may be used for the semi-automatic creation of the Legal Ontology by extracting terms that are important for that law.

Title IV: Regarding the Public Administrations Activities.

Chapter I: General Rules.

Art 35: Citizens rights:

a) Right to have knowledge about the procedure status and get copies of documents.
b) Right to identify the employees who are performing his procedure.
c) Right to provide a copy authorized by the Public Administration and keep the originals.
d) Right to make use of the regional language (in our case catalan)
e) Right to formulate allegations (complaints) and to provide documents in any phase of the procedure, these documents must be taken into account by the corresponding PA.
f) Right to not provide a document more than once.
g) Right to be informed about the procedure.
h) Right to have the ability to access the registries and Public Administration files according the Spanish Constitution and other laws (included this one).
i) Right to be treated with good manners.
j) Right to demand responsibilities.

Figure 33. An example of the Spanish legal documents

In Switzerland there is the similar structure of the legal documents as shown in Figure 34. A law has a title, a unique reference number, an abbreviation etc. It consists of parts (e.g. “1. Kapitel”) and a part consists of several articles (e.g. “Art 8”). An article has a title, an article number and a short description etc. First under-structure of the article is the so-called article alinea (or sub-Article). It is the smallest part referred in the OntoGov project.

Finally, the law in Greece (see Figure 35) consists of articles. Each article contains paragraphs, whereas a paragraph may contain bullets for further explanation of the articles. For each level of abstraction the textual description is defined.

Based on the analysis of the structure of the legal documents in three different countries we conclude that the legal documents have very similar structure independently of the country it is defined for. Even though different countries use different terminology\textsuperscript{48} to organise their legal documents, all them use three levels of abstractions as shown in Figure 36. Therefore, it was possible to extract the general structure of a law and the represent it in a form of the Legal Ontology. A part of the

\textsuperscript{48} Note that each country uses terminology in its own language. However, after the translation in English the terminology remains different.
Legal Ontology is shown in Figure 37. The different terminology is modelled through the lexical layer of the Legal Ontology. The lexical layer offers means for representing labels, synonyms, morphological modifications and documentations of an ontology entity in different languages.

Bundesverfassung
der Schweizerischen Eidgenossenschaft


2. Titel: Grundrechte, Bürgerrechte und Sozialziele
1. Kapitel: Grundrechte

Art. 7 Menschenwürde
Die Würde des Menschen ist zu achten und zu schützen.

Art. 8 Rechtsgleichheit
1 Alle Menschen sind vor dem Gesetz gleich.
2 Niemand darf diskriminiert werden, namentlich nicht wegen der Herkunft, der Rasse, des Geschlechts, des Alters, der Sprache, der sozialen Stellung, der Lebensform, der religiösen, weltanschaulichen oder politischen Überzeugung oder wegen einer körperlichen, geistigen oder psychischen Behinderung.
3 Mann und Frau sind gleichberechtigt. Das Gesetz sorgt für ihre rechtliche und tat-

Figure 34. An example of the Swiss legal documents

Figure 35. Overall structure of the laws in Greece
Legal documents have the structure that is typical for law, i.e. they contain explicit regulations (chapters, paragraphs, articles, etc.). This is modelled through the concepts “Law”, “Heading I”, “Heading II”, “Heading III”, “Amendment” etc. and the relationships between them. For example, the property “hasHeadingI” indicates that a law is specified through chapters, parts or articles that represent labels in different languages (in Spanish, German and Greek, respectively) of the concept “Heading I”. Further, the property “modifies” stands for the fact that an amendment modifies a law or some of its parts.
6.3.1.1.3 Other Ontologies

The Domain Ontology contains the domain specific knowledge. Returning to the example described in section 6.2, the domain ontology would consist of entities such as the concept “driving licence”, properties “hasName”, “dateOfIssuance”, etc.

Finally, for each service, a Service Ontology that includes all previously mentioned ontologies is defined, and it might include other Service Ontologies. The Meta Ontology is reused, since it contains entities needed to describe a web service. The Domain Ontology is included, since it defines the domain-specific vocabulary. Therefore, the Service Ontology specialises concepts from the Meta Ontology (e.g. atomic service), establishes relationships between these concepts (e.g. what is the next or previous atomic service) and relates them to the domain entities from the Domain Ontology. For example, the service ontology for the driving licence issuance E-Government service describes that it is a composite service that is realized through the application, verification/qualification etc. (see section 6.2), which can be considered as atomic services (i.e. an activity). Therefore, this ontology includes Meta Ontology, since the Meta Ontology defines the building blocks for the service description. Each of these services (application, verification/qualification etc.) is related to the domain ontology. For example, the application service requires the birth certificate that is the domain knowledge.

The dependencies between all previously mentioned ontologies are shown in Figure 38.

Figure 38. The inclusion graph of ontologies used for describing semantic web services

6.3.1.1.4 An example

In this section we show how a concrete example is modelled using the ontologies introduced in the previous sections.

---

49 “Realized” stands for representing a sequence between services.
Figure 39 shows the process model\(^{50}\) of a service that is about performing deregistration of a citizen.

![Diagram](image)

**Figure 39. An example of the E-Government service**

The service consists of several activities:

- starting the process ‘deregistration’ by the citizen;
- the citizen fills in a form for deregistration;
- the data filled in will be checked by the municipality whether a citizen is registered as resident or not;
- in case the citizen isn’t registered a refusal of deregistration is sent to the citizen;
- in case the citizen is registered other departments of the municipality will be informed about the deregistration;
- the database where the citizen’s data is stored will be updated;
- when update is performed a notification about deregistration is sent to the citizen;
- ending the process ‘deregistration’.

\(^{50}\) The process model is developed using the ivyGrid 3.1 business process modelling tool (http://www.ivyteam.com).
The profile of this service is shown in Figure 40. In Switzerland the service is based upon the following legal requirements:

- Bundesverfassung der Schweizerischen Eidgenossenschaft (Art.13; Art.24; Art.29);
- Kantonales Gemeindegesetz, e.g. Kanton Solothurn (§3 - §7);
- Zivilgesetzbuch (ZGB – Art.23 – Art.25).

![Diagram](image)

**Figure 40. The profile of the E-Government service shown in Figure 39**

The service is provided by the registration officer of the municipality and is annotated with the following keywords: deregistration, announcement of move, signing off, and notification of move. It has a reference to several software components:

- web-site (portal) of the municipality;
- e-mail system of the municipality (NotifyOthers, NotifyCitizen);
- database of the municipality.

Figure 42 and Figure 43 show the same service modelled as in a form of a service ontology. This ontology is based on the Meta Ontology (see section 6.3.1.1.1) and includes the domain ontology as well. The domain ontology is shown in Figure 41.

Due to the complexity of the representation of services, we show two different views on the model: the process flow and the input/output view. The entire ontology is given in the appendix.
Figure 41. A very simple domain ontology

Figure 42. The process model of the E-Government service shown in Figure 39
6.3.1.2 Changes

For the ontology evolution, we defined the set of ontology changes that includes all elementary changes\textsuperscript{51} (e.g. “AddConcept”) and some more complex changes, the so-called composite changes\textsuperscript{52} (e.g. “MoveConcept”) [17]. However, this granularity level should be extended, in order to enable a better management of changes in a service description. For example, to make the service s1 a predecessor of the service s2, the manager needs to apply a list of ontology changes that connects outputs of s1 to the corresponding inputs of s2. We cannot expect that she spends time finding, grouping and ordering the ontology changes to perform the desired update. In order to do that, she should be aware of the way of resolving a change, she should find out the right changes, foresee and solve the intermediate conflicts that might appear, and order changes in a right way. This activity is time consuming and error prone, especially if an ontology is large (e.g. thousand concepts) or complex (e.g. multiple concept hierarchy).

Therefore, managers require a method for expressing their needs in an exacter, easier and more declarative manner. For them, it would be more useful to know that they can connect two services, rather than to know how it is realized. To resolve the above mentioned problem, the intent of the changes has to be expressed on a more coarse level, with the intent of the change directly visible. Only in this way can managers focus on what has to be done, and not on how to do that.

\textsuperscript{51} Elementary ontology changes are derived from the underlying ontology model. They specify fine-grained changes that can be performed in the course of the ontology evolution. They are called elementary changes, since they cannot be decomposed into simpler changes.

\textsuperscript{52} Composite ontology changes represent a group of elementary or composite changes that are applied together.
To identify this new level of changes, we start from the Meta Ontology (see section 6.3.1.1). The abstract, simplified model of this ontology is shown in Figure 44. For each service, one can specify inputs, outputs, preconditions, postconditions, business rules and software components, other services that it either specializes or is connected with. Each of these entities can be updated by one of the meta-change transformations: add and remove [17]. A full set of changes can thus be defined by the cross product of the set of entities of the Meta Ontology and the set of meta-changes. They are shown in Table 12.

![Figure 44. Abstract model of the Meta Ontology](image)

Table 12. The taxonomy of changes of the semantic web ontology

<table>
<thead>
<tr>
<th></th>
<th>Additive Changes</th>
<th>Subtractive Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service</td>
<td>AddService</td>
<td>RemoveService</td>
</tr>
<tr>
<td>Input</td>
<td>AddServiceInput</td>
<td>RemoveServiceInput</td>
</tr>
<tr>
<td>Output</td>
<td>AddServiceOutput</td>
<td>RemoveServiceOutput</td>
</tr>
<tr>
<td>Precondition</td>
<td>AddServicePrecondition</td>
<td>RemoveServicePrecondition</td>
</tr>
<tr>
<td>Postcondition</td>
<td>AddServicePostcondition</td>
<td>RemoveServicePostcondition</td>
</tr>
<tr>
<td>Service Specialisation</td>
<td>AddServiceSpecialisation</td>
<td>RemoveServiceSpecialisation</td>
</tr>
<tr>
<td>Next Connection</td>
<td>AddServiceNextService</td>
<td>RemoveServiceNextService</td>
</tr>
<tr>
<td>Previous Connection</td>
<td>AddServicePreviousService</td>
<td>RemoveServicePreviousService</td>
</tr>
<tr>
<td>Business Rule</td>
<td>AddServiceBusinessRule</td>
<td>RemoveServiceBusinessRule</td>
</tr>
<tr>
<td>Software Component</td>
<td>AddServiceSoftware</td>
<td>RemoveServiceSoftware</td>
</tr>
<tr>
<td>Resource</td>
<td>AddServiceResource</td>
<td>RemoveServiceResource</td>
</tr>
</tbody>
</table>

53 We focus only on the entities that are important for management. Other aspects, such as properties defined in the service profile or grounding ontology, are ignored.

54 Due to the abstraction of the Meta Ontology, only the most typical and most frequently occurring changes are shown, since they are relevant from the management point of view.
The previously defined set of changes is complete and minimal. Completeness refers to the possibility of transforming an arbitrary service description in any other. Minimality refers to the achievement of completeness with a minimal set of changes. These changes can be further combined into more complex changes, such as grouping of services. Further, each of these changes is internally realized as a set of elementary or composite ontology changes.

Changes shown in Table 12 build the backbone of a semantic web service management system. They make the evolution of the semantic description of web services much easier, faster, more efficient, since they correspond to the “conceptual” operation that someone wants to apply without understanding the details (i.e. a set of ontology changes) that the management system has to perform.

6.3.1.3 Consistency

To define the consistency of the semantic web service ontologies, we start from the ontology consistency definition [39]:

**Definition 2** An ontology is **consistent** with the respect to its model if and only if it preserves the constraints defined for the underlying ontology model.

This set of constraints includes invariants, which are consistency rules that must hold for every ontology. For example, a concept hierarchy must be a directed acyclic graph. Since ontologies that are used to describe semantic web services include other ontologies, we define the dependent ontology consistency in the following way [11]:

**Definition 3** A dependent ontology is consistent if the ontology itself and all its included ontologies, observed alone and independently of the ontologies in which they are reused, are ontology consistent.

The *Meta Ontology* can be considered as the meta-level for the semantic web service description. Since the set of consistency constraints heavily depends on the underlying model, the semantics of the *Meta Ontology* defines a set of constraints that all service ontologies have to fulfil. In this section, we discuss how the existing dependent ontology consistency definition has to be enriched, in order to take into account the specificities of the *Meta Ontology*. We introduce the following additional constraints:

- **Service profile specific constraints**:
  - **Business knowledge specific constraints**
    - *C1*: Each service has to have a reference to at least one business rule.
  - **Traceability**
    - *C2*: Each service has to have at least one resource that controls its execution.
  - **Applicability**
    - *C3*: Each service has to have at least one software component attached to it that implements it.

- **Service process specific constraints**:
  - **Completeness**
C4: Each service has to have at least one input.

C5: Each service has to have at least one output.

C6: Each service input has to be either output of some other service or is specified by the end-user.

○ Satisfiability

C7: If the input of a service is the output of another service, then it has to be subsumed by this output.

C8: If the input of a service subsumes the input of the next service, then its preconditions have to subsume the preconditions of the next one.

C9: If two services are subsumed by the same service, then their preconditions have to be disjoint.

• Uniqueness

C10: If a service specialises another service, one of its parameters (i.e. inputs, outputs, preconditions or postconditions) has to be different. The difference can be achieved either through the subsumption relation with the corresponding counterpart or by introducing a new one.

○ Well-formedness

C11: Inputs, outputs, preconditions and postconditions have to be from the domain ontology.

- Domain specific constraints:

○ Structural dependency

C12: Any specialization of the activity A1 must always be a predecessor of any specialization of the activity A2, where A1 and A2 are two activities defined in the Meta Ontology and their order is given in advance (i.e. A1 precedes A2).

It is worth mentioning that only consistency constraints C1 and C12 are domain-dependent. Whereas C1 has a reference to the Business Rules Ontology (i.e. the Legal Ontology in the E-Government domain), C12 is related to the generic schema for the services and specifies the obligatory sequence among activities. In the E-Government domain, C1 requires that each service is related to a law. C12 states that the structure of Service Ontologies must follow predefined rules, so that a service specialising an application service has to precede a specialisation of a verification service.

We give short interpretations of some constraints from the change management point of view:

- C1 enables to find the corresponding service if a law is changed;

- C6 ensures that a change in an output of an activity is propagated to the inputs of successor activities and vice versa;

- C8 prohibits the changes which lead to non-optimal service reconfiguration. For example, if the preconditions for an activity include a constraint that a person has to be older than 18, the preconditions of the next activity cannot be that a person has to be older than 16.
Note that each of these constraints is formally defined. For example, the first constraint $C1$ is formalized as:

$$\forall x \exists y H_c^*(x, \text{Service}) \land H_c^*(y, \text{BusinessRule}) \land x \in \text{domain}(\text{has ReferencedBusinessRule}) \land y \in \text{range}(\text{has ReferencedBusinessRule}) \land \text{has ReferencedBusinessRule} \in P \land \text{Service}, \text{BusinessRule} \in C$$

where $H_c^*$ is the transitive closure of a concept hierarchy $H_c$, $P$ is the set of all properties and $C$ is the set of all concepts defined in an ontology, the function $\text{domain}:P \rightarrow 2^C$ gives the set of domain concepts for a property $p \in P$ and the function $\text{range}:P \rightarrow 2^C$ gives the set of range concepts for a property $p \in P$. The property $\text{hasReferencedBusinessRule}$ relates a Service to a Business Rule.

Finally, we define the consistency of the semantic web services in the following way:

**Definition 4** A semantic web service is a **consistent service** iff its description is dependent ontology consistent and the additional constraints ($C1$-$C12$) are fulfilled.

Note that a change in the business logic does not cause any ontology inconsistency. Regarding the E-Government domain, after the removal of a single input of an activity, the ontology consistency is still fulfilled. However, this change provokes the semantic web service inconsistency, since the consistency constraint $C4$ is not satisfied. Therefore, the extension of the consistency definition is a prerequisite for the management of the semantic web services.

Since semantic web services must be compliant with the set of semantic web service consistency constraints, in the rest of this section, we discuss how to preserve the consistency. We firstly define a procedure that informs managers about changes in the business rules that provoke some inconsistencies. Thereafter, we introduce the procedures for ensuring the semantic web service consistency.

### 6.3.2 Propagating changes from business rules to services

The basic requirement for a management system is that it has to be simple, correct and usable for managers and domain experts. Note that they are responsible for keeping semantic web services up-to-date and don’t need to be experienced ontology engineers. Thus, a management system must provide capabilities for the automatic identification of problems in the (description of the) semantic web services and ranking them according to the importance. When such problems arise, a management system must assist the domain experts in identifying the sources of the problem, in analysing and defining solutions for resolving them. Finally, the system should help in determining the ways for applying the proposed solutions.

In this section we define the procedure for finding the “weak places” in the description of the semantic web services by considering the changes in the business rules and their impact on the consistency. The procedure is focused on discovering inconsistencies in a semantic web service description, whose repairing improves the agreement of this ontology with the business rules. When we designed this support, we assumed that the update would be only a partially automated process rather than a
fully automated process. For example, we do not want to update web services automatically, but rather to notify the domain experts about problems. It is up to the domain expert to decide how to resolve those problems. Our experience shows that this assumption is reasonable. In the E-Government domain, certain tasks could be automated, while other tasks could be supported, but not fully automated. For example, the manager should be informed about a new amendment. However, the realization of this amendment must not be automated, since it requires a lot of domain knowledge that cannot be formally represented in the Legal Ontology, and is a result of experiences. Therefore, our system only makes recommendations about a potential resolution of a problem. For example, a new amendment might be realized through the specialisation of a web service that implements the law for which this amendment is defined.

Obviously, the information about the business rule that is implemented by a service is very important for the change management. It means that the consistency can be achieved only by referring to this knowledge. This was one of the reasons for defining the Meta Ontology (see section 0).

The procedure for propagating changes from business rules to web services is based on our previous work on the evolution between dependent and distributed ontologies, since we assume that the Business Rule Ontology is reused in the Meta Ontology through the replication [11]. In the E-Government domain, the physical distribution is very important, since E-Government services must follow federal, state and local laws that are defined externally. Note that a Service Ontology might reuse the Meta Ontology either through inclusion or replication, which depends whether they are within the same system or not.

The procedure consists of four steps:

1. **Checking actuality of the Business Rules Ontology** – Since each ontology has a version number associated with it that is incremented each time when the ontology is changed, checking the equivalence of the original of the Business Rules Ontology and the replica can be done by a simple comparison of the number.

2. **Extracting Deltas** – After determining that the included Business Rules Ontology needs to be updated, the Evolution Log (see section 6.3.2.1) for this ontology is accessed. The extracted deltas contain all changes that have been applied to the original after the last synchronisation with the replica, as determined by the version numbers. For example, after the addition of the new amendment $A_7$ in the Legal Ontology as the adaptation of the paragraph $P_2$, the delta will contain changes shown in Figure 45.

3. **Analysis of changes** – Each required change is analysed, in order to make recommendations how to adapt the semantic web services. We distinguish between the addition and the deletion of an entity from the Business Rule Ontology. Removals can be resolved directly by applying the existing ontology evolution system, since it ensures the consistency by generating addition changes [17]. However, the addition requires an additional effort that depends on the structure of the Business Rules Ontology. Here we describe how this problem is resolved in the E-Government domain by considering the Legal Ontology. We analyse the addition of a new amendment. The goal is to find services that realize the law related to this amendment, and to order them in an appropriate way. Since
each service is referred to a law/chapter/paragraph/article, the corresponding service can be easily found. In case there are several services referring to the given law (e.g. through a paragraph or an amendment), they are ranked according to the semantic similarity that is based on calculating the distance between two entities in the hierarchy we proposed in [3]. Currently, we are developing a search module that uses NLP-methods to calculate similarity between two texts, no matter the text is the description of an amendment or paragraph.

4. Making recommendation: In order to make recommendations how to adapt the up-to-date semantic web services we use the Lifecycle Ontology. It describes design decisions and their relationship to affected parts of the service as well as to the requirements that motivate the decisions [50]. Since the Lifecycle Ontology is a description of the service design process, which clarifies which design decisions were taken for which reasons, it proves to be valuable for further development and maintenance. During ongoing development, it helps the managers to avoid pursuing unpromising design alternatives repeatedly, but it also facilitates maintenance by improving the understandability of the service design. A description of the design process also supports traceability, since it links parts of the service design to the portions of the specification (i.e. to the activities/services in the process description) they were derived from and to the requirements that influenced design decisions in that derivation.

```
<a:AddInstanceOf rdf:ID="i-1079962974979-1202219624"
    a:has_referenceConcept="Legal#Amendment"
    a:has_referenceInstance="Legal#A7"
    a:inOIModel="file:/C:/ontoGov/Legal"
    a:version="10">...
</a:AddInstanceOf>
<a:AddPropertyInstance rdf:ID="i-1079962991620-1904187797"
    a:has_referenceProperty="Legal#modifies"
    a:has_referenceSourceInstance="Legal#A7"
    a:has_referenceTargetInstance="Legal#P2"
    a:has_previousChange rdf:resource="#i-1079962974979-1202219624"/>
</a:AddPropertyInstance>
```

Figure 45. A part of a log of the Legal Ontology

The proposed approach requires two additional ontologies:

- the Evolution Ontology that helps to detect problems;
- the Information Ontology that helps to resolve a detected problem.

Subsequently, we describe these two ontologies in detail.

6.3.2.1 Evolution Ontology

Information about changes can be represented in many different ways. To communicate about changes, we need a common understanding of a change model and a log model. Therefore, we introduce the Evolution Ontology and the Evolution Log. The Evolution Ontology is a model of ontology changes enabling better management of these changes. The Evolution Log tracks the history of applied
ontology changes as an order sequence of information (defined through the *Evolution Ontology*) about particular change.

Figure 46 explains the dependencies between these ontologies and the ontology that is changing (c.f. the domain ontology or the service ontology). There is a clear distinction between general knowledge about the evolution (i.e. the *Evolution Ontology*), the knowledge that is specific to a particular domain (i.e. the *Domain/Service Ontology*) and the knowledge about the development/maintenance of a model of that domain (i.e. the *Evolution Log*).

![Figure 46. Dependency between Domain/Service ontology, Evolution Ontology and Evolution Log](image)

Note that only a common understanding of the changes (achieved through the *Evolution Ontology*) and of a log (achieved through the *Evolution Log*) enables the synchronisation between the evolving *Domain/Service ontology* and the dependent artefacts (e.g. applications based on this ontology) that have to incorporate or adapt to those changes. Further, the *Evolution Log* based on the formal model of ontology changes (i.e. on the *Evolution Ontology*) enables the recovery from “failure” since it makes possible to undo and redo applied changes in an unlimited way.

In order to have the explicit representation of changes, we need an agreed-upon ontology of changes. Therefore, we develop a special ontology, the so-called *Evolution Ontology* that supports, alleviates and automates the ontology evolution process. It is about a meta-ontology that is used as a backbone for creating *Evolution Logs*. The *Evolution Ontology*, shown in Figure 47, models what changes, why, when, by whom and how are performed in an ontology. Therefore, the most important concept is the “Change” concept. The structure of the hierarchy of ontology changes reflects the underlying ontology model by including all possible types of changes. For example, elementary changes are decomposed into the changes causing the addition (the concept “AdditiveChange”) and the changes provoking deletion (the concept “SubtractiveChange”). The additive changes are further decomposed into e.g. “AddConcept”, “AddProperty”, etc. Indeed, each leaf concept in the hierarchy of the concept “Change” represents a specific ontology change. For the composite changes (see table Table 12) the property “includes” is defined indicating that these changes are realised as a sequence of elementary or composite changes. This sequence is represented in the *Evolution Ontology* explicitly, enabling the declarative specification of the semantics of the composite changes. This richer description of the
changes, their causes, and consequences provide more scope to resolve possible inconsistency.

![Figure 47. A part of the Evolution Ontology](image)

Representing the knowledge about ontology changes in a concept hierarchy enables us to specify the common properties of ontology changes in an efficient way since the inheritance mechanism may be exploited. For example, for each change the additional information, such as the date and time of the change, the version number, as well as the identity of the change initiator may be associated. This is modelled through appropriate properties defined for the concept “Change” and can serve as a source for different knowledge discovery methods, e.g. mining about change trends.

As described previously, elementary changes may cause new changes in order to keep the ontology consistent. Such dependencies may be represented using the “causesChange” property. Groups of changes of one request are maintained in a linked list using the “hasPreviousChange” property. These two properties have completely different semantics. The first one is used to model the “cause-effects” dependency between changes whereas the second one models the order in which the changes are requested. However, both of them are necessary for supporting the reversibility as well as the change propagation. The property “causesChange” makes it possible to reverse all side effects of the required change. The property “hasPreviousChange” allows reconstructing the sequence of required changes and is specialised into two subproperties: “hasPreviousActualChange” and “hasPreviousHistoryChange”. The “hasPreviousActualChange” property indicates
the current state of the ontology by excluding the effects of the inverse changes (i.e. reversibility). In this way, it specifies only the necessary changes to achieve the resulting ontology. On the contrary, the “hasPreviousHistoryChange” property takes into account the sequence of all the changes that have actually taken place. Therefore, it models the actual evolution process in a unique way (since it records all intermediate ontology versions as well). In the case that a log of changes does not contain changes that undo other changes (e.g. AddConcept(“X”) followed by RemoveConcept(“X”)), the “hasPreviousActualChange” and “hasPreviousHistoryChange” properties for each change instance point to the same target instance.

For properties establishing the relations between changes (i.e. “causesChange”, “hasPreviousActualChange” and “hasPreviousHistoryChange”) the corresponding inverse properties are defined enabling the usage of implicit knowledge for some purposes.

The above-introduced properties are universal in the sense that they can be recorded for all ontology changes. However, there are additional properties depending on the change type. These properties are used to represent the peculiarities of a particular type of a change, such as its arguments. The change-specific properties are modelled by using a property hierarchy. For example, entities from the ontology being changed are related to the instances of the “Change” concept through the “hasReferenceEntity” property and its subproperties. The number of properties referencing the domain entities defined for one ontology change and their semantics are specific to that change. Therefore, the property “hasReferenceEntity” is specialised into several subproperties indicating the type of entity that is considered. For example, the “hasReferenceConcept” property is used to reference the changes related to the concepts in the domain ontology or the “hasReferenceValue” designates the changes to the values of property instances. This information is used for the verification of performed ontology changes by checking the type of the entity that the change is applied to. Note that these properties are shared between several concepts in the hierarchy of the concept “Change”. For example, the property “hasReferenceInstance” is specified for changes “AddInstance”, “RemoveInstance”, “AddInstanceOf”, etc. since all of them operate on ontology instances.

We note that the “hasReferenceEntity” property (and its subproperties as well) is not defined as a relation since it is also used to reference the entities from the domain ontology that do not exist anymore (e.g. “RemoveConcept”). Therefore, it is an attribute whereas its value is the unique identifier of the entity from the domain ontology that is changed. This is the reason why the Evolution Log does not include the Domain/Service ontology, but rather refers to it as shown in Figure 46.

Note that the hierarchy of the concept “Change” invariably depends on the underlying ontology model. Defining the standard set would be realistic when there is a common ontology language. However, the set of properties defined for the concept “Change” and its subconcepts are general enough to be considered as a standard for the change representation.

The Evolution Ontology enables the representation, the analysis, the reasoning about, the realisation and the sharing of ontology changes in a more systematic and consistent way. Benefits of using the Evolution Ontology are manifold:
• Changes are represented formally and can be managed formally;
• A history of changes is stored and can be used for recovery or additional analysis;
• Based on the formal representation and the history of changes the change propagation problem may be approached (e.g. any software application that makes use of an evolving ontology would be able to recognise applied changes and to incorporate them easily);
• Using the same representation model for the domain ontology and ontology changes simplifies storage and allows reuse of system components (e.g. searching for entities from the domain ontology can be reused for the searching for the applied changes since they are entities of the Evolution Ontology).

While the structure of the Evolution Ontology aims at better characterisation and understanding of the changes, an Evolution Log records an exact sequence of changes that occurred when an ontology engineer updated an ontology. Therefore, it contains instances of subconcepts of the concept “Change”, which include the elementary as well as the composite ontology changes. Indeed, a request for a change and all its consequences are represented as instances of the corresponding change concepts. Each instance contains a data about a particular change. For example, for all changes a log includes a timestamp, author, version etc. Figure 45 shows a part of a possible log of changes. In this way the level of granularity at which changes are specified is close to a single user-interface operation (cf. AddInstanceOf(“Legal#A7”, “Legal#Amendment”)). However, an Evolution Log provides a complete and unambiguous change specification at a very fine level of detail since for each change request all effects are represented explicitly.

Keeping a record of changes is important. Introducing an ontology for representing them gives additional advantages. The advantage of our Evolution Log is the formal, explicit semantics that is provided through the Evolution Ontology. This enables easier synchronisation between dependent ontologies. Namely, this problem can be approached by comparing the Evolution Logs, which is not difficult due to the fact that logs reuse the same Evolution Ontology. Further, if the formal model for representing changes is used, the developers of applications depending on the ontologies will be well served since they will devote less time and expense to understand and manage changes due to possibility to automate the update. Finally, the meta knowledge provided by the Evolution Ontology can assist intelligent search in the Evolution Log for some previously made changes. For example, if a query (e.g. “Find all “RemoveConcept” changes”) sends back no result, then the Evolution Ontology can be used to generalise automatically the query to find nearest partial matches (e.g. “Find all “SubtractiveChange” changes”). Indeed, reasoning allows inferring implicitly represented knowledge from the knowledge that is explicitly contained in the Evolution Ontology.

6.3.2.2 Lifecycle Ontology

The goal of the OntoGov project is to develop a framework (and a system based on it) that allows for the change propagation and traceability, contributing in this way to the
bridging of decision making with technical realisation. To achieve this goal, it is not sufficient to use ontologies for modelling services. Rather it is required to model the dependencies between different stakeholders that define this business logic in a collaborative way.

In section 6.3.2 we describe the method we use to find E-Government services that have to be updated after changes in the law. Even though it focuses domain experts only on the relevant services, it is still up to domain experts to decide how to synchronise the service with the law. The main problem is to find out the right changes. This activity is time consuming and error prone. The need of domain experts is especially hard to fulfil if an ontology is large (e.g. thousand concepts) or complex (e.g. multiple concept hierarchy). In order to help them to do that, we develop the so-called Lifecycle Ontology that describes the information flow and the decision making process in the public administration.

The Lifecycle Ontology spans the range from the informal specification of requirements to a representation focusing on the realization of the service [50]. It is intended to support the transition from knowledge acquisition to implementation, i.e. the design phase. It includes entities for documenting design decisions and the underlying rationale. In this way it gives concrete clues on how a service ontology has to be modified.

There are indications that, like in conventional software engineering, the design phase is also decisive for the quality of an E-Government service. In order to reduce the potential for design errors, the design phase should be as transparent as possible. One way to achieve this is through documentation of design decisions and their underlying rationale. Design decisions can be viewed as contributions to the satisfaction of requirements. Thus, the rationale of a design decision is its relationship to such requirements. Consequently, the Lifecycle Ontology is used for describing design decisions and their relationship to affected parts of the service as well as to the requirements that motivate the decisions.

In order to achieve these potential benefits, the Lifecycle Ontology comprises a set of entities to describe the design process. In this model, the design process is viewed as a succession of states of the service design. The transition between two adjacent states is effected by activities of the designer, i.e. by a design decision. Therefore, the main concept is the concept “Design Decision”. The transition between states is modelled through the property “hasPreviousDesignDecision”.

If the designer takes a design decision, she does so since a particular goal shall be reached, namely a requirement posed towards the service shall be met. Thus, the justification for a design decision consists in its connection to the requirements which the design decision helps to meet. This is modelled through the concept “Requirement” and corresponding properties (i.e. the properties “isBasedOn” and “requires”) that establish references between a design decision and a requirement.

We identified many types of issues considered in the design process. We observed that decisions fell into one of these four categories:

- design goals, which are principles to be achieved through the decision process and that must be realized before the choice is considered complete;
- design resources, which are the resources --both physical and intellectual-- available to achieve the goal;
• design techniques, which are the strategies for achieving the goals using the design resources available;
• design constraints, which are outside influences that limit the use of resources and strategies to achieve a goal.

Choices made early in the process focused on goals. The next set of choices focused on using resources and techniques to achieve goals. Choices made later in the process focused on specific ways of addressing constraints that affected the ability to implement plans. Although one of the four components dominated a decision, staffs seemed to address each of the four components during each choice. The design process is a sequence of decisions involving each of the four components. Different components dominated decision making at different points in development process.

These elements of a design decision are modelled through the concepts “Goal”, “Resource”, “Technique”, “Constraint” and a set of properties that relate each of them with the concept “Design Decision”. These concepts share the property “hasDescription”, which describes in more detail the concrete instance of the corresponding concept.

The Lifecycle Ontology also models the name of the design decision (the “hasName” attribute), when the design decision is made (the “hasDate” attribute), why it is required (the “isRequiredBy” property), why it is realized (the “isRealisedBy” property) etc. Moreover, it has a reference (modelled through the “hasReference” property) to the Service Ontology or its activity that is related to this design decision.

Moreover, information supporting decision-making, such as cost, relevance, priority, impact, profit, textual description of the reason for a service etc. may also be included. The cost of a design decision modelled through “hasCost” property is used to determine whether it is valid to create/modify a service. The relevance of a design decision (the “hasRelevance” property” describes whether and how it can fulfil the requirements. For example, the costs can be estimated based on the structure of the service ontology that describes the service. If the costs of a design decision are huge and the relevance of a design decision is low, then this solution should be cancelled.

This information might be used to guide domain experts through the evolution process. Further, in order to keep a memory of past decisions, it may be important to record the decisions leading to updates. The “hasReason” property is used to explain the motivation for making a design decision. For instance, an ontology engineer should give an explanation or an example why a certain service should be introduced or its definition has to be changed, etc. This information may be used for learning about decision making process as well as about the competencies of domain experts. The “hasImpact” property allows to value the decisions and to determine the impact on the service. The property “hasProfit” identifies the components of the current and future business profit.

A part of the Lifecycle Ontology is shown in Figure 48. It can be concluded that the Lifecycle Ontology is a description of the service design process, which clarifies which design decisions were taken for which reasons, proves to be valuable for further development and maintenance.
6.3.3 Propagating changes within services

The key process in the change management is the resolution of the changes triggered by the procedure described in the previous section. It has to guarantee that a change is correctly propagated and that no inconsistency is left in the system. If this was left to the managers, the management process would be too error-prone and time consuming – it is unrealistic to expect that humans will be able to comprehend all the existing services and interdependencies between them. For example, in the E-Government domain, an unforeseen and uncorrected inconsistency is one of the most common problems.

Therefore, the change management has to be supported by a tool that improves the efficiency and the quality of this process. In order to develop such a tool, the problem has to be formulated in terms of a formal model. Since our approach is based on the semantic description of services, the formal model requires the specification of the semantics of changes that can be applied to the semantic web services.

For each change introduced in section 6.3.1.1.4, it is required to specify:

(i) the necessary **preconditions**

(ii) the sufficient **postconditions** and

(iii) the possible **actions**.

The **Preconditions** of a change are a set of assertions that must be true, to be able to apply the change. For example, the preconditions for the change `AddServiceSpecialisation(S1, S2)`, which results in the specialisation of the service `S1` in the service `S2`, are: (i) `S1` and `S2` are different services; (ii) `S2` is not an indirect parent (through the inheritance hierarchy) of `S1`; (iii) `S2` is not already defined as a
specialization of $S1$; (iv) for each input/output/preconditions/postconditions of $S1$, there is a corresponding element in $S2$ that is subsumed by the original$^{55}$.

The **Postconditions** of a change are a set of assertions that must be true after applying the change, and it describes the result of the change. For example, the removal of a service results in the fact that this service is not in this service ontology anymore.

The **Actions** are additional changes that have to be generated, in order to resolve the side effects of a change on other related entities. It means that each inconsistency problem is treated as a request for a new change, which can induce new problems that cause new changes, and so on. An inconsistency arises when one of the semantic web service consistency constraints (see section 0) is corrupted. For example, the addition of a service will trigger the addition of an input for this service (i.e. $AddServiceInput$ change), since the consistency constraint $C4$ requires that each service has to have at least one input.

To define the actions for changes introduced in section 6.3.1.1.4, we started by finding out the cause and effect relationship between them. The approach is based on a common technique for the maintenance of the knowledge-based systems [36], which states that dependencies between knowledge have to be represented explicitly. However, while in these systems the dependency graph consists of knowledge elements (e.g. rules in the expert systems), in our system the nodes of this graph are changes. For more details see [39].

Here we define the procedures for the $AddServiceInput(service, input)$ change:

- **Preconditions** – $input \notin Inputs(service)$, where $Inputs$ is a set of all inputs already defined for a service. This is in agreement with the single ontology consistency constraints that ensure the uniqueness of the definition.

- **Postconditions** – $input \in Inputs(service)$, which means that this input is defined for this service.

- **Actions** – $AddSameValues(input, x)$, where

$$\exists s1 \text{ service } \in hasNext(s1) \land x \in Outputs(s1) \land Hc^*(input, x),$$

where $hasNext^{56}$ is a property defined in the Meta Ontology for connecting services in a sequence, $Inputs/Outputs$ represent a set of inputs/outputs defined for a service and $Hc^*$ is already defined in section 6.3.1.2.

A new input might corrupt the $C6$ consistency constraint, since the inputs provided by the end-users are usually defined for the first service in the process flow. To resolve this problem, one has to specify that this input is provided by the output of the previous service. This can be realized as a request for a new change $AddSameValues$, which is an elementary change for the OWL ontologies.

For example, according to the changes in a law, the driving licence verification activity requires fingerprint. This change causes the inconsistency, since the new input

$^{55}$ Note that the first three preconditions are inherited from the $AddSubConceptOf$ ontology change, whereas the last one is specific for this particular change.

$^{56}$ $s2 \in hasNext(s1)$ means that $s2$ is one of the activities that $s1$ precedes. They are connected with a control construct.
hangs. The problem can be resolved by generating the additional change \textit{AddSameValues} between the verification activity and its predecessor. This further induces a new output of the predecessor, i.e. application activity, which can potentially trigger other changes, and so on.

Finally, it is important to note that any change in the domain ontology is resolved automatically by using the existing ontology evolution system [17]. For example, let’s consider that the domain ontology contains the concept “\textit{Person}” and two its specializations: “\textit{Child}” and “\textit{Adult}”. Since there is a special procedure for the passport issuance for the children, this service is a specialization of the standard service passport issuance service. The application service of the service for children requires an additional input (i.e. the parent authorization). The precondition for this application service is that it is required for a child. Let’s now consider that the concept “\textit{Child}” needs to be removed. The ontology evolution system will propagate this change to all ontologies that included the changing ontology. Therefore, the ontology describing the passport issuance procedure for children will also be informed about changes. Since, according to the ontology consistency definition, undefined entities are not allowed, the request for the removal of the corresponding application service will be generated.

6.4 The Conceptual Architecture

In this section we propose the conceptual architecture of the OntoGov Change Management System. We start (see section 6.4.1) with the discussion of the need to cope with changes in ontology-based applications in a more systematic way. Then in section 6.4.2 we explain how to achieve self-managing capabilities in the E-Government domain by introducing the MAPE model.

6.4.1 The need for ontology evolution

An important characteristic of today’s business systems is their ability to adapt themselves efficiently to the changes in their environment, as well as to the changes in their internal structures and processes. The continual reengineering of a business system, i.e. the need to be better and better, is becoming a prerequisite for surviving in the highly changing world. Although changes encompass several dimensions of a system (e.g. people, processes, technologies), most of them are reflected on its IT infrastructure. For example, the establishment of a new department in the organisational structure will require the corresponding changes in the enterprise portal, underlying groupware system, skill management system, etc. Therefore, the adaptability of the implemented IT solutions directly defines the efficiency of a system.

However, building and maintaining long-living applications that will be “open for changes” is still a challenge for the entire software engineering community. Most of today’s management tasks are performed manually. This can be time-consuming and error prone. Moreover, it requires a growing number of highly skilled personnel, making the maintenance of applications costly. It is clear that an ad hoc management of changes in applications might work only for particular cases. Moreover, it can scale neither in space nor in time. Therefore, in order to avoid drawbacks in the long run, the change management must be treated in a more systematic way. It is especially
important for applications that are distributed over different systems. Examples of such applications are E-Government services that enable integration of various, physically distributed knowledge sources differing in the structure and the level of formality.

In order to avoid unnecessary complexity and possible failures and/or even to ensure the realisation of a request for a change, the change management should deal with the conceptual model of such an application. Since ontologies are a key technology for semantics-driven modelling in the OntoGov project, the changes have to be reflected on the underlying ontology. This will improve the speed and will reduce costs of the modification. However, as ontologies grow in size, the complexity of change management increases significantly. If the underlying ontology is not up-to-date, then the reliability, accuracy and effectiveness of the system will decrease significantly.

Since ontologies have to be continually changed, the need for ontology evolution\(^{57}\) is inevitable. The task of the ontology evolution is to formally interpret all requests for changes coming from different sources (e.g. users, internal processes, business environment) and to perform them on the ontology and depending artefacts while keeping consistency of all of them. Figure 49 illustrates the role of ontology evolution in a business system.

Figure 49. The role of an ontology evolution in a business system

Figure 49 depicts three basic sources of changes:

- *the business environment* - the environment in which E-Government system operates can change, thereby invalidating assumptions made when the system was built;

---

\(^{57}\) The word "evolution" merely means "change over time". It implies neither a direction, nor, necessarily, improvement, but merely a change.
• the users of the system - users’ requirements (i.e. requirements of the domain experts) often change after the system has been built, warranting system adaptation;

• the internal processes - the business applications are coupled around the business processes that should be continually reengineered, in order to achieve better performances.

Indeed, the frequently changing business context implies the need to cope with changes in ontology-based business applications in a more systematic way. Firstly, different causes of changes (e.g. changes in the business environment, user’s preferences, internal processes, etc.) have to be uniformly represented, in order to enable their efficient processing. Secondly, the changes have to be consistently resolved in the application, and their effects have to be propagated to all dependent business systems. Moreover, in order to control the resolution of the changes (e.g. the identification and overcoming of undesired changes), the responsible persons have to be able to make appropriate decisions. Finally, the continual business reengineering requires an automatic discovery of new changes by analysing the manner in which the application is used (e.g. the detection of trends in the users’ behaviour). In order to fulfil these requirements efficiently, the managing of the changes in the ontology-based application has to be performed on the level of ontologies themselves. Therefore, the need for an efficient approach to the management of the changes in an ontology (e.g. ontology evolution) is obvious. In the OntoGov project we will develop such an approach for the E-Government domain.

6.4.2 The MAPE model

IBM’s Autonomic Computing initiative [7] attempts to address management problem by providing IT systems with powerful concepts for self-management including new capabilities for self-healing, self-protecting, self-optimising, and self-configuring. The goal is to reduce the burden associated with the management and the operation of IT systems. Autonomic Computing systems just work, repairing and tuning themselves as needed.

Similarly, our goal is to free E-Government public managers from many of today’s evolution tasks. We need a system that is not people-intensive anymore, which would result in decreasing of related management costs. Since autonomic computing systems allow people to concentrate on what they want to accomplish rather than figuring out how to rig the computer systems to get them there, we use the analogy with autonomic computing systems and try to apply their principles on the management of the semantic web services.

Therefore, the OntoGov Change Management System will be realised according to the MAPE (Monitor Analyse Plan Execute) model [7], which abstracts the management architecture into four common functions: collect data, analyse data, create a plan of action, and execute the plan. Indeed, our architecture decomposes the control loop into four parts:

• Monitor – mechanism that collects, organises and filters the data about changes in the law or in the needs of end-users;
• **Analyse** – mechanism that aggregates, transforms, correlates, visualises the collected data, and makes proposals for changes in the ontologies;

• **Plan** – mechanism to structure actions needed to apply the changes by keeping the consistency of the system. The planning mechanism uses evolution strategies to guide its work;

• **Execute** – mechanism to update the code of the underlying web services according to the changes applied in the ontology.

The OntoGov Change Management System based on the MAPE model is shown in Figure 50. We distinguish between two types of changes:

• **top-down changes** that include accounting the modification in the application domain (e.g. changes in laws) or in the business strategy; incorporating additional functionality according to changes in the users’ needs; organizing information in a better way, etc.;

• **bottom-up changes** that are discovered from the analysis of the end-users’ interactions with the E-Government system.

![Figure 50. The conceptual architecture of the management system](image)

The goal of the OntoGov system is to enable the transparency of a problem, which can be achieved through top-down and bottom-up change propagation. So far the changes are initiated and propagated manually, which causes a lot of errors and redundant steps in the change management process. The needs of end-users are generally not taken into account.

Since we distinguish between the top-down and the bottom-up request for a change, two sources can to be monitored. However, in the OntoGov project we focus only on
top-down request for changes, since the tracking the users’ interactions requires the implementation of a portal which is not in the scope of the project.

Therefore, the OntoGov Change Management System will continually monitor only its suppliers (e.g. politicians who define the law) to ensure that it is receiving up-to-date information. Each change in the Legal Ontology will be stored in the Evolution Log of this ontology [11]. This log is based on the formal model of ontology changes (i.e. on the Evolution Ontology that is described in the section 6.3.2.1).

On the other hand, even though it is possible and usable to monitor the customers of the E-Government system (i.e. the end-users) to ensure that the services, which it offers, meet the customers’ requirements, the usage-driven change management will not be done in the OntoGov project. Here we only mention how it can be done. To cover bottom-up changes all end-users’ interactions with the E-Government portal can be tracked in the semantic usage log file. The semantic log should be structured according to the Usage Ontology, and should therefore contain meta-information about the content of visited pages. Based on the analysis of this log, some recommendation for the continual improvement of the E-Government services may be generated.

The task of the change detection phase of the OntoGov Change Management system (cf. Change Detection in Figure 50) is (i) to locate services that are out-of-date and (ii) to determine how to change them. This is described in section 6.3.2. Our goal is to develop the change management system that allows for the change propagation and traceability, contributing in this way to the bridging of decision making with technical realisation. To achieve this goal, it is not sufficient to use ontologies for modelling services and to focus public administrators only on the relevant services. Rather it is required to model the dependencies between different stakeholders that define this business logic in a collaborative way. In order to help public administrators find out the right changes needed to synchronise the service with the law, we develop the so-called Lifecycle Ontology. It describes the information flow and the decision making process in the public administration.

Since the application of a single ontology change can cause the inconsistency effect in the other part of this ontology and all the artefacts that depend on it (e.g. dependent ontologies, applications based on it etc.), the ontology evolution process (cf. Ontology Evolution in Figure 50) has to be applied. It guarantees the transfer of the ontology and dependent artefacts into another consistent state. Indeed, it manages changes in six steps: (i) the process starts with capturing changes either from explicit requirements or from the result of change discovery methods; (ii) next, the changes are represented formally and explicitly; (iii) the semantics of the change phase prevents inconsistencies by computing the additional changes that guarantee the transition of the ontology into a consistent state; (iv) in the change propagation phase, all dependent artefacts (ontology instances on the Web, dependent ontologies and application programmes using the changed ontology) are updated; (v) during the change implementation phase, the required and induced changes are applied to the ontology in a transactional manner; (vi) in the change validation phase, the user evaluates the results and restarts the cycle if necessary. In this phase the specificities of the E-Government domain must be taken into account. This is discussed in section 6.3.1.
Finally, the OntoGov Change Management System has to notify programmers about the necessity to update a code. However, this phase cannot be automated, since a programmer has to modify the code manually.

6.5 Related work

Although the research related to Web Services has drastically increased recently, there are very few approaches that cope with the changes in the process flow of a web service. The management approaches are mainly focused on the composition of a web service from scratch, and neglect the problem of the continual improvement of the service. The change management approaches are mainly focused on re-implementing some software modules [31]. We found two reasons for such behaviour:

1) Since the technology is rather new, the real challenges for the change management are still to come. Indeed, in the workflow community, from which web services are transferring a lot of experiences, workflow maintenance is a well-researched topic;

2) The description of web services lacks a conceptual level on which the reasoning about a compositional model, including the reasons and the methods for its reconfiguration, will be possible. As we already mentioned, the emerging semantic web services approaches introduce such a level, and we give here a short overview of their achievement in the (re)composition.

Workflow

The workflow community has recently paid attention to configurable or extensible workflow systems which present some overlaps with our ideas. For example, the work on flexible workflows has focused on dynamic process modification [32]. In this publication, workflow changes are specified by transformation rules composed of a source schema, a destination schema and of conditions. The workflow system checks for parts of the process that are isomorphic with the source schema, and replaces them with the destination schema for all instances for which the conditions are satisfied. However, the workflow schema contains fewer primitives than an ontology, so that this approach is much less comprehensive than ours. Moreover, the change in the business policy is not treated at all.

The most similar to our approach is the work related to the workflow evolution [27]. This paper defines a minimal, complete and consistent set of modification primitives that allow modifications of workflow schemata. The authors introduce the taxonomy of policies to manage the evolution of running instances when the corresponding workflow schema is modified. However, the authors are focused on the dynamic workflow evolution, which is not the focus of our framework, as we mentioned in the introduction.

Semantic Web Services

Recently, the approaches for the composition of semantic web services have emerged drastically. We discuss only the most relevant to our approach.

In [33] a framework for interactive service composition is presented, where the system assists users in constructing a computational pathway by exploiting the semantic description of services. Given the computational pathway and the user’s task description (i.e. a set of initial inputs and expected results), the system performs a set
of checks (e.g. are all the expected results produced, are all the needed input data provided), in order to ensure the consistency of the resulting model. The checks used in this approach can be seen as a subset of the constraints (see section 3.1.3) we defined for ensuring the consistency. Moreover, since we derive the constraints from the ontology model behind the semantic web services, we can guarantee the completeness and the consistent propagation of the changes.

In [41] the authors present a prototype for dynamic binding of Web Services for the abstract specification of business integration flows using a constraint-based semantic-discovery mechanism. They provide a way of modelling and accommodating scoped constraints and inter-service dependencies within a process flow while dynamically binding services. The result is a system that allows people to focus on creating appropriate high-level flows, while providing a robust and adaptable runtime. Similarly to our approach, they contend that the selection of Web services for a step in a process flow is often not a stand-alone operation, as there may be dependencies on the previously chosen services for the process. They introduce two types of dependencies: description-based and domain constraints, whereas both of them can be easily mapped into our business-knowledge specific constraints that ensure the meaningful order between services in a flow. Additionally, we provide process specific constraints that ensure the consistency of the process flow.

Next, there are several approaches for the automatic composition of semantic web services [37], [28] that drive the design at a conceptual level, in order to guarantee its correctness and to avoid inconsistencies among its internal components. In that context, our approach can be seen as an automatic re-composition of a service driven by the constraint derived from the business environment, domain knowledge and internal structure of a service.

Finally, the main difference between our approach and all the related work is that we base our management framework on the systematic evolution of the model that underlines semantic web services (i.e. several dependent and distributed ontologies). It enables us to be predictive in the management (i.e. we can reason about the consequences of changes in the system), and to expand the framework, whereas the consistency of the managed system is ensured easily.

6.6 Conclusion

In this section, we presented an approach for the management of changes in semantic web services. The approach extends our previous work on the ontology evolution for multiple and distributed ontologies. We considered the E-Government domain, since E-Government services are under the continual adaptation to the political goals of a government and to the needs of the people. Up to now, the changes have been initiated and propagated manually, which causes a lot of errors and redundant steps in the change management process. Our approach enables the automation of the change propagation process, and ensures its consistent execution, since it is based on a formal, logic-based framework for coping with changes. Consequently, we can reason about the change management process, making it very flexible and efficient.

In the future, we want to extend this approach by suggesting changes that can improve services. This can be done (i) by monitoring the execution of E-Government services (e.g. the activity that causes the delay is a candidate for optimization) and/or (ii) by
taking into account the end-users’ complaints (e.g. end-users might not be satisfied with the quality of services, since they have to supply the same information several times).
7 References


## 8 Appendix

### 8.1 SIPOC Model templates

#### 8.1.1 Process template

<table>
<thead>
<tr>
<th>Mapping of a process based on the SIPOC model</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>About the company</strong></td>
</tr>
<tr>
<td><strong>Company name</strong></td>
</tr>
<tr>
<td><strong>Manager</strong></td>
</tr>
<tr>
<td><strong>Telephone</strong></td>
</tr>
<tr>
<td><strong>e-mail</strong></td>
</tr>
<tr>
<td><strong>Process SIPOC model</strong></td>
</tr>
<tr>
<td><strong>Process name</strong></td>
</tr>
<tr>
<td><strong>Process ID</strong></td>
</tr>
<tr>
<td><strong>Owner</strong></td>
</tr>
<tr>
<td>Who is the owner of the process? – What's the name of the employee nominated as the owner of the key process? This person is responsible for process functioning from its beginning to the very end, independently of its partial belonging to different functions and departments. Note that it is about a concrete person that should be available as a contact person for the discussion.</td>
</tr>
<tr>
<td><strong>Sponsor</strong></td>
</tr>
<tr>
<td>Who is the process sponsor? – What is the name of a board member responsible for the key process?</td>
</tr>
<tr>
<td><strong>S</strong> Supplier</td>
</tr>
<tr>
<td>Who is able to initiate the process?</td>
</tr>
<tr>
<td>Who are the suppliers of information and documents?</td>
</tr>
<tr>
<td>Is there a list of authorized suppliers?</td>
</tr>
<tr>
<td>Are the suppliers a part of the system and is a win-win scenario in place?</td>
</tr>
<tr>
<td><strong>I</strong> Input</td>
</tr>
<tr>
<td>Which are the input documents?</td>
</tr>
<tr>
<td>Which is the input information?</td>
</tr>
<tr>
<td>Note that inputs are more related to the data models, whereas suppliers are people who in some way participate in the process (e.g. by triggering it).</td>
</tr>
</tbody>
</table>
### Process

- Where does the process begin? – When do we get the information to initiate this process? Whom do we get the information from to initiate the process? If the process is often repeating, what is the initial activity when starting it?

- Where does the process end? – What is the final activity before the delivery of the service? Is there a customer support upon the delivery? Do we get a feedback from the customers? Is the feedback a part of this or some other process?

- Which are the subprocesses of this process? – Name the subprocesses indispensable for functioning of this process. Are any of the subprocesses connected with some other process? If yes, what are their dependencies?

- Which are the resources (e.g. personnel, materials, equipment, software, etc.) used by the process? Which are the key parameters (i.e. parameters used for the process efficiency and effectiveness evaluation) of the process?

- Which are the connections with the other processes? Are there any interfaces towards other processes, which are taking place in the same, or some other part of the organization?

- Why did you choose to monitor/analyze this process? How well is the process functioning? What do you hear from your customers?

### Output

What is the process output?

Which services? Which documents?

### Customer

Who are the people and organizations that use the output (i.e. a service)?

Do you make a difference between your returning and potential customers?

How is that going to affect external customers?

What are their needs with respect to this?

What do they need, but don’t get?

What do they get, but don’t need?

In which way have you established a feedback relation with the customers?

Do you incorporate a voice of the customer in improvement activities?

Do you statistically process gathered information?

### The impact of other processes

What is the relation with other processes?

Can other processes initiate this process?

Can this process initiate other processes?
Process description and flow diagram
### 8.1.2 Activity template

<table>
<thead>
<tr>
<th>Activities mapping based on SIPOC model</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>About the company</strong></td>
</tr>
<tr>
<td>Company name</td>
</tr>
<tr>
<td>Sector/Department</td>
</tr>
<tr>
<td>Sector manager</td>
</tr>
<tr>
<td><strong>About the process</strong></td>
</tr>
<tr>
<td>Process name</td>
</tr>
<tr>
<td>Owner</td>
</tr>
<tr>
<td>Who is the owner of the process? – What's the name of the employee nominated as the owner of the process?</td>
</tr>
<tr>
<td><strong>Activity description</strong></td>
</tr>
<tr>
<td>Activity name</td>
</tr>
<tr>
<td><strong>A brief activity description</strong></td>
</tr>
<tr>
<td>What is the activity's task? What is the main service it delivers? What is its purpose?</td>
</tr>
<tr>
<td>Input</td>
</tr>
<tr>
<td>Which are the input documents?</td>
</tr>
<tr>
<td>Which is the input information?</td>
</tr>
<tr>
<td>Output</td>
</tr>
<tr>
<td>What is output of the activity?</td>
</tr>
<tr>
<td>Output document?</td>
</tr>
<tr>
<td>Which information?</td>
</tr>
<tr>
<td>Which service?</td>
</tr>
<tr>
<td><strong>Utilized resources</strong></td>
</tr>
<tr>
<td>People</td>
</tr>
<tr>
<td>- Which activity participants are directly serving the customers or are adding value to services, within the process on its way to customers?</td>
</tr>
<tr>
<td>- Which of the participants are difficult to replace because of the extent and depth of the knowledge and skills (necessary for executing the functions) they posses?</td>
</tr>
<tr>
<td>- Which knowledge should a participant possess in order to perform this activity?</td>
</tr>
<tr>
<td>- What are lessons learned from this activity?</td>
</tr>
<tr>
<td><strong>Equipment (Hardware, Software, etc.)</strong></td>
</tr>
<tr>
<td>- Which of these resources directly create new value for</td>
</tr>
</tbody>
</table>
customers or services which are being delivered to them?
- Among these key resources which are expensive to obtain, maintain and install?
- Which of these key resources can turn out to be bottlenecks or in some other way decrease the capacity of customer service?
- Which of these resources are characterized by easy and safe utilization?

We want also to identify resources which have an impact on the precision, promptness, competency, time punctuality, easy-utilization for the customer as well as overall quality of services and safety & ease of use within our processes
- Which databases, third-party software etc. are needed?

<table>
<thead>
<tr>
<th>Functions that activities are performing</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Whom does it concern?</td>
</tr>
<tr>
<td>- Who else should be involved?</td>
</tr>
<tr>
<td>- Who can help you?</td>
</tr>
<tr>
<td>- Which knowledge can help you on that?</td>
</tr>
<tr>
<td>- With respect to this task who are: managers, shareholders, colleagues, personnel, and other important people?</td>
</tr>
<tr>
<td>• WHO is responsible?</td>
</tr>
<tr>
<td>• WHAT is needed?</td>
</tr>
<tr>
<td>• WHERE is it defined?</td>
</tr>
<tr>
<td>• WHY was that done?</td>
</tr>
<tr>
<td>• WHEN was it done?</td>
</tr>
<tr>
<td>• HOW was it done?</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameters to be followed</th>
</tr>
</thead>
<tbody>
<tr>
<td>- What do you monitor?</td>
</tr>
<tr>
<td>- Why do you monitor these indicators?</td>
</tr>
<tr>
<td>- How do you use information?</td>
</tr>
<tr>
<td>- Who is controlling the parameters?</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Relations, signals to the other activities, processes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Is it possible to initiate some other process (activity)?</td>
</tr>
<tr>
<td>Based on the value of which parameter?</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Special cases</th>
</tr>
</thead>
<tbody>
<tr>
<td>- What is the history of this activity? Are there any documents describing the modification of this activity and reasons for doing that.</td>
</tr>
<tr>
<td>- What precedes the activity?</td>
</tr>
<tr>
<td>- What kind of impact is it going to provoke in the future?</td>
</tr>
</tbody>
</table>
- What can go out of control?
- Are you ready to address the special cases?
- Is there any trend in their appearance?
- Which new approaches or functions you expect to initiate?
- In what way are they connected to problems, needs and gaps?
- What can go wrong with the introduction and usage of this new approach?
- How is it possible to escape potential problems by minimizing reactions necessary to be taken once the problems appear?
8.1.3 Process map symbols

8.1.3.1 Process

We suggest representing a process or an activity as a rectangle. In the centre of this rectangle is the name of the process (activity), whereas its ID is in the upper left corner of the rectangle.

```
<table>
<thead>
<tr>
<th>ProcessID</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProcessName</td>
</tr>
</tbody>
</table>
```

Note that an activity is an atomic service, i.e. it is directly invocable (by passing it the appropriate data) and it is executed in a single step. On the other hand, a process is a composite service which implies that it is decomposable into other (non-composite or composite) services. The decomposition can be specified by using control structures such as Sequence, If-Then etc. as described later.

In the rest of this document we firstly define symbols that are used to represent information required by a process or by an activity. Further, we show how the process decomposition can be represented graphically by introducing symbols for the control structure of a process.

A process (or an activity) is a central object on a process map, since resources and documents are attached to it.

8.1.3.2 Resources

We distinguish two types of resources: (1) human resources who perform a process (an activity) and (2) equipment (i.e. hardware, software etc.) that is occupied during the associated process. Note that equipment is needed to perform an activity. However, it is released after finishing this activity.

```
<table>
<thead>
<tr>
<th>Resource</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProcessID</td>
</tr>
<tr>
<td>ProcessName</td>
</tr>
</tbody>
</table>
```

8.1.3.3 Documents

We distinguish three types of documents: input, output and intern documents. Input documents contain concrete information needed to perform an activity. Output documents represent the result of an activity. Intern documents are general documents that have to be used in an activity and they reference e.g. directories, price lists etc. Special types of intern documents are folders, databases, or data-warehouses.

For example, the birthday certification process would have the following documents:
- input document is an birthday application form that contains information such as person ID;
- output documents are a birthday certificate and a receipt that has to be paid;
- intern document is a price list that is used to determine amount needed to be paid.

8.1.3.4 Control structure

8.1.3.4.1 Terminators

We define two specific activities, so called terminators, which represent the start and the end of a process map. There is one and only one “start” element. However, there can be several “end” elements.

8.1.3.4.2 Sequence

Sequence is used to connect processes (activities) or other control constructs in an ordered list.

8.1.3.4.3 “AND”

The goal of the “AND” control construct is to join outputs of several processes (activities), while producing an output signal when there is a signal on each input in this control construct.
8.1.3.4.4 “OR” element

The goal of the “OR” control construct is to join outputs of several processes (activities), while producing an output signal when there is a signal on at least one input in this control construct.

8.1.3.4.5 IF-THEN

The IF-Then control construct is used to broadcast a signal in one of two possible directions depending on a value of a signal.

8.1.3.4.6 Split

The split control structure decomposes a process into a bag of processes (activities) to be executed concurrently.
8.1.3.4.7 Join

The join control structure composes a bag of processes (activities) into a process.

8.2 Ontologies for the modelling of the E-Government services

8.2.1 Process Ontology

```xml
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE rdf:RDF [ 
    <!ENTITY kaon 'http://kaon.semanticweb.org/2001/11/kaon-lexical#'>
    <!ENTITY rdf 'http://www.w3.org/1999/02/22-rdf-syntax-ns#'>
    <!ENTITY rdfs 'http://www.w3.org/2000/01/rdf-schema#'> ]>

<?include-rdf logicalURI="http://kaon.semanticweb.org/2001/11/kaon-lexical" physicalURI="file:/C:/kaon-lexical.xml"?>

<?model-attribute key="OMModel.version" value="138"?>

<rdf:RDF xml:base="file:/Process"
    xmlns:kaon="&kaon;"
    xmlns:rdf="&rdf;"
```
xmlns:rdfs="&rdfs;">

<rdfs:Class rdf:ID="Atomic-Service">
  <rdfs:subClassOf rdf:resource="#Service"/>
</rdfs:Class>

<rdfs:Class rdf:ID="Composite-Service">
  <rdfs:subClassOf rdf:resource="#Service"/>
</rdfs:Class>

<rdfs:Class rdf:ID="Condition">
  <rdfs:subClassOf rdf:resource="&kaon;Root"/>
</rdfs:Class>

<rdfs:Class rdf:ID="ControlConstruct">
  <rdfs:subClassOf rdf:resource="&kaon;Root"/>
</rdfs:Class>

<rdfs:Class rdf:ID="If-Then">
  <rdfs:subClassOf rdf:resource="#ControlConstruct"/>
</rdfs:Class>

<rdfs:Class rdf:ID="Input">
  <rdfs:subClassOf rdf:resource="#Parameter"/>
</rdfs:Class>

<rdfs:Class rdf:ID="Join">
  <rdfs:subClassOf rdf:resource="#ControlConstruct"/>
</rdfs:Class>

<rdfs:Class rdf:ID="Output">
  <rdfs:subClassOf rdf:resource="#Parameter"/>
</rdfs:Class>

<rdfs:Class rdf:ID="Parameter">
  <rdfs:subClassOf rdf:resource="&kaon;Root"/>
</rdfs:Class>

<rdfs:Class rdf:ID="Reference">
  <rdfs:subClassOf rdf:resource="&kaon;Root"/>
</rdfs:Class>

<rdfs:Class rdf:ID="Sequence">
  <rdfs:subClassOf rdf:resource="#ControlConstruct"/>
</rdfs:Class>

<rdfs:Class rdf:ID="Service">
  <rdfs:subClassOf rdf:resource="&kaon;Root"/>
</rdfs:Class>

<rdfs:Class rdf:ID="Split">
  <rdfs:subClassOf rdf:resource="#ControlConstruct"/>
</rdfs:Class>

<rdfs:Class rdf:ID="User-defined-Input">
  <rdfs:subClassOf rdf:resource="#Input"/>
</rdfs:Class>

<rdf:Property rdf:ID="consistsOf">
  <rdfs:domain rdf:resource="#Composite-Service"/>
  <rdfs:range rdf:resource="#Service"/>
</rdf:Property>

<rdf:Property rdf:ID="documentation"
<rdfs:domain rdf:resource="#Service"/>
<rdfs:range rdf:resource="&rdfs;Literal"/>
</rdf:Property>
</rdf:Property>
<!-- Property: hasConcept
<rdfs:domain rdf:resource="#Reference"/>
<rdfs:range rdf:resource="&kaon;Root"/>
</rdf:Property>
</rdf:Property>
<!-- Property: hasCondition
<rdfs:domain rdf:resource="#If-Then"/>
<rdfs:range rdf:resource="#Reference"/>
</rdf:Property>
</rdf:Property>
<!-- Property: hasCost
<rdfs:domain rdf:resource="#Service"/>
<rdfs:range rdf:resource="&rdfs;Literal"/>
</rdf:Property>
</rdf:Property>
<!-- Property: hasFirst
<rdfs:domain rdf:resource="#Composite-Service"/>
<rdfs:range rdf:resource="#Service"/>
</rdf:Property>
</rdf:Property>
<!-- Property: hasInput
<rdfs:domain rdf:resource="#Service"/>
<rdfs:range rdf:resource="#Input"/>
</rdf:Property>
</rdf:Property>
<!-- Property: hasName
<rdfs:domain rdf:resource="#Reference"/>
<rdfs:range rdf:resource="&rdfs;Literal"/>
</rdf:Property>
</rdf:Property>
<!-- Property: hasNext
<kaon:inverse rdf:resource="#hasPreviousControlConstruct"/>
<rdfs:domain rdf:resource="#ControlConstruct"/>
<rdfs:range rdf:resource="#Service"/>
</rdf:Property>
</rdf:Property>
<!-- Property: hasNextControlConstruct
<kaon:inverse rdf:resource="#hasPrevious"/>
<rdfs:domain rdf:resource="#Service"/>
<rdfs:range rdf:resource="#ControlConstruct"/>
</rdf:Property>
</rdf:Property>
<!-- Property: hasNextElseService
<rdfs:domain rdf:resource="#If-Then"/>
<rdfs:range rdf:resource="#Service"/>
<rdfs:subPropertyOf rdf:resource="#hasNext"/>
</rdf:Property>
</rdf:Property>
<!-- Property: hasNextThenService
<rdfs:domain rdf:resource="#If-Then"/>
<rdfs:range rdf:resource="#Service"/>
<rdfs:subPropertyOf rdf:resource="#hasNext"/>
</rdf:Property>
</rdf:Property>
<!-- Property: hasOutput
<rdfs:domain rdf:resource="#Service"/>
<kaon:Label rdf:ID="i-1087497730705-366038769"
  kaon:value="hasNextThenService">
  <kaon:inLanguage rdf:resource="&kaon;en"/>
  <kaon:references rdf:resource="#hasNextThenService"/>
</kaon:Label>

<kaon:Label rdf:ID="i-1087497739958-1117763677"
  kaon:value="hasNextElseService">
  <kaon:inLanguage rdf:resource="&kaon;en"/>
  <kaon:references rdf:resource="#hasNextElseService"/>
</kaon:Label>

<kaon:Label rdf:ID="i-1087830212044-1413423888"
  kaon:value="User-defined Input">
  <kaon:inLanguage rdf:resource="&kaon;en"/>
  <kaon:references rdf:resource="#User-defined-Input"/>
</kaon:Label>

<kaon:Label rdf:ID="i-1087830443425-411333366"
  kaon:value="name">
  <kaon:inLanguage rdf:resource="&kaon;en"/>
  <kaon:references rdf:resource="#name"/>
</kaon:Label>

<kaon:Label rdf:ID="i-1087830450115-1199248003"
  kaon:value="documentation">
  <kaon:inLanguage rdf:resource="&kaon;en"/>
  <kaon:references rdf:resource="#documentation"/>
</kaon:Label>

<kaon:Label rdf:ID="i-1087830493565-67675198"
  kaon:value="hasReferencedSoftware">
  <kaon:inLanguage rdf:resource="&kaon;en"/>
  <kaon:references rdf:resource="#hasReferencedSoftware"/>
</kaon:Label>

<kaon:Label rdf:ID="i-1087830517388-1572814031"
  kaon:value="hasSecurityLevel">
  <kaon:inLanguage rdf:resource="&kaon;en"/>
  <kaon:references rdf:resource="#hasSecurityLevel"/>
</kaon:Label>

<kaon:Label rdf:ID="i-1087830532669-218516495"
  kaon:value="hasCost">
  <kaon:inLanguage rdf:resource="&kaon;en"/>
  <kaon:references rdf:resource="#hasCost"/>
</kaon:Label>

<kaon:Label rdf:ID="i-1087830561249-298932855"
  kaon:value="hasTimeRestriction">
  <kaon:inLanguage rdf:resource="&kaon;en"/>
  <kaon:references rdf:resource="#hasTimeRestriction"/>
</kaon:Label>

<kaon:Label rdf:ID="i-1087830576831-908818421"
  kaon:value="hasReferencedBusinessRule">
  <kaon:inLanguage rdf:resource="&kaon;en"/>
  <kaon:references rdf:resource="#hasReferencedBusinessRule"/>
</kaon:Label>
8.2.2 Service Ontology for the example shown in section 6.3.1.1.4

<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE rdf:RDF [
  <!ENTITY a 'file:/Process#'>
  <!ENTITY kaon 'http://kaon.semanticweb.org/2001/11/kaon-lexical#'>
  <!ENTITY rdf 'http://www.w3.org/1999/02/22-rdf-syntax-ns#'>
]
<?include-rdf logicalURI="http://kaon.semanticweb.org/2001/11/kaon-lexical" physicalURI="file:/C:/kaon-lexical.xml"?>
<?include-rdf logicalURI="file:/Domain" physicalURI="DomainOntology.kaon"?>
<?include-rdf logicalURI="file:/Process" physicalURI="ProcessOntology.kaon"?>

<?model-attribute key="OIModel.version" value="148"?>

<rdf:RDF xml:base="file:/Service"
  xmlns:a="&a;"
  xmlns:kaon="&kaon;"
  xmlns:rdf="&rdf;">
<a:Atomic-Service rdf:ID="NotifyOthers">
  <a:hasInput rdf:resource="#InputRefusal"/>
  <a:hasNextControlConstruct rdf:resource="#Join1"/>
  <a:hasOutput rdf:resource="#OutputPerformDeregistration"/>
  <a:hasPreviousControlConstruct rdf:resource="#Split1"/>
</a:Atomic-Service>

<a:Atomic-Service rdf:ID="NotifyOthers">
  <a:hasInput rdf:resource="#InputRefusal"/>
  <a:hasNextControlConstruct rdf:resource="#Join1"/>
  <a:hasOutput rdf:resource="#OutputPerformDeregistration"/>
  <a:hasPreviousControlConstruct rdf:resource="#Split1"/>
</a:Atomic-Service>

<a:Output rdf:ID="OutputAnnounceMove">
  <a:hasReference rdf:resource="#Reference2"/>
  <a:hasReference rdf:resource="#Reference3"/>
  <a:hasReference rdf:resource="#Reference4"/>
  <a:hasReference rdf:resource="#ReferenceInputAnnounce"/>
</a:Output>

<a:Output rdf:ID="OutputCheckRegistration">
  <a:hasReference rdf:resource="#ReferenceCondition"/>
</a:Output>

<a:Output rdf:ID="OutputNotifyCitizen">
  <a:hasReference rdf:resource="#Reference2"/>
  <a:hasReference rdf:resource="#ReferenceCondition"/>
  <a:hasReference rdf:resource="#ReferenceInputAnnounce"/>
</a:Output>

<a:Output rdf:ID="OutputPerformDeregistration">
  <a:hasReference rdf:resource="#Reference2"/>
  <a:hasReference rdf:resource="#Reference4"/>
  <a:hasReference rdf:resource="#ReferenceInputAnnounce"/>
</a:Output>

<a:Output rdf:ID="OutputRefusal">
  <a:hasReference rdf:resource="#Reference2"/>
  <a:hasReference rdf:resource="#Reference3"/>
  <a:hasReference rdf:resource="#ReferenceInputAnnounce"/>
</a:Output>

<a:Output rdf:ID="OutputUpdateRegister">
  <a:hasReference rdf:resource="#ReferenceCondition"/>
</a:Output>

<a:Atomic-Service rdf:ID="PerformDeregistration">
  <a:hasInput rdf:resource="#InputRefusal"/>
  <a:hasNextControlConstruct rdf:resource="#Split1"/>
  <a:hasOutput rdf:resource="#OutputPerformDeregistration"/>
  <a:hasPreviousControlConstruct rdf:resource="#Alternative"/>
</a:Atomic-Service>

<a:Reference rdf:ID="Reference2" a:hasName="Name">
  <a:hasConcept rdf:resource="file:/Domain#Person"/>
  <a:hasProperty rdf:resource="file:/Domain#hasLastName"/>
</a:Reference>

<a:Reference rdf:ID="Reference3"
<a:Reference rdf:ID="Reference4" a:hasName="To Municipality">
  <a:hasConcept rdf:resource="file:/Domain#Person"/>
  <a:hasProperty rdf:resource="file:/Domain#hasAddress"/>
</a:Reference>

<a:Reference rdf:ID="ReferenceCondition">
  <a:hasConcept rdf:resource="file:/Domain#Person"/>
  <a:hasProperty rdf:resource="file:/Domain#isRegistered"/>
</a:Reference>

<a:Reference rdf:ID="ReferenceInputAnnounce" a:hasName="First Name">
  <a:hasConcept rdf:resource="file:/Domain#Person"/>
  <a:hasProperty rdf:resource="file:/Domain#hasFirstName"/>
</a:Reference>

<a:Atomic-Service rdf:ID="Refusal">
  <a:hasInput rdf:resource="#InputRefusal"/>
  <a:hasOutput rdf:resource="#OutputRefusal"/>
  <a:hasPreviousControlConstruct rdf:resource="#Alternative"/>
</a:Atomic-Service>

<a:Split rdf:ID="Split1">
  <a:hasNext rdf:resource="#NotifyOthers"/>
  <a:hasNext rdf:resource="#UpdateRegister"/>
  <a:hasPrevious rdf:resource="#PerformDeregistration"/>
</a:Split>

<a:Atomic-Service rdf:ID="UpdateRegister">
  <a:hasInput rdf:resource="#InputRefusal"/>
  <a:hasInput rdf:resource="#InputUpdateRegistry"/>
  <a:hasNextControlConstruct rdf:resource="#Join1"/>
  <a:hasOutput rdf:resource="#OutputPerformDeregistration"/>
  <a:hasOutput rdf:resource="#OutputUpdateRegister"/>
  <a:hasPreviousControlConstruct rdf:resource="#Split1"/>
</a:Atomic-Service>

<kaon:Label rdf:ID="i-1087497157097-1831452776" kaon:value="Example">
  <kaon:inLanguage rdf:resource="&kaon;en"/>
  <kaon:references rdf:resource="#Example"/>
</kaon:Label>

<kaon:Label rdf:ID="i-1087497169344-904629592" kaon:value="AnnounceMove">
  <kaon:inLanguage rdf:resource="&kaon;en"/>
  <kaon:references rdf:resource="#AnnounceMove"/>
</kaon:Label>

<kaon:Label rdf:ID="i-1087497183664-1936365229" kaon:value="CheckRegistration">
  <kaon:inLanguage rdf:resource="&kaon;en"/>
  <kaon:references rdf:resource="#CheckRegistration"/>
</kaon:Label>
<kaon:Label rdf:ID="i-1087497203111-655030372" kaon:value="Refusal">
  <kaon:inLanguage rdf:resource="&kaon;en"/>
  <kaon:references rdf:resource="#Refusal"/>
</kaon:Label>

<kaon:Label rdf:ID="i-1087497213156-919726355" kaon:value="UpdateRegister">
  <kaon:inLanguage rdf:resource="&kaon;en"/>
  <kaon:references rdf:resource="#UpdateRegister"/>
</kaon:Label>

<kaon:Label rdf:ID="i-1087497244530-2026743844" kaon:value="PerformDeregistration">
  <kaon:inLanguage rdf:resource="&kaon;en"/>
  <kaon:references rdf:resource="#PerformDeregistration"/>
</kaon:Label>

<kaon:Label rdf:ID="i-1087497256427-385625607" kaon:value="NotifyOthers">
  <kaon:inLanguage rdf:resource="&kaon;en"/>
  <kaon:references rdf:resource="#NotifyOthers"/>
</kaon:Label>

<kaon:Label rdf:ID="i-1087497275664-2012317095" kaon:value="NotifyCitizen">
  <kaon:inLanguage rdf:resource="&kaon;en"/>
  <kaon:references rdf:resource="#NotifyCitizen"/>
</kaon:Label>

<kaon:Label rdf:ID="i-1087497512253-1164485090" kaon:value="Alternative">
  <kaon:inLanguage rdf:resource="&kaon;en"/>
  <kaon:references rdf:resource="#Alternative"/>
</kaon:Label>

<kaon:Label rdf:ID="i-1087498080834-2031638457" kaon:value="Seq1">
  <kaon:inLanguage rdf:resource="&kaon;en"/>
  <kaon:references rdf:resource="#Seq1"/>
</kaon:Label>

<kaon:Label rdf:ID="i-1087498221681-1421164840" kaon:value="Split1">
  <kaon:inLanguage rdf:resource="&kaon;en"/>
  <kaon:references rdf:resource="#Split1"/>
</kaon:Label>

<kaon:Label rdf:ID="i-1087498296416-613083054" kaon:value="Join1">
  <kaon:inLanguage rdf:resource="&kaon;en"/>
  <kaon:references rdf:resource="#Join1"/>
</kaon:Label>

<kaon:Label rdf:ID="i-1087498414932-215148639" kaon:value="InputAnnounceMove"/>
kaon:value="OutputRefusal">
  <kaon:inLanguage rdf:resource="&kaon;en"/>
  <kaon:references rdf:resource="#OutputRefusal"/>
</kaon:Label>
<kaon:Label rdf:ID="i-1087500561467-1607592032"
  kaon:value="OutputPerformDeregistration">
  <kaon:inLanguage rdf:resource="&kaon;en"/>
  <kaon:references rdf:resource="#OutputPerformDeregistration"/>
</kaon:Label>
<kaon:Label rdf:ID="i-1087501185400-496210962"
  kaon:value="OutputUpdateRegister">
  <kaon:inLanguage rdf:resource="&kaon;en"/>
  <kaon:references rdf:resource="#OutputUpdateRegister"/>
</kaon:Label>
<kaon:Label rdf:ID="i-1087501216855-1540026976"
  kaon:value="OutputNotifyCitizen">
  <kaon:inLanguage rdf:resource="&kaon;en"/>
  <kaon:references rdf:resource="#OutputNotifyCitizen"/>
</kaon:Label>
<kaon:Label rdf:ID="i-1087501428763-466190614"
  kaon:value="InputUpdateRegistry">
  <kaon:inLanguage rdf:resource="&kaon;en"/>
  <kaon:references rdf:resource="#InputUpdateRegistry"/>
</kaon:Label>
<rdf:RDF>