Semantic MDA for E-Government Service Development

Peter Salhofer
FH JOANNEUM
University of Applied Sciences Graz
peter.salhofer@fh-joanneum.at

Bernd Stadlhofer
FH JOANNEUM
University of Applied Sciences Graz
bernd.stadlhofer@fh-joanneum.at

Abstract

This paper presents an approach to apply the principles of Model Driven Architecture (MDA) combined with a semantic model to the creation of E-Government services. As a result, these services will be available as semantic web services that can be searched for meeting the current goal or desire of a citizen. For this approach to be successful it is important that all required artifacts are automatically generated from the model and that there is no need for manual coding. This ensures short development cycles and fast and easy adaption to shifting requirements. Besides this, the quality and capabilities of the services provided will be significantly improved.

1. Introduction

Currently public agencies are facing significant budget cuts. As a result they have to make sure that scarce resources are used most effectively and efficiently. In the field of ICT this means that every new investment has to keep its value for the organization for as long as possible. This requires for systems and infrastructures that are capable of being adapted to shifting requirements at low costs and with almost no effort. At the architectural level Service Oriented Architecture[1] (SOA) is an approach that allows for the integration of otherwise de-coupled services. Consequently a software system that provides particular services can be replaced without causing an enormous impact on the rest of the ICT landscape. This facilitates the adoption of new technologies and reduces the risk of vendor lock-in situations.

Also in the field of software engineering exist several approaches in order to reduce the development effort for software systems. One of these approaches is known under the term agile development[2]. This encompasses a set of methodologies, technologies and tools that are emphasizing on the fast production of code and thereby trying to minimize the need for analysis, design and documentation tasks1. In relative short development iterations more and more parts of the system are implemented in close cooperation with the customer. This ensures compliance with user requirements and early availability of deliverables.

Model Driven Architecture is another approach that also tries to reduce the development effort for software systems. However, in contrast to agile development it tries to eliminate or at least to minimize the coding phase of a development project. Instead it focuses on the creation of models that should be turned into code automatically by code generators.

Yet another technology that does not seem to have any obvious relations to the before mentioned ones is known under the term semantic web[3]. For years semantic technologies were said to revolutionize the web but for the time being the adoption rate is rather low. The basic idea is to make the content of the web understandable for machines via the creation of semantic knowledge bases called ontologies. This should allow software systems – so called agents – to automatically figure out how to accomplish complex tasks.

In this paper we want to show, how a combination of MDA principles and semantic technologies can be used to comprehensively provide E-Government services with minimal effort and high flexibility. Thus, the next few chapters will briefly introduce the methodologies and technologies used. Then we will present our approach to provide services solely based on semantic models. Finally we will point out the differences to other similar efforts and will summarize the benefits of our approach.

2. Model Driven Architecture (MDA)

MDA was initially proposed by the Object Management Group (OMG) in 2001:

“The MDA defines an approach to IT system specification that separates the specification of system functionality from the specification of the implementation of that functionality on a specific technology platform. To this end, the MDA defines an architecture for models that provides a set of

---

1 http://www.agilemanifesto.org/
guidelines for structuring specifications expressed as models.”[4]

Thus the core idea is to create a model of a system that only represents its functionality but is not influenced by any technological platform. This model is called the Platform Independent Model (PIM). Once the functionality of a system is completely modeled – MDA uses UML as its modeling language – the next step in the development process is the creation of the so called Platform Specific Model (PSM). In contrast to the PIM this model now holds information about how the functionality is mapped to components and capabilities of the target platform. In an ideal model driven development environment, the PSM is turned into source code by a code generator.

MDA wants to improve the development of a system by automizing the model transformation process and by enabling re-use of models. Based on this concept modeled requirements can easily be implemented on different technological platforms, simply by transforming them into an appropriate PSM.

The PIM is the most abstract model in MDA. According to ISO 10746-1 abstraction is “the process of suppressing irrelevant detail to establish a simplified model, or the result of that process”[5]. However, when turning an abstract model into a more concrete one, information that was previously considered to be irrelevant now becomes relevant again. MDA supports two options for adding the information necessary for model transformation. One is called marking and the other one is called metamodel mapping. In the first approach also known as model instance mapping the PIM is extended with additional model elements called marks. These marks represent concepts of the PSM and provide additional hints for the transformation process. The result of marking is a so called marked PIM, which is than turned into a PSM. Metamodel or model type mapping creates a mapping between elements of the PIM and PSM metamodels. MDA models are based on MOF[6], which basically means that there exist meta-models for all MDA models that in turn are based on a common meta-meta-model. This allows for creating a mapping between the PIM and PSM metamodel that determines the transformation between elements of these models.

Although the MDA concept seems to improve the software development process it is still not widely adopted. One of the problems is the fact that still significant programming is needed in the development cycle. This leads to problems when – due to shifting requirements – the model needs to be changed, which in turn results in re-generated code that may not work with the already manually created one. On the other side, there is some evidence that MDA approaches that are tailored for particular problem domains are more successful[7][8]. Domain specific models (DSM) are easier to maintain resulting in less modeling and mapping errors[9].

3. Semantic Web Services (SWS)

Semantic Web Services are typically extensions to conventional web services[10]. Whereas web services already come with a comprehensive syntactic description by the means of a WSDL[11][12] document, semantic web services add extra semantic information in order to support[13]:
- automatic web service discovery
- automatic web service invocation
- automatic web service composition and interoperation

To achieve these goals semantic web services have to describe their functionality in a way that allows a software agent to figure out whether the service is relevant for achieving a particular result or not. Therefore semantic web services describe their IOPE's (inputs, outputs, preconditions and effects). The input describes which information and data is needed by the service. The output element describes what is delivered by the service. The preconditions describe some facts about the state of the world that have to hold true in order for the service to deliver the expected result (e.g. the credit card used has to be valid). The effect states those facts that will be true once the service was successfully executed (e.g. the applicant will be the owner of a driving license and therefore will be allowed to drive a car).

To model these aspects there has to exist at least one ontology that defines all relevant concepts or classes of the application domain (e.g. person, applicant, credit card, …). Furthermore, to enable software agents to automatically invoke the actual web service, some description of how these concepts refer to the individual message parts that have to be sent to the web service endpoint is needed. This type of mapping is called service grounding. If a web service offers several operations that have to be called in certain sequences (e.g. login, place order, …) a description of useful or valid call sequences – called choreography – is needed as well. There exist several frameworks that support the creation of semantic web services like OWL-S[14], WSMO[15], SAWSDL[16] or WSDL-S[17].
4. Combining MDA and SWS

The research question behind the approach presented in this paper is whether semantic models could be used as core models for an MDA-like approach to the efficient development of E-Government services. The goal is to provide a mean capable of creating new services with minimal effort that can easily be adapted to shifting requirements as well. This should be achieved by utilizing core MDA principles, that emphasize on model-transformation and try to avoid the necessity of programming.

As already pointed out in section 2 MDA is still not adopted on a large scale since there is still significant need for manual coding that results in complex round-trip engineering procedures whenever the model has to be changed during development to keep everything in sync. Thus, one key success factor for any new approach based on the MDA idea is to eliminate the need for coding especially when this code will interfere with automatically created code. Furthermore, empirical evidence shows that MDA is more successful when it is specifically tailored for a particular domain. This allows for using domain specific model elements that can comprehensively be converted into code due to their clear meaning in the current context. However, what are the specifics of E-Government – more specifically in the context of the Government-to-Citizen (G2C) interaction – that will lead to a domain specific approach? As indicated in Figure 1 the business protocol for consuming a public electronic service is rather simple. There is one single entry-point representing the submission of an application. The electronic service might be synchronous, meaning that the final deliverable of the public procedure (e.g. a permanent parking permit) is immediately returned by the service, or asynchronous, in which case the service is likely to return some acknowledgement of receipt, whereas the actual deliverable is sent later using some (electronic) delivery service.

![Figure 1: Public services represent interfaces to the public agencies' business processes.](image)

Therefore, every public service can be implemented as a single (web) service operation without the need for a complex service choreography. Despite the simple business protocol another thing that is specific to public services are the different preconditions that have to be met before a service can be used, which distinguishes E-Government from other electronic services like E-Commerce. From the citizen's point of view this involves questions like “Which services are available/suitable for my specific situation?” or “Am I eligible to use this service?”. We will refer to the series of tasks that have to be accomplished before the appropriate service can be used as the service identification phase. Thus, for our approach to be successful it is necessary that the MDA based system supports the following:

- Service Identification: Identifying those services that are available or necessary for a particular situation.
- Service Utilization: Invoking the identified service(s) by collecting and submitting the required information.
- Service Implementation: Providing a service endpoint and a process implementation.

5. The Model

MDA proposes the use of UML as its preferred modeling language. UML provides a broad range of diagrams that can be used to model different aspects of a system. Since it is based on MOF, the same meta-model instances can be used in different diagrams and either by modifying these meta-models or by using so called UML profiles, the language can be adapted to particular domains. Thus, why should we use a semantic model represented by ontologies rather than UML? The motivation is that semantic models are extremely expressive when modeling structural knowledge. Thus, instead of having a model that is scattered over a set of different diagrams one comprehensive semantic model can be used. This

![Figure 2: Core elements of the service meta-model](image)
should facilitate modeling as well as maintenance of a model. On the other side, however, semantic technologies fall short in capturing procedural knowledge. This means that it is hard to model dynamic aspects like processes or flows. In the E-Government domain, the impact of these shortcomings is not too extensive, since the interaction pattern is simple and almost identical for all public services as pointed out in the previous section.

Nevertheless a meta-model that is a modeling guideline for all public service models is needed. Literature research showed that there is only one recommendation for a general E-Government reference model called the Governance Enterprise Architecture for Public Administration (GEA-PA)[18]. Thus, we have based our own meta-model on this recommendation. The core element is the so called ConstrainedPublicService class (see Figure 2), which represents an available public electronic service. As a sub-concept of PublicService it specifies effects and outputs, which can be used by software agents to determine the relevance of a service. Additionally every service is linked to a desire. A desire reflects the citizen's point of view and represents something a citizen would like to do or to achieve. A typical desire would be something like “I would like to open a restaurant.”. Additionally every service specifies the required input and logical preconditions that have to be met in order to use it.

Apart from the meta-model that is kept in a separate ontology, the framework comes with a few general purpose ontologies that specify those classes that might be needed for all different types of public services (like persons, governmental levels and structures, …). Therefore the actual model of a service consists of instances of the meta-model ontology (e.g. an instance of ConstrainedPublicService) and some domain specific ontologies that describe necessary elements in the context of the current public service (e.g. different types of businesses, professions and activities in the case of the businesses registration process). Classes defined in the common and the domain-specific ontologies are used in the service model to specify input, output, effects and desires (see Figure 3).

### 6. The Implementation

The current implementation of the framework is based on experience gained from a feasibility study[19]. In this prototype an OWL[20] based model was created that was then transformed into a runnable application based on XML and XForms. The problem with this approach was that there was still some need for manual modifications of the generated artifacts and several transformation steps were needed to incorporate semantic reasoning on user provided data. This has led to significant reengineering and technology changes. Currently WSML[21] is used as the modeling language instead of OWL. WSML offers language variants (e.g. WSML-Rule) that are based on logical programming, which support an intuitive way to model constraints and show better computational complexity than description logics based languages. While the prototype was generating a running system, the new version provides a runtime infrastructure that directly interprets the semantic model. This infrastructure is built around a semantic reasoner so that all relevant data is kept in the information space of the reasoner. This allows for utilizing the expressive power of semantic models during runtime without the need for additional data transformations. Consequently no additional artifacts are created or needed since the entire system behavior is based on the semantic models of the services offered.

#### 6.1. Service Identification

Service identification is based on so called desires. Every public service might be related to several desires and also every desire might be referenced by several services that could meet this desire. Whether a service actually matches a specific desire or not is determined in the service identification phase. In the previous section we used the example “I would like to open a restaurant”. This is a relatively concrete desire that can be used by the system to look up appropriate services. The question that remains is, how do we get our citizens' desires. There are various options ranging from natural language processing, allowing users to express their needs in an open form, semi-natural/structured language support where possible options are restricted to some patterns or simple selection-lists where users can choose from a list of
available desires. In our system we have adopted a structured approach based on so called desire templates. The motivation for this was that natural language processing is significantly more complex, and evidence from another project using it, indicates that users need several attempts in formulating their needs before they get the intended result[22]. A desire or goal template is an abstract formulation for an entire category of goals. In our restaurant example, such an abstraction would be “I would like to open a business”. This desire can be used by all services that are needed for business registration. So how do we get from this abstract template to the more concrete one indicating the wish to open a restaurant? In our algorithm a goal template can refer to an arbitrary number of concepts/classes of the domain-specific or common ontologies. In this example the template refers to the concept business. In the ontologies there might exist various sub-concepts of the class business. In fact every concept that is further specified by sub-concepts is considered to be abstract, whereas every leaf in such a taxonomy is considered to be concrete. A desire template is fully specified when it refers to concrete concepts only. Thus, all abstract concepts of a desire template have to be replaced by one of their concrete sub-concepts. In our example, business has to be replaced by a more specific sub-concept like restaurant. The system supports users in specifying their desires either via lists of sub-concepts or automatic classification based on axioms. The set of concrete concepts associated with a desire is used to identify the services appropriate for the given situation. A detailed description of the service identification algorithm can be found in [23].

6.2. Service Utilization – Semantic Forms

Every service modeled in the ontologies can be directly accessed via the run-time infrastructure. If the service identification component was used to look it up, any data that was gathered throughout the identification phase is re-used. Whenever a service is about to be invoked, the infrastructure figures out which information is needed by the service by analyzing its semantic description. The required input is typically a set of (abstract) concepts. The goal of the information elicitation phase is to get instances for all the (concrete sub-)concepts in the input set with fillers for all mandatory attributes. The electronic forms necessary to collect these attribute values from the user are dynamically rendered based on the semantic model.

The form creation process starts with the attributes of the first root level concept in the set of input concepts. If an attribute is of type concept, a separate sub-form is recursively created for this attribute. If it is of a simple type (e.g. string, number, date,...) the user can provide a value for this property. A typical scenario is shown in Figure 4. There is a root concept called BuildingPermitApplicationRequest.

![Figure 4: Constructing an electronic form](image)

This concept contains a property called applicant which is of type Person. Since the person concept is refined by the two sub-concepts CorporateBody and PhysicalPerson it is abstract and has to be replaced by one of its leafs. Thus, the user has to determine the actual type of person. Then an instance of the selected concept is created and its attributes are rendered in a form. If any of these attributes was of type concept a button for forking off a concept-specific sub-form would be rendered. Every time an instance is completely specified by the user, it is registered with the reasoner and therefore checked against all the constraints and axioms of the ontologies. This allows for extreme complex plausibility and consistency checks. If an instance passes these checks, the user is able to proceed with specifying the next concept until all the required input is complete and the application can be submitted to the service end-point.

6.3. Marking the Semantic Model

As already pointed out in section 2 a PIM does not always contain all the information necessary to turn it into a PSM or running system. This is also true for our semantic PIM. For example an attribute of type string is typically rendered as an input field. However, if a string-typed attribute should capture some longer text, a text area element would be more appropriate. MDA provides two mechanism to bridge this type of information gap: metamodel mapping and marking. Since in our case the PIM is directly interpreted without the generation of anything like a PSM we have
decided to adopt marking. For adding some extra meta-information to a semantic model, WSML provides so called annotations. These are sets of properties that are not considered by a reasoner but can be used by our run-time environment. Most of the supported marks are defined in a separate utility ontology (like definitions of the HTML appearance or the size of an attribute). There are also marks that force a check, whether a date is in the past (e.g. for a date of birth) or in the future. Constraints like this are not allowed in ontologies, since modeled and derived facts have to be invariant over time.

A very powerful set of marks is represented by the auxiliary service ontology. This ontology provides means to incorporate external services into the form-generation and processing flow (see Figure 5).

![Figure 5: The auxiliary service ontology](image)

These services can be used to provide lists of values from external datasources or can be used to validate concept instances. A typical use case is the integration of a service that provides street addresses. This service can be wrapped inside a domain provider service that is used to render autocompletion fields and/or wrapped into a validation service that can be used to check the correctness of a provided address against a database. All available instances of auxiliary services are registered with the reasoner. Every such instance can be considered a semantic (web) service in its own, since it holds a semantic description of its functionality, its input and its output. If such a service is also an instance of ScopedAuxiliaryService, its applicability can be described in a detailed, E-Government-specific way. A ScopedAuxiliaryService defines the two attributes called appliesTo and validWithin. The first attribute refers to an instance of AdministrationLevel and the latter to an instance of GovernmentalEntity. A service with the entries appliesTo=Municipality, validWithin=TexasState could be used by all municipalities in the State of Texas.

If any of these services should be used, the appropriate property or concept has to be annotated (e.g with annotations like ValuesProvidedByService or ValuesValidatedByService). These annotations do not include a reference to any specific auxiliary service but only indicate that a service should be used for this property or concept. If an element is annotated with such a value, the system tries to look-up a service with the required functionality (e.g a service that provides street addresses). If such a service exists, it is used otherwise the annotation is ignored. If no such service was found but becomes available any time later, it will be used as soon as it is registered with the reasoner. Thus, auxiliary services and properties or concepts that prefer to use them are configured dynamically based on their semantic descriptions and availability.

Generally the semantic model's capabilities can be significantly extended by marks, so that there is no need for additional programming.

6.4. Implementing the Service-Endpoint

Generally the mapping of the semantic description of a web service and its syntactic description (usually its WSDL[11] document) is called grounding[24][25]. As argued in section 4 an electronic public service represents an interface to a public procedure. Consequently, implementing such a service means realizing this procedure's business process. Since semantic frameworks are typically based on description logics or logic programming, they are not very well suited to describe processes. Thus we decided to propose the use of BPEL(4WS) for implementing the actual service endpoints:

“BPEL4WS should define business processes that interact with external entities through Web service operations defined using WSDL 1.1 and that manifest themselves as Web services defined using WSDL 1.1...”[26]

Thus, a BPEL process exposes itself as a web service, which perfectly meets our requirements. A schematic overview of the default grounding mechanism used is given in Figure 6.

The overall idea of our approach is to create all elements needed to offer an E-Government service automatically based on a semantic PIM. When it comes to grounding, this means that all artifacts necessary to provide an operational web service have to be generated as well. One key issue is the generation of XML schema types that represent the concepts in the ontologies and can be used as message types for the web services' operations. As a matter of fact it is virtually impossible to translate ontologies into corresponding XML schemes without compromising. This is mainly due to the different means that are available to express class/type hierarchies and
therefore inheritance. Whereas WSML supports multiple-inheritance, which can either be expressed explicitly using the subConceptOf construct or implicitly via axioms there is fairly limited support for inheritance in XML schema.

![Diagram of inheritance](image)

**Figure 6: Default grounding of the semantic service model to a WSDL based web service implemented as BPEL process.**

Although there exists some literature on mapping ontologies to XML schema[27] we wanted to create a transformation that preserves the concept hierarchy in the ontologies. This is motivated in the goal being able to use abstract super-types in WSDL documents as well. For example, it shall be possible to use the type person as a message type in a WSDL document but to send an instance of one of its sub-types (compare Figure 4) to the service endpoint. Thus, abstract super-types can be used as placeholders for their concrete sub-types. XML schema provides a type extension mechanism that can be used to explicitly define abstract super-types that have to be extended to concrete sub-types[28]. Multiple inheritance was imitated by adding all properties of the additional super-concepts to the sub-type in XML-schema. Although this means that the inheritance relation gets lost, it still allows to translate semantic instances into the corresponding XML-type, since all inherited properties will be available and can be automatically mapped. Besides a set of XML schema documents – there will be one schema document for every ontology namespace – for every service one WSDL document is created. This WSDL document contains one operation representing the public service (compare Figure 6). So when all required data was collected by the user and checked for consistency by the reasoner and probably external validation services these instances are automatically converted into XML that is based on the previously created schemes and that will be sent to the service endpoint.

Although the generated WSDL file can be used by any contract-first web service implementation approach, we recommend the use of BPEL as already pointed out at the beginning of this section. BPEL perfectly fits into a service oriented environment and eliminates the need for programming. Assuming that there exists a service repository representing the required business functionality a process can be easily composed using appropriate tooling. Figure 7 shows a BPEL process designed using Netbeans'2 SOA modules. This fragment is taken from a business registration process that creates a new file in a document managed system, adds the data that comes with the request, produces a new task in the workflow system and associates the file with this task.

![BPEL process diagram](image)

**Figure 7: Fragment of a BPEL process**

After that a digitally signed acknowledgement of receipt is returned as the web service operation's output message. All of this is a design and modeling task instead of programming and thus complements the overall MDA-based approach presented here.

7. Similar Approaches

There exists a variety of projects that also try to leverage the expressive power of semantic technologies in the context of E-Government. One of these projects is SemanticGov[3]. This project has contributed to the development of WSML, which is used in our approach as well. Besides this, the GEA-PA reference model is another outcome of this project, that has inspired our approach to model services. The

2 www.netbeans.org

3 www.semantic-gov.org
SemanticGov project provides a mechanism for the service identification phase. This approach is also based on templates that refer to trees of concepts. These trees, however, are not taxonomies like in our case, but explicitly modeled structures. All concepts that are not leafs in these trees are also considered to be abstract. To enable a user to traverse the tree towards a leaf, questions that are model elements as well have to be answered, whereas our refinement process is based on reasoner-supported specialization that does not need pre-defined questions. The focus of SemanticGov lies on the promotion of already existing web services by turning them into semantic web services and not on the creation of new (semantic) web services[29].

TERREGOV⁴ is a project that tries to combine OWL-S semantic web services and BPEL. The focus here is on using BPEL to orchestrate the interaction of different services whereas we expect BPEL to provide a choreography for the internal implementation of an electronic public service.

A very interesting approach is used in a project called SUPER⁵ (Semantics Utilized for Process management within and between Enterprises). This project is not specifically related to E-Government but also combines BPEL and WSML. One important outcome is a specification of BPEL for Semantic Web Services (BPEL4SWS)[30]. Extensions to the BPEL XML dialect allow for the integration of WSML based semantic web services into BPEL processes, whereas our approach proposes the use of BPEL for implementing web services that are operational endpoints for semantically modeled services. BPEL4SWS uses one of WSML’s standard grounding mechanisms that requires explicit mapping between existing XML schema types and WSML concepts, which hampers the use of abstract types in the WSDL description of a service.

8. Summary and Conclusions

As pointed out in section 2 MDA tries to improve the re-use of models and also tries to minimize development effort by automatically generating applications based on PIMs and PSMs. As pointed out in section 2, the application of MDA seems to be more successful when it is adapted to particular problem domains. In our example this domain is E-Government, which allows for simple service models as shown in section 4. By using a model that is expressed in an ontology modeling language like WSML and by directly interpreting this model, the expressiveness of formal semantic methods can be used. In our framework this spans all relevant phases in service utilization, starting with the identification of relevant services based on a citizen’s desire, the collection of information that is needed by the service and ends with the actual implementation and execution of the public service. Via the possibility of marks that can be added to the logical model, state-of-the-art web front-ends will be automatically and dynamically created without the need for manual intervention. On the other side creating a formal model like an ontology is a non-trivial task. This, however, pays-off by the huge variety of functionality that becomes available with the model. Providing new services means to define the relevant concepts of the application domain and to create instances of PublicServices for every new service. The system will automatically create XML schema and WSDL documents. These documents can be imported into a BPEL editor and the process behind the service interfaces can be modeled. Thus, creating a new high quality public service is a pure modeling task that does not require any programming. This reduces implementation time and costs, which represents a huge advantage for public agencies that are already suffering from constantly declining budgets. Besides this, every change to the model will become immediately effective in the user interface. Thus, changes due to shifting requirements or modified regulations can be performed within minutes or hours, again without any programming. This keeps maintenance costs at a minimum.

What makes this approach unique when compared to other similar projects is the fact that it covers all phases of the service provisioning process and that it provides a purely model-based solution. This includes service identification, service enactment and service execution. The modeling process focuses on the actual public service, its required input information and its business rules. Domain and Ontology experts can therefore concentrate on describing the procedures and do not have to worry about framework specific artifacts. The resulting formal model helps to make public procedures transparent since all decision rules are made explicit via axioms.

From a users point of view, the use of semantic models as a basis for interactively rendered forms provides a whole new user-experience. With every decision of the user to replace an abstract concept by a concrete sub-concept, the system adapts to the specific situation and renders input dialogs that perfectly represent the current specific situation. Thus, every

---

⁴ http://www.terregov.eupm.net
⁵ http://www.ip-super.org
form adapts to the user's situation leading to a highly personalized system.

One of the biggest potential drawbacks of the approach presented here is the inherent complexity of formal semantic methods. Although we have tried to support the modeling process by providing a relatively simple meta-model that guides designers when defining new services, there is still some significant mathematical background needed. We have to admit that a model can hardly be created solely by a business engineer. Either specific training or the support of an ontology expert is needed. Thus, the simplification of the modeling process is one future field of research. This could for example be achieved by providing a new simple domain specific language or by better tooling supporting a pure graphical modeling approach.

Our solution has already left the trial phase and is currently deployed at federal (a wizard to support foreign companies in starting a new business as part of the implementation of the EU service directive, http://www.eap.gv.at/) as well as at municipal level providing services in various domains (building permit and business registration, http://egov.graz.gv.at/). Although there does not exist any formal analysis of the results of these two installations yet, their implementation has demonstrated the advantages of our approach. The first application was about supporting the building permit scenario. The system was actually evolving around this problem domain but every single requirement was generalized to the greatest possible extend. This generalization made it possible to create the service directive wizard within three weeks, although the system was never specifically designed to support such a scenario. To point out the effectiveness of our approach it is probably worth to mention that the federal chancellory had initially planed for a six month software development project. These results should indicate the potential of applying semantic MDA to the E-Government domain.

9. References

[15] Dumitru Roman, Uwe Keller, Holger Lausen, Jos de Bruijn, Rubén Lara, Michael Stollberg, Axel Polleres, Cristina Feier, Christoph Bussler, and Dieter Fensel, "Web


