Towards a Semantically-driven Software Engineering Environment for eGovernment

Dimitris Apostolou[1], Ljiljana Stojanovic[2], Tomas Pariente Lobo[3], Barbara Thoenssen[4]

[1] Planet S.A., Apollon Tower, 64 Louise Riencourt Str., 11523 Athens, Greece
dapost@planet.gr
[2] Forschungszentrum Informatik, Haid-und-Neu-Str. 10-14, 76131 Karlsruhe, Germany
Stojanovic@fzi.de
[3] INDRA, Avda. De Brusselas, 35, 28108 Alcobendas, Spain,
tpariente@indra.es
[4] University of Applied Sciences Solothurn, Riggenbachstrasse 16, CH-4600 Olten, Switzerland,
Barbara.Thoenssen@fhso.ch

Abstract. As software processes for developing eGovernment services become more complex, it is necessary to provide computer-based tools to support the software engineering process. Furthermore, actions should be taken to limit the loss of critical knowledge during the life cycle of eGovernment services. In this paper we first illustrate the overall architecture of ONTOGOV, an under-development software engineering environment for developing and managing the life-cycle of eGovernment services. We then outline two ontologies upon which ONTOGOV is based. Finally, an application scenario is described and the paper concludes with the identification of further steps and research directions.

1 Introduction

In developing eGovernment services, problems arise from the gap and inconsistencies that exist between the perspective of policy makers and managers of Public Administrations (PAs) on the one hand and the technical realization of eGovernment services on the other hand. Moreover, large amounts of information can be derived in an eGovernment software development project. Such information may vary form policy-enforcement information to information related to programming objects (e.g. modules, classes). As software processes for developing eGovernment services become more complex, it is necessary to provide computer-based tools to support the software engineering process that spans, horizontally, many PAs and, vertically, several levels of software engineering – from decision makers to programmers.

In order to support today's fast software development approaches (e.g. iterative prototyping, extreme programming), software models and code must be easily reconfigurable. Reconfigurability demands consistent representations of software

engineering information, homogeneous means of communication between software engineers (and other stakeholders) and tools, and support for managing changes in the software lifecycle. Another dimension of the problem addressed relates to eGovernment systems' architectures: Recently, novel component-oriented runtime environments have paved the way for service oriented infrastructures [1]. In the eGovernment domain, since there may be a considerable number of service providers which offer very similar functionality, it is difficult to choose the most appropriate service by interpreting syntactic operation names as provided by state of the art Web service interface descriptions [2].

To deal on the one hand with reconfigurability and changes of eGov services and on the other hand with integration between services provided by different providers, we need a software engineering environment based on robust conceptual models. We have used Semantic Web technologies for constructing ontologies, which represent the meaning of processed data and resources and provided functionality of eGovernment services. In this paper, we first illustrate the overall architecture of ONTOGOV, an under-development eGovernment software engineering environment. We then outline the ontologies upon which ONTOGOV will be based. An application scenario is described then, and the paper concludes with the identification of further steps and research directions.

2 Pertinent Technologies and Related Work

2.1 Semantic Technologies in eGovernment

The eGovernment scenario is in some respects a more obvious and promising application field for ontologies than many other e-business areas, since legislative knowledge is by nature already "formal" to a big extent and it is by definition shared by many stakeholders. The e-POWER project [3] has employed knowledge modelling techniques for inferences for, e.g., consistency checks, harmonisation or consistency enforcement in legislation. The SmartGov project [4] developed a knowledge-based platform for assisting public sector employees to generate online transaction services by simplifying their integration with already installed IT systems. Similarly, the ICTE-PAN project [5] developed a methodology for modeling PA operations, and tools to transform these models into design specifications for eGovernment portals. Further there are a number of ongoing projects e.g. Terregov [6], Qualeg [7] that make use of semantic technologies for achieving interoperability and integration between eGovernment systems. Although such projects have convincingly demonstrated the feasibility of semantic technologies in eGovernment, they did not adequately address the matter of eGovernment service software engineering, and in particular the lifecycle aspects of eGovernment services.

2.2 Web Services in eGovernment

In developing the ONTOGOV, we assume and utilise Web Services as the executable application interfaces logically accessible using standard Internet protocols (WSDL and SOAP). Current languages for describing web services (WSDL) and their composition on the level of business processes (BPEL4WS[1]) lack semantic expressivity that is crucial for capturing service capabilities at abstract levels. OWL-S[2] and WSMO[3]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 but they lack sufficient support for the process model itself. 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 and extend the OWL-S and WSMO ontologies so that they are able to better support process and life-cycle modeling.

2.3 Semantic-driven Software Engineering Environments

Software Engineering Environments (SEEs) are defined as integrated collections of tools that facilitate software engineering activities across the software lifecycle [8]. Deng et al. [9] have surveyed a number of knowledge-based software engineering systems: (i) most existing systems focus on a specific aspect of software development and do not support the whole lifecycle. In fact only two of the systems surveyed support the maintenance phase; and (ii) most existing systems aim to replace existing CASE tools and they do not support assertion of knowledge on top of existing CASE tools.

Ontologies are a promising means to achieve these conceptual models, since they can serve as a basis for comprehensive information representation and communication. Further ontologies can be used to address software engineering sub-domains, such as software versioning, change management, software quality, etc. Finally they can allow for involvement of non-technical people (e.g. public authorities' officers) in the software engineering process as ontologies can be used as coarse- or fine-grained models, therefore hiding or exposing details respectively and according to the intended audience.

[1] http://www-106.ibm.com/developerworks/library/ws-bpel/
[2] http://www.daml.org/services/owl-s/1.0/
[3] http://www.wsmo.org/

3 ONTOGOV Architecture

eGovernment services have strict procedures that do not allow choosing but a concrete service among several of them that may offer similar functionality. In the eGovernment domain, it is difficult to select the most appropriate service by querying dynamically, at run-time (*late-binding*), a service description interface, as for instance a UDDI registry. This is particularly true when the selection of the service should be context-aware. This is the case for instance of some geographically-distributed services where multiple eGovernment providers (as local authorities) may offer the same type of service, but the law states how to choose the correct authority that must provide the service. The complexity that the late-biding approach puts on the service description interface pointed to an architecture that integrates a top level design of the process model of the services with the orchestration of the underlying atomic services that perform the whole process. In this architecture, the sequence of atomic services execution as well as the conditional paths of execution are being set in advance, during the configuration phase (*early-binding*). The proposed approach is a deviation from a pure Service Oriented Architecture, where the concept of process modelling does not exist, but a chain of autonomous atomic services that inter-relate ad-hoc, without supervision or guidelines. Advantages of the early-binding approach include: (i) Better control of the atomic service selection process and better runtime performance, as atomic services are set in advance to a Web Service implementation and thus the time for discovering the most appropriate service is considerably reduced. (ii) Less deadlocks during the service execution, as pre-setting atomic services to concrete implementations decreases the possibility of faults during the execution.

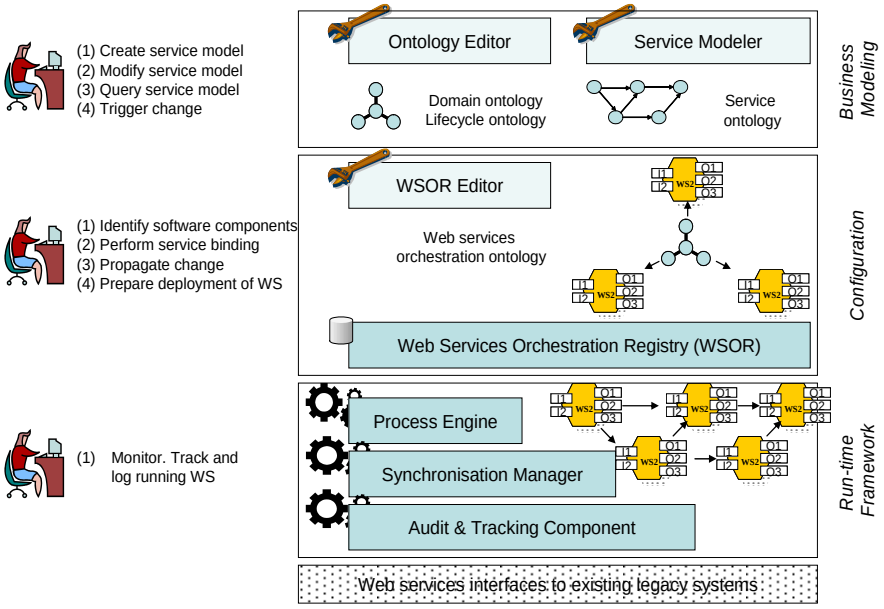

Fig. 1. ONTOGOV logical architecture

The proposed architecture (shown in Fig. 1) can be divided in three layers:

1. The Business Modelling layer is where a top level service model is drawn. Users of this layer will typically be PA domain experts that have sufficient knowledge of the domain. This knowledge includes the legislation that a service is based on, related directives, prerequisites etc.
2. The Configuration layer allows referencing the implementation of the business logic in actual software components. This task is carried out by the IT Consultant, who is responsible for the configuration and deployment of OntoGov services. In our platform, the software implementation will always be achieved through Web Services interfaces.
3. The Runtime layer should orchestrate and control the execution of the atomic services by making the correct invocations of the Web Services configured in the Configuration layer.

In principal, the lifecycle of an eGovernment service starts when PA Managers trigger the generation or the change of a service. In order to accomplish this task, PA Managers need to have a high-level view of service models, links to related laws, resources involved and inter-relations with other services. Such a high-level view is provided by the service models developed through the Business Model layer. The service ontology (or service model) becomes the main source of information for the Configuration layer. During configuration, the IT Consultant should identify the actual software components (Web Services) that enact the service model and the policy and security level that their SOAP messages should accomplish. The WS Orchestration Registry (described in detail in the next section) is an ontology-based repository that stores the mappings between atomic services defined in the service model and Web services that carry on with the task. According to the WSDL definition, these mappings comprise the selection of the WSDL operation (method) that should be called once the web service is invoked, and the linking of the WSDL parts (I/O attributes) to the atomic service inputs and outputs. A Runtime Framework should be properly installed in a broker machine to allow the execution of Web services. A key component here is the Process Engine that acts as an orchestration machine extracting the service ontology from the ontologies and proceeding to deliver the request to the first atomic service described in the process model. The engine relies on the use of a component called Synchronization Manager that hides the complexity of the synchronous or asynchronous behaviour of the Web services.

4 ONTOGOV Ontologies

In [10, 11], we introduced the following ontologies for modeling EGovernment services (Fig. 2): (i) Meta Ontology contains entities needed to describe services; (ii) Legal Ontology describes the structure of the legal documents; (iii) Domain Ontology contains domain specific knowledge; and (iv) Service Ontology describes a concrete service.

In this paper we extend our previous work aiming to resolve the two previously mentioned problems (reconfigurability and service integration) and to support the logical architecture outlined in section 3: (i) Lifecycle ontology that describes the

information flow and the decision making process in the public administration; and (ii) Web Service Orchestration Ontology that allows binding of services during execution.

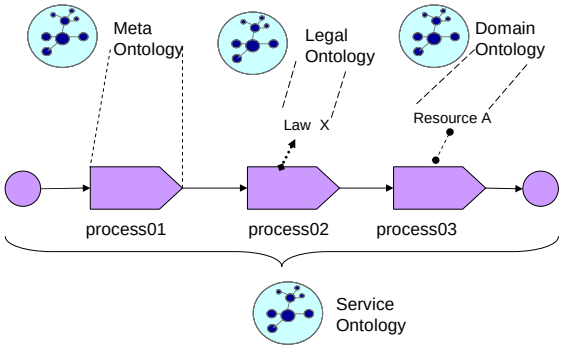

Fig. 2. Different ontologies used for describing semantic web services

4.1 Lifecycle Ontology

If an application aims at being useful, it is essential that it is able to accommodate the changes that will inevitably occur due to changes in the environment, users' needs or changes in its internal structures and processes. To avoid drawbacks of ad hoc management of changes, changes have to be applied on the model of the application. In order to do so, we developed the so-called *Lifecycle Ontology*.

The *Lifecycle Ontology* spans the range from the informal specification of requirements to a representation focusing on the realization of the service [12]. 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 has to be modified. 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 the *Lifecycle Ontology*, 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 two inverse properties "*hasReason*" and "*isReasonFor*" that are defined for the top concept of the concept "*Design Decision*" i.e. for the concept "*Reason*". The hierarchy of the concept "*Design Decision*" is shown in Fig. 3.

If the designer takes a design decision, s/he 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 of 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.

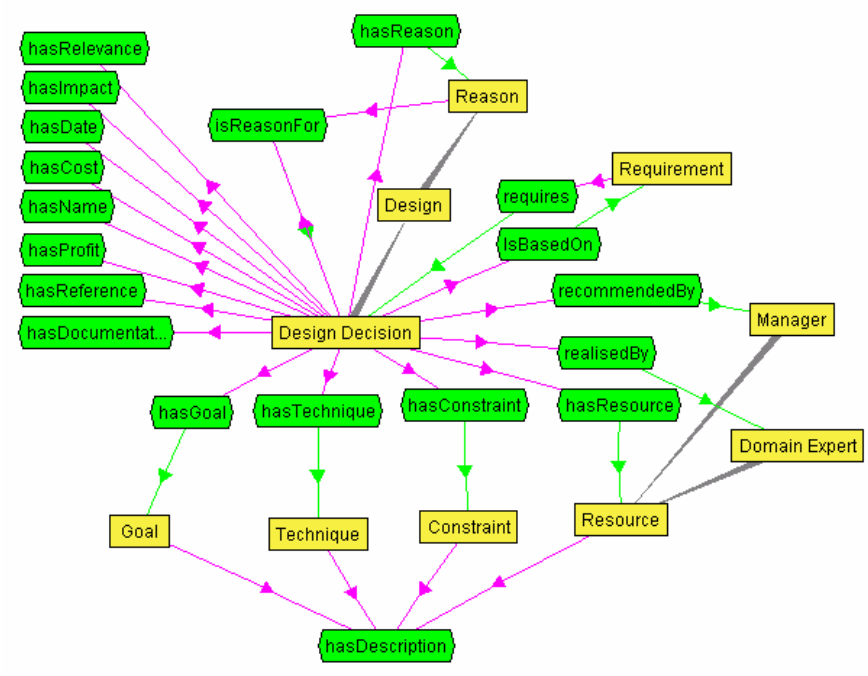

Fig. 3. A part of the Lifecycle Ontology

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.

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 (modeled through the "*hasReference*" property) to the *Service Ontology* or its activity that is related to this design decision.

Finally, information supporting decision-making, such as cost, relevance, priority, impact, profit, textual description of the reason for a service etc. may also be included (not show in detail herein). A part of the *Lifecycle Ontology* is shown in Fig. 3. 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.

4.2 WS Orchestration Ontology

In order to resolve the integration problem between software components and a service ontology, we have defined the *Web Service Orchestration Registry (WSOR) ontology*. It describes all information needed to finalise the configuration of the Web Services, which will be called during the service execution. This configuration consists in linking each atomic service of a service ontology to a WSDL description of real (existing) web services (i.e. software component). Moreover, it allows the dynamic binding of services during the execution.

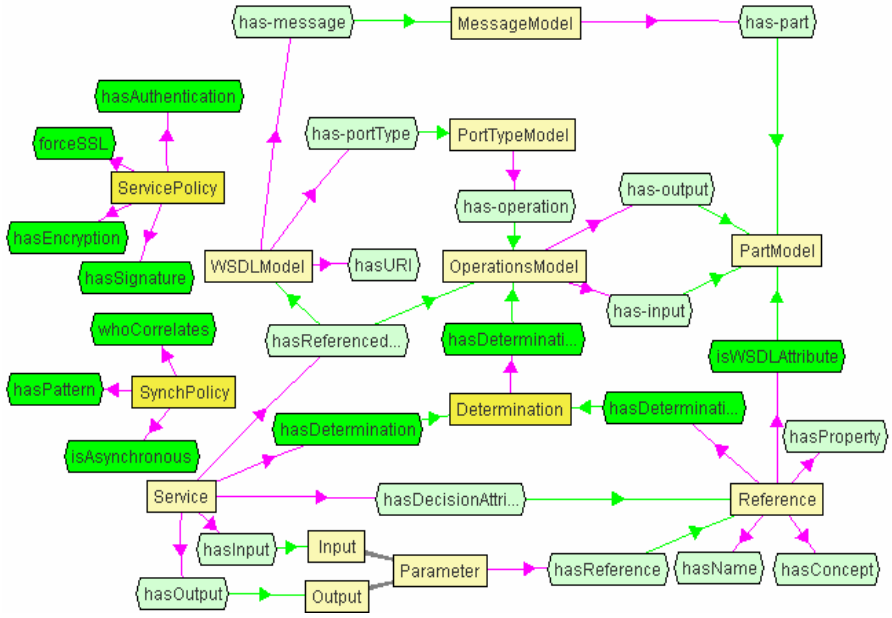

Fig. 4. The WSOR ontology

The *WSOR* is illustrated in Fig. 4. It includes the *WS ontology* that is shown in the upper part, the *Meta Ontology* shown in the bottom part. The middle part of this ontology shows entities that are defined in this ontology:

The concept "*Decision*" represents the value that a Reference has to hold during the runtime in order to lead to a concrete software component (WSDLModel).

"*hasReferencedSoftware*", links the Service with its WS implementations (operations in a WS).

"*isWSDLAttribute*" links the Reference concept with the PartModel.

"hasDecisionReference" liks the WSDLModel with the Decision.

"hasDecision" links the Service with the Decision, since one instance of the Service may have more than one Decision instance.

"hasDecisionValue" links the Reference with the Decision.

The WSOR that describes a concrete service establishes the mappings between the instantiation of the WSDL ontology and the service ontology. Indeed, it comprises the selection of the WSDL operation (method) that should be called once the web service is invoked, and the linking of the WSDL parts (I/O attributes) to the atomic service inputs and outputs. This mapping cannot be completely automated, but at least some recommendations can be generated. We propose three levels of mappings:

syntactical mappings –based on the string comparison. The names of the entities from the WSDL ontologies are compared to the lexical information about entities from the service ontology as well as to the synset [13] extension of them;

structural mappings – take into account the all inputs and outputs information at the same time;

context mappings – consider a set of activities in order to clarify the context in which an operation is used.

5 Application Scenario

In this section we illustrate how change detection and reconfiguration are addressed by ONTOGOV on the basis of an example using the service "Announcement of move". Today, the service provided is split into few separated tasks. De-registration has to be performed in one municipality, while several other entities, like telecommunication companies, have to be notified about the change of address. In addition, the person has to register in the new municipality. In order to improve service quality, there should be *one* task performed by the citizen regardless what and how much (technical) processes run behind. However, as a citizen may move from one municipality to any other – or even abroad – the change of address, deregistration and registration as well as the link between these processes can not be hard-coded because participating entities are changing every time the service is being performed.

The development of this (simplified) service with ONTOGOV will be as follows: The Domain Expert designs the service using the Service Modeler, based on the service ontology. Moreover, the domain expert adds more semantics by creating instances of the related ontologies:

Domain ontology, comprising concepts like data (e.g. name, first_name, municipality_from, municipality_to) and documents (e.g. application form, administration leaflet etc.)

Legal ontology, comprising instances of process relevant law or regulations, e.g. basis of the new process is a regulation about settlement. Then several instances will be initiated in the legal ontology indicating the related law[4] (1_Landesrecht), the paragraph ('14_Bürgerrecht) and article ('142_Niederlassungsrecht').

[4] Note: example is taken from the Swiss legislation

Organisational ontology, comprising instances of process relevant organizational units, e.g. involved in the new service are the organizational units 'Registration Office' and 'Administration Office' with its roles and personal.

Lifecycle ontology, comprising instances of all (design) decisions relevant for the new service (e.g. technical or process immanent reasons), including instances of the legal and organizational ontologies.

Working only with instances of (meta-)ontologies allows for strong governance of the modelling as a whole. For example, adding the same organisational unit to two atomic services in a sequence will evoke a warning (as usually the activities will be performed as one) even though the process flow per se is correct. Up to now no framework (like BPEL, ivyGrid or others) allow for such semantically checks.

After the design process is completed, a machine readable version of the definition of the new service will be generated. The IT-Consultant uses the WS Orchestration Registry in order to finalise the configuration of the web services. S/he links each atomic service of the service ontology to the WSDL description of real (existing) web services (e.g. of the municipality of Olten), performs the mappings between the WSDL and the attributes used in the service ontology and stores the links and the mappings in the WSOR.

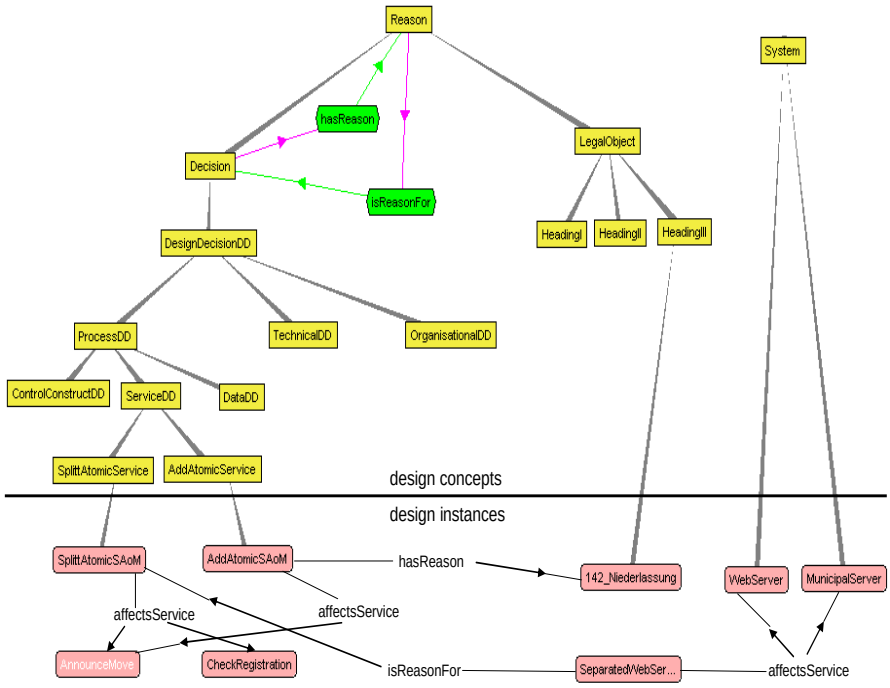

Fig. 5. Links between design decisions

In case a service needs to be modified at a later stage e.g. due to changes of a regulation, it is important to detect all affected services, respectively activities. That is why for every service component (activity or control construct), design decisions data

are modelled using the lifecycle ontology. In our example, “announce move” (checks, if all information required is filled in) and “check registration” (checks if the applicant is registered) are based on various decisions (Fig. 5). One decision is legally grounded: what data PAs need to know are defined by law (e.g. the applicant must give his/her name, current address, civil status etc.); another decision is technically grounded: due to security issues the activity “announce move” will be performed by a web server whereas the “check registration” activity will be run on a legacy system. A further decision is based on organizational reasons (e.g. activity 1 is performed by an organizational unit A (e.g. registration office) whereas activity 2 is performed by an organization unit B (e.g. administration office). Yet another decision is taken because of service-immanent reasons (e.g. activities are spited in order to make them reusable in other processes).

In case of a change of a law the ONTOGOV system can be queried to retrieve affected activities. Assume that the change affects all processes related to article ‘142_Niederlassungsrecht’. ONTOGOV searches for all decisions based on this legal reason. As a result, all affected services and activities are listed and proposed for modification. In the example, this is the service “Announcement of Move” with its activity “AnnounceMove”.

6 Conclusions and Outlook

In this article, we highlighted a novel application of semantic technologies in the eGovernment domain: utilising semantics to drive and support the software engineering process for the development of eGovernment services. We considered the eGovernment domain, since eGovernment services are under the continual adaptation to the political goals of a government and to the needs of the people. Our approach (i) covers all the phases from definition and design through to deployment and reconfiguration of eGovernment services); (ii) provides the basis for designing lower-level domain ontologies specific to the service offerings of participating public authorities; and (iii) provides the basis for limiting the loss of critical knowledge during the life cycle of the software engineering process, in which a number of stakeholders (from policy-makers to developers) is involved.

In the near future, we will work to develop the actual software engineering environment and test it in three real-life governmental pilots. Future research directions include addressing adaptivity: extending this approach so that its implementation is capable of suggesting changes that can improve services. This can be done (i) by monitoring the execution of EGovernment services (e.g. the activity that causes the delay is a candidate for optimisation) 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).

Acknowledgement

We would like to thank the European Commission for funding the OntoGov project through the IST programme.

References

1. M. Endrei, J. Ang, A. Arsanjani, Patterns: Service-oriented Architecture and Web Services, IBM Redbook, April 2004.
2. A. Paar, W. F. Tichy, “Semantic Software Engineering”, in proceedings of AMS 2003.
3. T. van Engers, J. M. Patries, J. Kordelaar, J. den Hartog, E. Glasséee (2002). Available at http://lri.jur.uva.nl/~epower/
4. N. Adams, S. Haston, A. Macintosh, J. Fraser, A. McKay-Hubbard, and A. Unsworth, SmartGov: A Knowledge-Based Design Approach to Online Social Service Creation, in Bramer, M., Ellis, R., Macintosh, A; (eds.). 'Applications and Innovations in Knowledge-Based Systems and Applied Artificial Intelligence XI'; Proceedings of AI-2003 the 23rd Annual International Conference of the British Computer Society's Specialist Group on Artificial Intelligence ; Peterhouse College, Cambridge, UK, 16th-17th December, 2003
5. E Loukis, S. Kokolakis, Computer supported collaboration in the Public Sector: the ICTE-PAN Project, in preceedings of eGOV 2003 / DEXA 2003.
6. N. Benamou, “Terregov project overview”, IANOS conference, Budapest, 2004.
7. C. Tatsiopoulos “QUALEG approach to intra-government interoperability”, in Workshop on Technological and architectural challenges, eGOV04 Conference, Zaragoza Spain, 1st September 2004.
8. R. S. Pressman, “Software Engineering: A Practitioner’s Approach”, 5th Edition, New York, McGraw-Hill, 2000.
9. D. Deng, P. C.-Y. Sheu, T. Wang, H. Maezawa, F. Tsunoda and Akira K. Onoma, “DPSSEE: A Distributed Proactive Semantic Software Engineering Environment”, The Fifth IEEE International Symposium on Multimedia Software Engineering (MSE 2003), Taichung, Taiwan, ROC, December 10-12, 2003.
10. L. Stojanovic, A. Abecker, N. Stojanovic, R. Studer, An Approach for the Change Management in the EGovernment Domain, In Proceedings of Second International Conference on Knowledge Economy and Development of Science and Technology (KEST’04), Beijing, China, 2004.
11. L. Stojanovic, A. Abecker, N. Stojanovic, R. Studer, On Managing Changes in the ontology-based EGovernment, to appear in Proceedings of the 3rd International Conference on Ontologies, Databases and Application of Semantics (ODBASE 2004), Larnaca, Cyprus, October 2004.
12. D. Landes, Design KARL – A language for the design of knowledge-based systems, In Proceedings of the 6th International conference on Software Engineering and Knowledge Engineering (SEKE’94), Jurmala, Lettland, pp. 78-85, 1994.
13. C. Fellbaum, WordNet - An electronic lexical database, MIT Press, 1998.