A collaborative decision framework for managing changes in e-Government services

Dimitris Apostolou a, Gregoris Mentzas b,*, Ljiljana Stojanovic c, Barbara Thoenssen d, Tomás Pariente Lobo e

a University of Piraeus, Karooli & Dimitriou 80, 18534 Piraeus, Greece
b National Technical University of Athens, 10682, Athens, Greece
c Forschungszentrum Informatik, Haid-und-Neu-Str. 10–14, 76131 Karlsruhe, Germany
d University of Applied Sciences Northwestern Switzerland, Riggenbachstrasse 16, CH-4600 Olten, Switzerland
e Semantics, Software & Service Engineering ATOS Research & Innovation, Spain

ARTICLE INFO
Available online 12 October 2010

Keywords:
e-Government services
Change management
Ontologies
Decision support

ABSTRACT
Developing and maintaining e-Government services that can effectively deal with changes is a challenge for public administrations. In this paper, we address this challenge by presenting an ontology-based approach that: (i) enables systematic response of e-Government systems to changes by applying formal methods for achieving consistency when a change is discovered; (ii) enables knowledgeable response of service designers and implementers to changes by utilizing design rationale knowledge. We argue that such a synthesis of systematic response to changes with knowledge to deal with them has a positive impact on the change management process. Evaluation of the proposed approach in three case studies let us develop useful propositions for practitioners, discuss policy implications and identify future research topics.

© 2010 Elsevier Inc. All rights reserved.

1. Introduction

In a continuously changing political and societal environment, e-Government services need to be continually improved in order to reflect political and societal changes. Changes that affect e-Government services may be caused by changing citizens’ needs, changing legal regulations, availability of new technologies, outsourcing opportunities, and new service provision models. Although changes encompass several dimensions of Government service provision (e.g., people, processes, and technologies), most of them are reflected on its software infrastructure. For example, the establishment of a new department in the organization will require changes in business processes which will in turn have an impact on the delivery of e-Government services. Building and maintaining long-living, e-Government services that are “open for changes” is a challenge.

Change management in general refers to the task of addressing changes in a timely, planned and systematic manner. Change management has been widely acknowledged as a critical success factor in software systems (Rajlich, 2006). Business process change management theory (Scheer, Abolhassan, Jost, & Kirchner, 2003) poses the following conditions for successful resolution of changes: (a) necessary actions are initiated after the change has happened; (b) necessary actions are executed in a fast and effective way; and (c) all reactions and actions are initiated and executed in a controlled manner.

E-Government services pose unique challenges to change management because they require the co-evolution of the front-office service along with the related back-office IT infrastructure. E-Government services are frequently distributed over different IT systems and organizations. Even if they are provided and managed by a single organization, their design, development, and operation relies on the collaboration of many people with different roles (Anthopoulos, Siouzos, & Tsoukalas, 2007). Moreover, business processes of different public administrations that are often physically distributed and with different levels of formality and structure, need to be seamlessly integrated in order to maximize usefulness for the citizens in the form of “one-stop services”.

Changes affecting e-Government services may originate within public administrations. Frequently changes are caused by organizational re-structuring or due to the possibility to organize services in a better way. Moreover, changes may be triggered by events originating outside the public administration such as changing laws and regulations. Hence, change management must take into account the response to changes, such as changing legislation, over which public administrations exercise little or no control.

Taking into account a wealth of e-Government services and an even larger number of dependant back-office business processes and relationships between them as well as the complexity of interpreting and implementing changes in government regulations, it is highly complex to reconfigure e-Government services. It is necessary to provide support for propagating changes to all dependent service artifacts, for
deciding collaboratively how to deal with the identified changes and for ensuring the consistency of the service and the software infrastructure that enacts it throughout its lifecycle. Although change management in software engineering, in particular software evolution (Lehman & Ramil, 2003) and requirements traceability (Etien & Salinesi, 2005), as well as change management of business processes (Earl, Sampler, & Short, 1995; Kettinger & Grover, 1995) have been extensively studied, corresponding methods and tools that support change management of e-Government services are still missing at large. Software Configuration Management tools only partially address the problem; although they are commonly used to identify necessary changes, analyze the impact of changes, and track changes in system versions (Estublier et al., 2005; Hass, 2002), consistencies of other artifacts such as design models as well as dependencies among artifacts, particularly among legal regulations and services, are not adequately managed. In fact, it is the presence of these dependencies that makes change management of e-Government services very complex. For example, a change in a federal law may lead to multiple modifications in design models, source code, and test cases. Without the capability to systematically deal with changers in a variety of artifacts and acquire and use knowledge about how these artifacts are related, it is very difficult to incorporate modifications in the system. This motivates our research question: “How can the change management process of e-Government services be enhanced with means to consistently track changes and with knowledge to deal with these changes?” In this paper, we address this research question by presenting an ontology-based decision framework that: (a) enables systematic response of e-Government services to changes by applying formal methods for achieving consistency when a change is discovered, and (b) enables knowledgeable service designers and implementers’ response to changes by utilizing design rationale knowledge. We argue that the capability to respond systematically and knowledgeably to changes has a positive impact on the change management process.

2. Underlying theories for ontology-based change management

Our approach combines the means for documenting knowledge about cause–decision relationships rooted in the area of design rationale in engineering with formal verification methods applied in the area of ontology evolution. An overview of underlying theories is presented in this section.

The development of large-scale, complex software has been widely recognized as a knowledge intensive activity. Knowledge artifacts that are needed for shaping crucial design decisions exist as chunks scattered in various development environments (Mohan, Xu, Cao, & Ramesh, 2008). Integration of such distributed knowledge artifacts is considered as a key to successful software development (Walz, Elam, & Curtis, 1993). In distributed software development — a commonplace in the development of e-Government systems where parts of the system can be outsourced — knowledge integration becomes especially challenging when the team members are involved in the co-construction of “collective work products” (Katzenbach & Smith, 1993). Our research is based on the premise that while Web service technologies are being widely utilized in e-Government systems providing benefits in terms of easy integration and interoperability, augmenting them with ontology-based facilities for knowledge integration will help achieve more effective control of changes and service evolution by capturing the semantics of changes and the dependencies among related artifacts across and within different phases of the development lifecycle (e.g., the dependency among requirements, design elements, source code, etc.).

In software engineering, knowledge about both the artifacts of the software system and the process of their development and evolution must be managed (De Lucia, Fasano, Oliveto, & Tortora, 2007). These two types of knowledge are characterized as product and process knowledge, respectively (Mohan et al., 2008; Far, Ohmori, Baba, Yamasaki, & Koono, 1996). Product knowledge refers to knowledge about artifacts of the software system (models, specifications, documents, versions of these artifacts, etc.); process knowledge refers to knowledge about the process followed in the development of these artifacts (for example, how a requirement is implemented in the design model or code and why a design decision was made). Components of process knowledge include design decisions and dependencies among artifacts (Mohan et al., 2008). Design decisions include reasons that explain why system components are designed in a specific way. Dependencies among artifacts refer to how changes in some artifacts may impact other artifacts.

One of the most valuable types of process knowledge created during software projects, but usually not explicitly captured, is rationale. Rationale is the justification behind decisions. Rationale becomes a critical knowledge, particularly if changes in the problem or the solution domain occur (e.g., requirements change or a new technology arises); this makes it difficult for someone that was not involved in the decisions to understand why the component is designed the way it is. Recent studies demonstrated the importance of rationale for the different software-lifecycle activities (Dutoit, McCall, Mistrik, & Paech, 2006; Dutoit & Paech, 2003). There exist many proposals for approaches, systems, and representation of design rationale knowledge (Regli, Hu, Atwood, and Sun (2000) provide an excellent survey of design rationale systems and concepts while Bruegge and Dutoit (2004) give a detailed introduction to rationale and its management.

Unfortunately, rationale is one of the most complex types of information developers generate. For that reason, externalizing and representing rationale knowledge as issues, alternatives, and arguments for a decision (Conklin & Begeman, 1988) is exhausting. In most of the cases, rationale knowledge is communicated informally and remembered by individuals. Over time, this information degrades and gets lost when staff members leave the organization. Managers, as well as developers, always scrutinize the profitability of such a major investment of time and resource. Since our work aims at a practical solution for the e-Government domain, we opted for a light-weight rationale representation approach as it will be presented in the next section.

Drawing from the literature on knowledge-based software engineering and design rationale in engineering, we argue that the integration of product and process knowledge, leveraged with ontology-based tools, will improve stakeholders’ understanding of the e-Government system evolution process and thereby improve the change management process. Next, we discuss the role of ontologies in the e-Government domain and we focus on the theory of ontology evolution which will be exploited in our effort to meet the conditions for successful resolution of changes affecting the e-Government system.

An ontology — a formal and explicit specification of a shared conceptualization of a domain of interest (Gruber, 1993) — 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 conceptualization of a domain explicit; they serve as a means for establishing a conceptually concise basis for communicating knowledge for many purposes. E-Government systems represent promising applications of ontologies because they heavily depend on legislative knowledge, which is by nature formal, explicit, and shared by many stakeholders. Ontologies for modeling legislative documents have been developed in the E-Power project (Winkels et al., 2005; Boer, van Engers, & Winkels, 2003; van Engers & Glassée, 2001). SmartGov (Vassilakis et al., 2007) project used ontologies for representing the profile of an e-Government service. The ICTE-PAN (Loukis & Kokolakis, 2003) project focused on ontologies supporting public sector collaborative public policy and program development, monitoring and management, and decision making. In the e-Power and in the e-Gov projects (Tambouris, 2001), e-Government metadata standards were defined. These standards can be extended into ontologies.
for describing the profile of a service. In the Terregoy (Moulin & Shodio, 2005) project, ontologies have been utilized for achieving semantic interoperability and integration between e-Government systems. Moreover, ontologies have been utilized for government information management, search in retrieval (Prokopiadou, Papatheodorou, & Moschopoulou, 2004). Such works have convincingly shown the feasibility of ontology approaches in e-Government and have pointed out challenges that need to be addressed such as ontology evolution.

Ontology evolution (Stojanovic, 2004) refers to the process of modifying an ontology while keeping its consistency. The modification is achieved by systematically applying elementary and composite ontology changes. A full set of elementary changes can be defined by the cross product of the set of entities of the ontology model (e.g., concept, relation, and instance), and the set of meta-changes (e.g., addition and removal). Ontology consistency is ensured by conforming to a set of consistency constraints. Once consistency constraints have been defined and formally represented, automatic consistency checks can be performed. In our work, we exploit the theory of ontology evolution in order to consistently perform changes in e-Government services, which are modeled using ontologies, as explained in the following section.

3. Ontology-based change management of e-Government services

In this section we present our decision framework for managing changes in e-Government services. We start by illustrating the practical aspects of the problem outlined in the introduction. We then describe how ontologies are used by the framework in order to:
(a) enable systematic response of e-Government services to changes by applying formal methods for achieving consistency when a change is discovered, and
(b) enable knowledgeable response to changes by utilizing design rationale knowledge.

3.1. Motivating example

Let us consider the “Announcement of Moving,” a service launched when a citizen changes his/her place of residence in Switzerland. The citizen is asked first to provide all information needed to perform the complete service (cf. “EnterApplicationForm” in Fig. 1). After submitting the requested information, the eligibility is checked (cf. “CheckEligibility”). Based on the result, the service can be either finished (cf. “RejectApplication”) or continued. The next steps include deregistering (cf. “Deregistration”) from one municipality and registering (cf. “Registration”) in another. In the meantime several other entities, like utility companies, have to be notified about the change of the address (cf. “GetThirdPartiesAddress” and “NotifyThirdParties”). Finally, the citizen has to be informed about the result of the service.

By integrating distributed services and by making them available as an end-to-end service, the aim is to provide users with a high-quality, “one-stop” service, one that is provided as a single, seamless task regardless of what and how many activities run in the background and in which organization. However, the task of administering a “one-stop” service is complex. Knowledge required to administer services should be captured and represented. Not only knowledge about service artifacts (e.g., models, specifications, documents and versions of these artifacts) should be represented but also why it (e.g., relative legislation) was designed as it is. Therefore, for every entity in the model of a service (i.e., either an activity or a control construct), information on the underlying design decisions may be required. An example of the design decision defined for the activity “CheckEligibility” is shown in Fig. 1. This decision is legally grounded: the information that service stakeholders need to know regarding this activity is defined by law (cf. SR 101 and SR 201 Art. 22A–26A). Additionally, a decision may stem from technical or organizational reasons.

In the instance that a reason is changed, the relevant information should be propagated to the affected constituent processes and activities. For example, in the event of a change in law SR210, first all service models such as “Announcement of Moving” and their activities affected by the change such as “Check Eligibility” must be identified. Then, the service administrator should be informed of all consequences of the change. The administrator should also be assisted in consistently applying all changes required by the law and other changes that are possibly derived from them, and in keeping track of performed changes. The software programmer responsible for the IT infrastructure enacting the service should be notified when a new service model is available and assisted in re-configuring the corresponding software. For this scenario to be realized, information on the underlying design decisions should be represented and stored, while means for assisting the identification and consistent propagation of changes should be provided.

3.2. Overview of e-Government service change management approach and model

To illustrate our approach for the adaptation of e-Government services to changes in legislation, business requirements, users’ needs, etc. as well as the consistent propagation of these changes to dependent artifacts, we focus on e-Government services implemented using Web service technologies. The underlying change management model of an e-Government service is comprised of three layers:

- Top layer, consisting of an upper-model defining the fundamental e-Government service change management elements.
- Middle layer, comprising a set of reference models specifying in more detail the e-Government service change management elements. These models are represented as ontologies, which can be customized within the scope defined in the meta-model. The reference ontologies are available at: http://sourceforge.net/projects/ontogov
- Bottom layer, containing possibly distributed, application-specific ontologies that instantiate the reference ontologies and Web services enhanced with metadata annotations that take their values from the application-specific ontologies. The composition of Web services provides the e-Government service.

The upper-model, which provides an anchor to the set of reference ontologies, is presented in Fig. 2. The upper-model is built around the service (process) entity. Similar to the Business Process Execution Language (http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf), the Service (Process) ontology models the process flow of a service using activities (which can be either atomic or composite) and control constructs (e.g., sequence, split, join, switch, etc.), and the data flow through inputs and outputs as well as the equivalence relationship between them. Similar to the Web Service Semantics approach (http://www.w3.org/Submission/WSDL-S/), the reference ontologies containing the semantics of change management elements are maintained outside of Service (Process) ontology and are referenced from it.

Inputs and outputs of activities are represented using entities defined in the Domain ontology, which models concepts of the domain of the organization such as certificates and documents. The Legal ontology models the structure of the legal documents, which includes paragraphs, sections, articles, amendments, etc. The Organizational ontology provides constructs to model an organization by defining its organizational units, roles, persons, resources etc. The Lifecycle ontology includes entities for documenting design decisions of service models. The Life Event ontology models the categorization of the e-Government service and its mapping to citizen situations where the service is applicable. The Profile ontology contains metadata about the e-Government service defined in all previously mentioned ontologies and is used for service discovery. Finally, the Service Evolution and the Web Service Orchestration Registry ontologies are used for tracking changes in the e-Government service, and for the deployment and execution of the service, respectively.
3.3 Focus on systematic response to changes

Based upon the set of ontologies outlined above, we define four phases for the change management process. The process starts with explicitly representing a request for change. Then, the consistency preservation phase prevents inconsistencies, the change generation phase computes derived changes that guarantee the transition of the model into another consistent state, and finally, in the change propagation phase, all dependent artifacts are found and updated.

3.3.1 Phase 1: change representation

Changes are explicitly represented in the Service Evolution ontology, shown in Fig. 3, which models what, why, when, how, and by whom the changes are performed in the service model and in related ontologies. The core concept in the ontology is the “Change” concept. Changes are broken down into (a) changes causing addition (concept “AddChange”), (b) changes provoking deletion (concept “RemoveChange”), and (c) changes for modification (concept “SetChange”). Each leaf concept in the hierarchy of the concept “Change” represents a specific change that can be applied to a service description, e.g. “AddAtomicService”, “AddSequence”, etc. Moreover, there are many specializations of the property “has_referencedEntity”, which indicates the type of entity that is considered. For example, the “has_referencedAtomicService” property is used to reference changes related to an atomic service in the service description.
Changes may necessitate additional new changes in order to keep the service model consistent. Moreover, there may be groups of changes triggered by a single request. Further, in certain cases one should be able to reverse the effects of a change, such as when a change has been performed for experimentation or simulation purposes. To address these issues, we introduce the properties “has_previousHistoryChange” and “causesChange”, which enable the reconstruction of sequences of performed changes, as well as the concept “LOG”, which provides an anchor to the list of performed changes and the property “lastChange”, which indicates the performed change. Since one change might provoke additional changes, the service evolution log does not have a linear structure. It is organized as a list of performed changes (Fig. 4) where the order of changes in a list is defined through the “has_previousHistoryChange” property. Each element of a list is represented as a tree of all its consequences (i.e. generated changes) where the property “causesChange” represents the cause–consequence relationship between performed changes.

There is one and only one instance of the concept “LOG” that indicates the end of the list, i.e. the last required change. The fact that one change is explicitly requested (and therefore is not a consequence of other changes) is represented explicitly through the “requestedChange” attribute. The “has_previousActualChange” property indicates the current state of the service description by excluding the effects of inverse changes (i.e. reversibility). In this way, it specifies only the necessary changes to achieve the resulting service description. On the contrary, the “has_previousHistoryChange” property takes into account the sequence of all changes that have actually taken place. It models the actual evolution process in a unique way because it records all intermediate versions as well. In the case that a log of changes does not contain changes that undo other changes (e.g. AddAtomicActivity(“X”) followed by RemoveAtomicActivity(“X”)), the “has_previousActualChange” and “has_previousHistoryChange” properties for each change instance point to the same target instance.

A partial example of the service evolution ontology is shown in Fig. 5. It contains instances of sub-concepts of the concept “Change” (cf. RemoveAtomicService). Each instance contains data about a particular change such as the version number (cf. 8). This required change caused three additional changes that are related through the property “causesChange”. The “has_previousActualChange” and “has_previousHistoryChange” point the same change, since there were no undo operations.

3.3.2. Phase 2: consistency preservation

Changes should be applied to a consistent service model and, after all the changes are performed, the model must pass into another consistent state. The proposed approach incorporates mechanisms for verifying the service model with respect to different consistency constrains as well as mechanisms enabling us to take actions to optimize the service model. We have introduced a set of predefined consistency constrains, presented in Table 1. Moreover, the service description may conform to additional constrains, specific to public administrators’ requirements. Therefore, we allow public administrators to extend this set of conditions with additional constraints that represent their needs.

We treat consistency preservation with a proof-theoretic formal verification method in which consistency constrains are formally represented and an inference process is followed to verify whether constraints are satisfied. Once we have a service model and formally defined consistency constrains, we automatically check whether these constraints are satisfied with the help of the KAON2 inference engine, which implements the proof-theory for DL and DL-safe rules. Consistency constrains are transformed into DL-safe rules (Oberle, Staab, Studer, & Raphael, 2005); an example is given in Fig. 6.
3.3.3. Phase 3: change generation

Change generation refers to the generation of additional changes that resolve detected inconsistencies in the service model. In our approach, a change definition includes rules that specify the side effects of a change on other related entities. To define the rules for each change, we started by finding out the cause–effect relationship between changes. This kind of dependency between the changes forms the so-called change dependency graph. A change dependency graph is a directed graph defined as:

$$CDG := (CH, E)$$

where:

- $CH = \{Ch_i\}, 1 \leq i \leq |CH|$, is a set of nodes and each node represents a change $Ch_i$;
- $E = \{E_k\}, 1 \leq k \leq |E|$, is a set of labeled edges and each edge represents the cause–effect dependency between changes (i.e., nodes). An edge is defined in the following way:

$$E_k = (Ch_i, Condition_j, Ch_l), Ch_i \in CH, 1 \leq i \leq |CH|, j \leq l \leq |CH|, i \neq l.$$
service model, a list of all implications to the model is generated and presented to the user who modifies the model. The user has possibilities to accept or even to abort the entire modification when she/he realizes that it would have undesired consequences for other parts of the service.

3.3.4. Phase 4: change propagation

Ontologies may be distributed in different public authorities; therefore, there is a need to synchronize them. In particular, there is a need to synchronize the Service ontology and associated ontologies such as the Legal ontology and other Service ontologies themselves in case of composite services. There are generally two approaches to perform synchronization of Web data (Bhide, Deogas, Katkar, Panchbudhe, & Ramamritham, 2002): (a) push-based approach, where changes from the changed ontology are propagated to dependent ontologies as they happen; and (b) pull-based approach, where changes from the changed ontology are propagated to dependent ontologies only at their explicit request. We follow the pull-based approach, in which information about associated ontologies is stored in the dependent ontology. Each ontology is identified by a Universal Resource Identifier (URI) and can be included (copied) in a service model. The original versions of included ontologies are checked periodically to detect changes. Each ontology has a version number associated with it that is incremented each time the ontology is changed. Checking the equality of the original of the included ontology and the replica can be done by a simple comparison of the version numbers. After determining that the included ontology needs to be updated, the evolution log for this ontology is accessed. The extracted deltas contain all changes that have been applied to the original after the last synchronization with the replica, as determined by the version numbers. Each performed change is analyzed, in order to find services or activities that have to be updated. Consider as an example the addition of a new amendment to a law. All services and activities that reference the specific law can be retrieved and presented to the user together with an analysis of all changes that have been applied in the law after the last synchronization. The user can then take further action, taking into account design rationale knowledge as explained next.

```
ErrorReachable(X) <-
\neg isReachable(X)

isReachable(Y) <-
isReachable(X) \land
hasNext(Y, X)

isReachable(Y) <-
hasFirstService(X, Y)
```

Fig. 6. Verification based consistency rules (left). The right side shows an inconsistent process model that violates consistency constrain C11 (reachability).

3.4. Focus on knowledge support to deal with changes

We have demonstrated how our approach supports the following: representing changes in service models, finding inconsistencies caused by a requested change in a service model, generating and propagating changes to affected artifacts, and alerting the user to them. Moreover, our approach is capable of capturing product knowledge, for example, knowledge about artifacts of the e-Government service (models, specifications, documents, etc.) and linking it to the Service (Process) ontology. Helping users notice inconsistencies and access product knowledge only partially addresses the issue, however. Ideally, there should also be support for public servants in resolving changes by providing useful process knowledge how to do that.

When developers need to process change requests, they typically need to understand the ramifications of the request in terms of the number of different artifacts that would be affected and details about past design decisions that may be relevant for the change request (Mohan et al., 2008). Design rationale support refers to documenting the decisions made during the definition of the service model and to formally stating the reasons for these decisions.

- **Design Decision**: A Design Decision is related to (parts of) the service model. It is represented by a name and a description and is used to answer the question: “How have the service design (i.e., activities) and flow (i.e., control constructs) been realized?”
- **Reason**: Any Design Decision is based on at least one Reason. The Reason itself contains a name, a description, and links to the respective Design Decisions and relevant elements in the Legal, Domain or Organizational ontologies. A Design Decision can again be a Reason for any other Design Decision; it answers the question: “Why has a Design Decision been made?”

If we generalize the above mentioned concept, we can consider Reasons that are shared among different Design Decisions. Moreover, Design Decisions can be used as Reasons themselves, whereby chains of Design Decisions and Reasons can be built. The process knowledge is stored in the Lifecycle ontology (Fig. 7), which is realized as a shared resource between all provided services. The advantage of this approach is the ability to share Reasons between different Service ontologies but also to allow referring Design Decisions from other models as Reasons. The reuse of Reasons is illustrated in the following two examples:

- The law about data protection is relevant to many services. Instead of creating a new instance of Reason every time this law is referenced, the Reason is created and reused for every Design Decision. Moreover, only one Reason has to be synchronized, when changes in the law occur.
- Referring Design Decisions as Reason is used when creating ‘Design-Decision-Chains’. For instance the decision, to split an action “checkInputInformation” into two activities is caused by the fact
that two different authorities are responsible for this action. For example the residents' registration verifies personal information and the “immigration office” verifies the validity of the visa, in case the citizen is a foreigner. As a consequence, the design of the activities to get the information from these different departments is based on the previous Design Decision that is now used as a Reason.

Linking domain, legal, organizational, service, and rationale spaces can be valuable in understanding the reasons behind the ripple effects of changes and resolving emerging problems, as it will be illustrated in the application scenario of Section 5.

4. Implementation

Different types of information systems are being utilized to provide e-Government services, ranging from database and workflow systems to dedicated e-Government systems. In this work we make the assumption that services are provided in a service oriented architecture (Bichler & Lin, 2006; Endrei et al., 2004) in which different activities that constitute business processes are enacted as Web services. On top of such an environment, we have developed OntoGov, an open source software framework for modeling, configuring, and executing e-Government services, which implements the proposed change management approach (http://sourceforge.net/projects/ontogov/).

OntoGov is a tree-tier platform (Fig. 8) that comprises three major systems: (1) The Ontology Management System (OMS) is a system for creating, modifying, versioning, querying, and storing service (process) models; (2) The Service Configuration System — Configuration Framework assists the configuration of Web service-based applications that enact the service modeled with the OMS; and (3) The Service Configuration System — Runtime Framework allows the execution of the Web service-based applications. Finally, there is an auxiliary component called User Management Component that performs user authentication and authorization.

The Ontology Management System (OMS) supports service model lifecycle management. It contains two user interfaces: (a) the Service Modeler used for editing Service and other ontologies, and (b) the Service Registry used for accessing distributed ontologies. The middleware layer of OMS implements the change management processes described in Section 3, implemented in the modules for consistency preservation, change implementation and change propagation. The lifecycle and registry modules support versioning aspects of ontologies and their consistent, distributed management. Data storage is based on KAON2 (http://kaon2.semanticweb.org) which provides an API for storing, querying and inferencing ontologies.

The Configuration Framework of the OMS creates with the OMS and produces a package, which includes among others the Web Services Orchestration Registry (WSOR) Ontology and service configuration data. The Configuration Framework consists of three basic components: (1) The SCS Builder, which interacts with the OMS and initiates the process of the service configuration. (2) The WSOR Manager, which binds the service model developed in OMS to Web service parameters that automate parts (e.g. activities) of the service. (3) The Deployment Component, which retrieves the results of the previous components, makes some necessary configurations and provides a package for the executable service in the form of a Web service.

The SCS Runtime Framework is a J2EE platform based on Open Source Software (JBoss, OpenJMS, MySQL) for executing Web services configured in the SCS Configuration Framework. It acts both as a Web service provider and as a Web service orchestrator of the underlying Web services which are provided by collaborating organizations and which participate in the e-Government service. The SCS-RF consists of four basic components:

1. The Process Engine component exposes a workflow-like interface to enact the overall Web service. It queries the WSOR Ontology to locate subsequent Web services to be executed until the service ends.
2. The Web Services Manager component handles generation and processing of the Simple Object Access Protocol (SOAP) messages. It retrieves from the WSOR ontology the mappings of activities to concrete Web services and selects which ones should be invoked at runtime. This selection is done by evaluating the driven expression associated to the activity using a rule evaluation system (not described in this paper).
3. The Synchronization Manager component provides message persistence for reliable invocation of asynchronous Web services.
4. The Audit & Tracking component allows authorized users to query the SCS-RF runtime log and retrieve runtime-related information, such as service status and date of last modification. It also allows the canceling of services in execution.

5. Application scenario

To explain how changes affecting e-Government services are managed with OntoGov, we refer to a typical scenario in which a state law changes. At the outset of the scenario, a service model for
“Announcement of Move” has been created using OMS together with the associated ontologies (e.g., Legal, Organizational, etc.) and the corresponding links. Verification for checking the model consistency has been performed (not shown). Design Decisions and Reasons are added to the model as means to make explicit the domain expert’s knowledge about the service design. Some Design Decisions have legal reasons, which are captured in text and are linked to the knowledge about the service design. Some Design Decisions have been updated (not shown) to reflect the recent change in the state law. Because of this change, the e-Government service, “Announcement of Move”, may have been affected. The domain expert has to check whether the service has been indeed affected and to make the corresponding changes to it. The user starts by opening the current “Announcement of Move” service model in OMS and synchronizes it with the Legal ontology (Fig. 10, section A). After selecting “Synchronize with legal ontology” a window pops up indicating which activities are affected by the detected changes (Fig. 10, section B). Some of the detected changes affect Design Decisions related to the specific service model. The View/Modify Design Decisions window flags all affected Design Decisions with “to be checked” (Fig. 10, section C). In this example, the flag has been raised because two instances of the associated Legal ontology have been deleted (Fig. 10, section D). The domain expert should examine why this change in the Legal ontology has been performed and rectify the dependency accordingly, by removing the Reason completely or modifying it and linking it to other instances from the Legal ontology. Finally, the user flags the status of the Reason as “OK”.

Continuing with our example, let us consider the scenario that the service “Announcement of Move” contains a composite activity, for example the activity entitled “Check Payment”, which is a service comprised of several activities and modeled by another authority. The “Announcement of Move” updates the “Check Payment” service model as indicated in Fig. 11.

By selecting “Synchronize with composite service” the domain expert detects the change and keeps the “Announcement of Move” model up-to-date (Fig. 12). (The reverse/upwards synchronization procedure is also possible by selecting “Synchronize with original service model”.)

Service models, along with their semantics (design decisions, dependencies, etc.) and their status (finished, revised, and deployed), are made available to the developer through the SCS-CF, which provides tools for mapping service model elements to Web service parameters and creates a Web service package for the e-Government service that is then deployed and enacted by the SCS-RF. SCS stores different versions of the service source code. Therefore, the developer can trace the life of a service and its requested and derived changes from design time to source code versions. Moreover, design decisions help developers understand the current design and the rationale behind dependencies. During each service re-configuration, the system ensures consistency of models based on the underlying consistency criteria. To maintain the integrity of the system, both domain experts and developers need to update the service knowledge and document changes and rationale behind these changes.

Through the application scenario, we aimed to illustrate how OntoGov addresses the two challenges identified in Section 1:

- Enabling systematic response to changes: Necessary actions are initiated after a change has happened and are executed in a controlled manner, thereby enabling recovery from situations where it is desired to reverse the effects of change, generating derived changes, and preserving consistencies at each evolution stage of the service.
- Providing knowledge to deal with changes: Acting as a repository of design rationale knowledge, OntoGov helps domain experts and developers trace decisions, reasons, and dependencies among design decisions, as well as service influence factors such as legal, organizational, and domain aspects.

6. Evaluation

In this section we aim to identify the links between ontology-based service change management and change management impacts. We aim to derive several propositions that can be further verified in future research.
Fig. 9. Adding a legal reason to a design decision.

Fig. 10. Synchronizing with an updated legal ontology.
We conducted an evaluation of OntoGov following the case study method, which helps investigate a phenomenon in depth (Yin, 1989) and refers to both the identification of the key factors that may affect the impact of a system and the documentation of the impacts, constraints, resources, and outputs of the system on a set of issues under examination (Fenton & Pfleeger, 1997). The case study evaluation method was selected over a more rigorous experiment because it was not possible to manipulate the independent variables affecting the impact of our system in a controlled manner. Moreover, the case study method has been suggested as the best suited method to understand interactions among information technology innovations and organizational context (Yin, 1989). However, since single case studies may yield particularistic findings, OntoGov has been evaluated in three case studies, one at the central government level and two at the municipality level.

Evaluation was performed in a period of six months. Two sister projects were executed in the real environments of public administrations. Users were asked to design a new service in the first project and to modify an existing one in the second project. Users received training on the usage of OntoGov following its installation. OMS was used by domain experts of the public administrations, while SCS was used by employees of the IT department. Qualitative data were collected through semi-structured interviews while in some cases the on-site observation method was also used. Transcribed and documented data were analyzed using open coding techniques (Strauss & Corbin, 1990) for breaking down, examining, comparing, and categorizing data. Our aim was to reveal essential information about the dependent variables of our examination as they related to the effort and complexity of the change management process. The specific dependent variables of the evaluation were: duration, flexibility, and dependency of change process on other processes; errors and impact of errors; regulatory compliance of service; access to information during the change process; and security of process. Key factors affecting the evaluation have been identified such as the users’ profiles and their role in the organization, the IT-literacy of the users, the training received, the number of collaborating public administrations in the service, the complexity of service models and the maturity of the service. In our study, these factors were treated as control variables. A summary of categorized data is presented in Table 2. The following are the propositions deriving from our evaluation:

- In order to allow for systematic handling of changes and for provision of design rationale knowledge, effort is required to build dependency models and to capture knowledge into the system. The effort and complexity required to perform these activities when designing a new service is balanced by the time gains obtained when re-configuring a service due to the system’s ability to generate change dependency traces which leads to proposition (P1) of Table 2.

![Fig. 11. Old (left) and new (right) “Check Payment” service models.](image1)

![Fig. 12. Synchronizing composite activities.](image2)
Table 2
Case study evaluation results.

<table>
<thead>
<tr>
<th>Case study evaluation results</th>
<th>Project: design a new service</th>
<th>Project: modify an existing service</th>
<th>Proposition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Impact on project duration</strong></td>
<td>Participants reported that working with OntoGov requires additional effort and knowledge in order to develop the necessary ontologies. This has a negative effect on task duration that ranged from 10% to 30%. Time savings were noted in cases where the new service is based on the adaptation of an existing service and in cases of ontology reuse in general.</td>
<td>The tests performed indicated an approximately linear relationship between number of changed ontology artifacts (e.g., activities, inputs, and outputs) and time needed for implementing the changes. This can be attributed to the system's ability to generate change dependency traces which in turn allow users progressively see and apply change impacts.</td>
<td>P1: Ontology-driven change management has no prominent effect on the duration of the change management process.</td>
</tr>
<tr>
<td><strong>Impact on project flexibility (e.g. how fixed is the order and number of steps)</strong></td>
<td>Participants reported that with existing systems, there is a standard series of steps that have to be followed to design a new e-Service but without any formal dependency among them. With OntoGov, there is again a clear workflow but formal dependencies exist. The later allow different roles to work autonomously on specific steps. E.g., a programmer may be configuring a service based on the current process model while a domain expert may be working on the new process model version. Moreover, since ontologies are identified by unique URIs and can be accessed on the Web, distributed partners can work collaboratively to design a new process.</td>
<td>With existing systems, regulations and policies models.</td>
<td>P2: Ontology-driven change management improves the flexibility of the change management process.</td>
</tr>
<tr>
<td><strong>Impact on dependent and linked tasks</strong></td>
<td>Detailed knowledge about the service and the related design decisions as well as the domain, organizational, regulatory information must be in place. OntoGov provided the capability to represent the same knowledge and to define inter-dependencies between knowledge artifacts.</td>
<td>Without OntoGov there is a risk that part of the service model that is affected by the change is not identified. There is no systematic way to resolve which parts are affected by a change. Within OntoGov, once the domain, organizational, regulatory and other necessary information is captured and a change is performed in one of them, the system synchronizes the changed model with all models that are linked to it. Moreover, the system generates derived changes for resolving inconsistencies.</td>
<td>P3: Ontology-driven change management improves the identification of dependent changes during the change process.</td>
</tr>
<tr>
<td><strong>Impact on errors and consequences of errors</strong></td>
<td>With existing systems, logical errors generated during service modeling, if detected in further service development phases, they can be corrected without having to come back to service models leaving them invalid. With OntoGov, more logical errors are caught during service modeling. Moreover, in case errors are detected in further development phases the system permits effortless correction of affected service models.</td>
<td>Without OntoGov, more logical errors are caught during service re-modeling. Moreover, if an error is not caught automatically but later identified by a user, the system can either roll-back to a previously committed version or can support the user in resolving the error to all affected models and software components enacting the models.</td>
<td>P4: Ontology-driven change management improves the consistency of the service during the change process.</td>
</tr>
<tr>
<td><strong>Impact on regulatory compliance</strong></td>
<td>With existing systems, regulations and policies affecting the service are typically hard-coded and filed. If regulations need to be consulted, they need to be accessed and studied. With OntoGov, because regulatory issues are linked to the service model, they can be easily controlled. Most existing systems in the case studies allow users to capture information needed to perform the specific activities of the service during service modeling and access it during process execution. Additionally, OntoGov allowed participants to formally describe decisions made during the definition of the service model and reasons for these decisions.</td>
<td>With existing systems it is up to the users to ensure that the modified service does not violate any policies and regulations. With OntoGov, modified services are automatically checked against regulations and policies. Changes can be tracked and synchronized easily and at any time if necessary. The capability for documented design decisions linked to change propagation was found to be useful not only to assess the impact on software artifacts, but also during the process of obtaining approval for changes. In the absence of a model and tool such as ours, participants often relied on their past experience to understand rationale behind past decisions which can be documented in version control tools, albeit at a high level of detail. Participants reported that structured rationale documentation with links to appropriate software artifacts improves understanding of the service components and constituents in two ways during service re-configuration — by reducing the time required and by improving accuracy.</td>
<td>P5: Ontology-driven change management improves the regulatory compliance of the service during the change process.</td>
</tr>
<tr>
<td><strong>Impact on access to information needed to perform project</strong></td>
<td>With existing systems, capturing information needed to perform the specific activities of the service during service modeling and access it during process execution. Additionally, OntoGov allowed participants to formally describe decisions made during the definition of the service model and reasons for these decisions.</td>
<td>The capability for documented design decisions linked to change propagation was found to be useful not only to assess the impact on software artifacts, but also during the process of obtaining approval for changes. In the absence of a model and tool such as ours, participants often relied on their past experience to understand rationale behind past decisions which can be documented in version control tools, albeit at a high level of detail. Participants reported that structured rationale documentation with links to appropriate software artifacts improves understanding of the service components and constituents in two ways during service re-configuration — by reducing the time required and by improving accuracy.</td>
<td>P6: Ontology-driven change management improves the comprehension of the change management process.</td>
</tr>
<tr>
<td><strong>Impact on security</strong></td>
<td>With OntoGov, each collaborating user is uniquely authenticated by the system and has specific read/write/update/delete rights for every service model as well as associated ontology. Moreover, groups of users can be defined. Service models and ontologies are uniquely identified while the system prohibits unauthorized access.</td>
<td>OntoGov provides ability to check status of a service (requested/design/developed/deployed). Moreover, user activity tracking ensures that every user’s activity is logged and can be subsequently traced back. Finally, security is enforced even within a particular process as every step or activity of the underlying business model is defined with pre- and post-conditions. A condition can be the successful authentication of a particular user or a user role. This way, unauthorized actions in any part of the service are reduced.</td>
<td>P7: Ontology-driven change management enhances the granularity of security enforcement during the change process.</td>
</tr>
</tbody>
</table>
Ontology-driven change management improves the flexibility of the change management process (P2) because, although a certain rigidity in the number of steps that have to be followed in service design and re-design remains, the existence of formal dependencies between different models and steps allow different roles to work autonomously and from distributed locations.

Ontology-based change management ameliorates the risk that part of the service model that is affected by the change is not identified. Once the domain, organizational, regulatory, and other necessary information is captured and a change is performed in one of them, the system synchronizes the changed model with all models that are linked to it, leading to proposition (P3).

Ontology-driven change management improves the consistency of the service during the change process (P4) as more logical errors can be detected while versioning allows easy roll-back to consistent service models and enacting software.

The capability to represent legal, regulatory, and policy issues in ontologies and link them to service models improves the regulatory compliance of the service during the change process (P5).

The capability to keep track of design rationale knowledge during service configuration and re-configuration helps domain experts and developers better and faster understand the various artifacts such as design fragments and software components (P6).

The capability to define and assign change status (requested/design/developed/deployed), track user activity and define access rights as pre- and post-conditions at any step of the service model enhances the granularity of security enforcement during the change process (P7).

7. Related work

Service Configuration Management (SCM) is the discipline of controlling the evolution of complex software systems, helping manage changes to software and ensuring correctness and consistency of systems (Conradi & Westfechtel, 1998; van der Hoek, Heimbigner, & Wolf, 2002; Gunter, 2000; Estublier et al., 2005). SCM includes both technical and administrative practices to establish and maintain the integrity of work products using configuration identification, configuration control, configuration status accounting, and configuration audits. As such, SCM focuses and supports mainly management of product knowledge (models, specifications, documents, versions of these artifacts, etc.). Moreover, SCM does not typically describe and explain the links between regulatory and policy aspects that guide a software system and the system itself; neither does it help developers analyze the impact of changes made to these aspects during system development and maintenance (Ramesh & Jarke, 2001). An approach to combine SCM with traceability in order to help manage knowledge about the process of the development of software artifacts throughout the development lifecycle is proposed by Mohan et al. (2008), which nevertheless does not explicitly address links to regulatory and policy aspects.

Other related work can be found in the area of Web service management that involves semantic technologies. The core idea in this area is the annotation of Web service elements with terms from domain models including industry standards, vocabularies, taxonomies, and ontologies. This approach aims to alleviate terminological discrepancy in Web service descriptions that can in turn make annotations machine-processable and ease the human effort required to manage the lifecycle of a Web service, e.g., the modeling, discovery, and composition of Web services (see Arpinar, Zhang, Aleman-Meza, & Maduku, 2005; Gomez-Perez, Fernandez-Lopez, & Corcho, 2003; Cardoso & Sheth, 2003; Narayanan & McIlraith, 2003). Since 2002, two approaches have been followed by researchers in the Semantic Web services area (Verma & Sheth, 2007). One path represented a revolutionary rethinking of all aspects of semantic services and resulted in two main recommendations — the service extension of the Web Ontology Language (OWL-S) and the Web Service Modeling Ontology (WSMO). The other path took a more evolutionary approach that stayed consistent with existing standards and industrial practices and resulted in the SAWSDL (www.w3.org/2002/ws/sawSDL) recommendation. However, few of these approaches aimed at a comprehensive account of influence take into consideration the design decisions and consistency constraints for service design which would address the problem of long term, consistent evolution of services in a dynamic environment. Kim and Gil (2004) describe an interactive service-composition framework where the system performs a number of checks to ensure structural consistency of a composed model (e.g., whether all expected results are produced or whether all the needed inputs are provided). Similarly (Ankolekar, Paolucci, & Sycara, 2005), model control flow and data flow of an OWL-S process and verify whether these two dimensions are correct, both examined separately and in combination. Verma, Akkiraju, Goodwin, Doshi, and Lee (2004) 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 introduce two types of dependencies: description-based and domain constraints, whereas both of them can be easily implemented with OntoGov. Additionally, we provide service specific constraints that ensure the consistency of the service flow.

Other related work comes from business process and workflow management that typically addresses specific kinds of consistency checks like reachability analyses and consistent evolution of process instances and process models in the case of exceptions and not model-compliant process instantiations (for an overview of such methods, see Rinderle, Reichert, & Dadam, 2004). Related work includes the approach of pockets of flexibility where placeholders are left within the process and are to be specified at runtime by obeying certain constraints (Sadiq, Sadiq, & Orlowska, 2001). Furthermore, there is work on adaptive process management that aim at enabling more flexible process executions (e.g., Sadiq, Orlowska, & Sadiq, 2005). Such approaches allow the execution of a partially defined process template; once an unspecified sub-process is reached, a process expert can complete the specification of the sub-process. The sub-process is then verified based on constraints (e.g., composition constraints), which are used to restrict the composition possibilities. An approach for ensuring the integrity of process instances based on rule-based process adaptation is introduced in Mueller, Greiner, and Rahm (2004). Rules are applied when certain conditions in the problem domain apply. A similar approach is presented by Adams, ter Hofstede, Edmond and van der Aalst (2006) where parts of processes are left to be specified at runtime using selection rules. More recent and promising work is a framework for detecting semantic conflicts when modeling process templates, applying ad hoc changes at process instance level, and propagating process template modifications to already running process instances (Ly, Rinderle, & Dadam, 2008). This approach goes beyond verifying a process model into verifying real ad hoc changes at runtime. However, expressiveness of the presented constraints is limited while the authors state that they plan to examine how the information referred to by semantic constraints can be better organized (e.g., within an ontology) in order to decrease evaluation effort. Moreover, this approach has not been implemented in an information system and tested in real world cases yet.

Related work also includes approaches related to the workflow evolution. Casati, Ceri, Pernici, and Pozzi (1998) define 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. We argue that the applicability of approaches that aim to dynamically resolve changes — 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 — is constrained in the e-Government domain. Taking into account an
enormous number of public services and dependencies between them, as well as the complexity of interpreting and implementing changes in government regulations, the process of consistently re-configuring e-Government systems reflecting external changes (the so-called static modification) seems to be quite complex, already. Indeed, an efficient change management system must provide premises to allow the progressive refinement of the e-Government service without rewriting it from scratch, and must guarantee that the new version of the service is syntactically and semantically correct.

There exist “soft” approaches to change management, which are often linked to business process re-engineering (Hammer & Stanton, 1995). Such approaches focus on making people involved in changing processes understand and accept changes and make them happen with the emphasis being on providing information and training, and enabling communication between stakeholders. Stakeholders typically exchange information about how to resolve the change, but not semantically more complex structures that can be machine-processable, since they miss a commonly agreed description of problems. Moreover, “soft” approaches require a growing number of highly skilled personnel, making the maintenance costly. Finally, changes that affect the system are resolved and propagated in an ad hoc manner.

Our work focuses on managing the different levels of e-Government services (from service models to executable Web services) in their socio-technological context. As such it follows a pragmatic approach which supports dealing with generic domain constraints which can be user-defined, extended, or specifically configured according to an organization’s own business rules and compliance requirements.

8. Discussion

In this section, we present the contributions of our research, its limitations, future research directions, implications to research, and practice and policy implications.

8.1. Research contributions

This research makes the following contributions by presenting a novel approach to change management of e-Government services by means of ontologies. Specifically, this research:

• Presents a conceptual framework defining the fundamental e-Government service change management models and a set of such reference models, represented in ontologies.
• Presents an approach and a software framework for representing changes in service models, finding inconsistencies caused by a requested change, generating and propagating changes to affected service artifacts, and alerting users about such changes.
• Presents an approach and a decision framework for representing knowledge that helps users deal with changes in e-Government services.

8.2. Limitations

Generalization of the results from this research must be done with caution in light of its limitations. As far as ontology engineering is concerned, although there are no constraints regarding the depth of ontology specialization (e.g., we may define the Legal-Federal ontology that contains the entities representing the laws that hold at federal level, the Legal-State ontology that specializes the Legal-Federal ontology by extending it with the knowledge related to the federal state laws, and the Legal-Municipality ontology that extends the Legal-State ontology with some regulations), our approach is currently limited to including entire models rather than including subsets. Also, when a model is reused, information can only be added, and not retracted. Implementation-wise, we illustrated our approach by applying it to Web service-based e-Government systems. In principle, our approach can be generalized to other technologies used in the development of e-Government systems; however, further work is required to reach this stage.

Another limitation of OntoGov is linked to the time required to build and maintain the semantics-based service models (including the associated ontologies). The case studies results indicate that OntoGov demonstrates marginal improvements if there are no frequent changes or if it is used primarily for designing new services without reuse of past models. Moreover, OntoGov is ineffective in cases where the semantics of a service are modeled in a proprietary fashion by each collaborating organization, therefore not allowing organizations to take advantage of OntoGov consistency enforcement and change propagation capabilities in cases where a change is made in an ontology that should be modeled in a shared manner (such as the Legal ontology).

8.3. Future work

In the future, we want to extend our approach by suggesting changes that can improve e-Government services. This can be done by monitoring critical issues in the enactment of services, for example, activities that cause delays are candidates for optimization. Moreover, improvements can be derived by taking into account the end-users' complaints, e.g., end-users might not be satisfied with the quality of service, since they have to supply the same information several times. Further field studies are necessary to enhance the generalizability of our approach. Several application-related factors, such as complexity and size of service development environments used, should be considered when applying the results of this research. The effectiveness of our approach as well as the propositions developed should be validated by detailed empirical studies for various types of services and public administrations.

8.4. Research and practice implications

This work has important implications for research on service management. Research in the area of e-Government services can benefit from formal verification approaches of the service description as well as in the use of formal methods for achieving consistency when a change-induced problem arises. Our work also identifies propositions related to the impact of ontology-driven change management on e-Government services.

This research contributes to practice by proposing a set of reference ontologies pertinent to e-Government service change management that can be used by domain experts as templates for the development of application-specific ontologies. Moreover, domain experts can model domain-specific consistency rules. We note that the proposed formal mechanisms for consistency preservation and change propagation are realized independently of the content of ontologies. As such they can function in any specializations of the reference ontologies. Finally, practitioners can readily use the OntoGov open source software in their e-Government service deployments.

Our approach and software framework can help when a development team is tasked to make changes. While performing the changes, our system identifies errors even in an early modeling phase and generates specific suggestions on how to fix errors based on the type of errors and the situation at hand. Moreover, it improves understanding of the service description and helps public administrators avoid pursuing unpromising design alternatives. While the benefits to a team that was not involved in the original development are apparent, it should be noted that even a team that did implement the original system may benefit from this knowledge, especially in cases of services that are complex and span different public authorities. This knowledge will help the team understand past decisions and the impact of changes on software artifacts. Moreover, in cases in which the development of the service is outsourced, the knowledge documented in the system will
help retain at least part of the knowledge that would have been lost otherwise.

In the aforementioned cases, the benefits of the system capability to support distributed development and change management are apparent. Further, we note that the Legal, Organizational, and Domain ontologies may be shared between several service models. The same applies for Design Decisions and Reasons. Other teams may find this knowledge useful by generalizing and applying it in different contexts. In essence, this knowledge can lead to the development of general best practices that can be shared across services or across levels of administration.

8.5. Policy implications

The proliferation of e-Government services has been leveraged by, among other factors, the adoption of access-to-information laws by governments from around the world. Freedom of information laws are written to increase governmental transparency and to advance accountability (Potrowski & Rosenblum, 2002). This reform in government information policy parallels a global movement of intergovernmental and nongovernmental organizations, which have pressured countries to advance the norm of transparency (Relly & Sabharwal, 2009). Transparency however, is not limited to giving citizens and business access to governmental information and data. As observed by Florini and Stiglitz, transparency refers to “the degree to which information is available to outsiders that enables them to have informed voice in decisions and/or to assess the decisions made by insiders” (Florini & Stiglitz, 2007, p. 5). Such a definition of transparency promotes the idea of allowing citizens and businesses to have access to the rationale behind governmental decisions.

OntoGov provides a technical infrastructure capable of capturing design rationale regarding the design and implementation of e-Government services and therefore technically functions as an enabler for transparency in decision making. Nevertheless, the technical infrastructure needs to be supported by policies mandating the documentation of decisions and specifying the exact ways in which decisions should be documented and by whom. Agencies should seek capturing service design information that is particularly useful and relevant to service stakeholders and then proactively advertise it so people are alerted to it.

Beyond the issue of transparency, policymakers and service providers are also interested in strategic information gathering to support the development of services. An approach such as OntoGov allows the systematic capture of knowledge regarding service design, re-design, and delivery that may be instrumental in better understanding the issues and pinpoints underlying e-Government services. The use of these data to facilitate constructive dialog may further the building of critical relationships between public agencies involved in service provision, thus strengthening the public service infrastructure in a community.

Moreover, documenting decisions with a technical infrastructure like OntoGov can differ significantly from the way the same information would be captured through paper and print formats. For instance, through OntoGov information about service decisions has the potential to reach a broader audience than traditional government documents and create opportunities to utilize service decisions in more ways than previously possible. However, this fluidity in audience and purpose raises the need to develop policies about how to handle digital information related to the ways e-Government services are designed and maintained. For example, policies should specify if service design decisions are accessible by citizens or whether such information is intended for public officers only.

From an infrastructure perspective, our approach for managing changes in e-Government services requires collaboration within and across agencies. A collaborative approach for service management can enable participants to develop economies of scale and service expertise while simultaneously through collaboration providing public servants with a deeper and more thorough expert access to knowledge underlying service design. The emergence of “one-stop” e-Government service has already created the need for policies focusing on collaboration within and across agencies for the purpose of delivering composite services that span more than one department or agency. Such policies need to be extended in order to cover issues linked to the maintenance and evolution of composite services. For example, policies should specify how a department or agency should trigger the service change process in case an amendment to a law is detected. Policymakers should bear in mind that the effective use of a collaborative infrastructure also depends on a trustworthy environment and on other agreements referring to political, legal, and economic issues, and that these should be addressed.

Acknowledgments

We thank the European Commission for its funding and our partners in the OntoGov project for their contributions to the development of OntoGov.

References


