Transactional e-Government Services: an Integrated Approach

C. Vassilakis[1], G. Laskaridis[1], G. Lepouras[1], S. Rouvas[1], P. Georgiadis[1]

[1] e-Gov Lab, Department of Informatics, University of Athens, 15784, Greece
{costas, glask, gl, rouvas, p.georgiadis}@e-gov.gr

Abstract. Although form-based services are fundamental to e-government activities, their widespread does neither meet the citizen's expectations, nor the offered technological potential. The main reason for this lag is that traditional software engineering approaches cannot satisfactorily handle all of electronic services lifecycle aspects. In this paper we present experiences from the Greek Ministry of Finance's e-services lifecycle, and propose a new approach for handling e-service projects. The proposed approach has been used successfully for extending existing services, as well as developing new ones.

1 Introduction

e-Government development may be quantified through a set of indicators to measure comparative progress. *eEurope* has published a list of 20 basic public services [1] which should be considered as the *first steps* towards "Electronic Government", along with a methodology for measuring government online services [2]. It is worth noting that among the basic public services listed in [1], 75% of them include e-forms filling and submission. Moreover, the citizen interface must be connected to the back-office in order to provide the transactional capability, which will allow it to offer the rich mix of services that customers want and governments have promised [3].

However, transactional services widespread currently lags behind the expected level, taking into account potential offered by technology. Besides structural reforms, and the adaptation of a customer-centric model [4], the development and maintenance processes for such services are quite complex: firstly, service requirements must be analysed; secondly, the service has to be designed, considering functional requirements, user interface aspects, and administrative issues. Implementation and deployment should then commence, and the e-service platform should be linked to some installed IT system, for exchanging data. Finally, when changes to the service are required, the whole process must be carried out, resulting in costs and delays.

Experiences from an electronic service's life cycle

The development of the electronic tax return form service for year 2001 followed a classic software engineering paradigm, starting off with the user requirements

analysis. Four requirement dimensions were identified in this procedure (a) appropriate input forms (b) *input validation checks*, which verify that user input is conformant to tax legislation rules (c) forwarding of collected data to a back-end system for the tax computation process and (d) collection of the results of the tax computation process and user notification about the final result. These four requirement dimensions are in fact the "computerised" counterparts of the paper-based tax return form and tax computation process; some additional issues had also to be faced due to e-service operation (a) issuing of *identification credentials* service users (b) personal data management for authenticated users and (c) provision of a full service administration framework for the GmoF's administration team, including account management, database backup and recovery, statistics reports etc.

The design phase followed, producing detailed specifications of the various components and procedures, along with specification of the interfaces between the different stages of the information flow. Subsequently, each portion of the work was implemented "autonomously", and an integration step consolidated the different work dimensions into a single, operational platform. After the service became operational, maintenance tasks were performed, mainly for the purpose of correcting appearance problems and modifying or enhancing input validation checks.

During the life cycle of the project, a number of shortcomings of the followed approach were identified, which led to increased product delivery times and the need to involve more staff. These shortcomings are discussed in the following paragraphs.

1. Knowledge existed *implicitly* within the organisation, usually under the possession of experienced individuals, rather than stored in some publicly accessible repository in an explicit form. This affected all subtasks dealing with the organisation's *business logic*. During interviews, the analysis team often collected partial, or contradicting descriptions of the rules that applied to different cases.
2. User interface design did not include adequate domain expertise into forms. For instance, semantically related fields should be placed close together on the form for easier access. The user interface should also compensate the lack of expert assistance offered by tax officers in local tax offices.
3. Code reusability remained low, since code incorporated business rules, presentational elements, administrative issues and data repository accesses in an environment of increased cohesion. Thus changes to one of these issues triggered modifications to the other dimensions.
4. *Communication with back-end systems*. A full transaction processing cycle usually involves data exchange between the service platform and some organisational information system, either in an on-line or an off-line fashion. In both cases, building interfaces tailored for each case is not a good practice, since it requires substantial programming effort to address the peculiarities of the back-end system.

Regarding these issues, a number of technological solutions are available. Recording knowledge in an explicit and reusable format is addressed by knowledge management tools, while a number of research projects introduce novel methods to knowledge management ([5], [6]). For the service development and deploying phases both commercial products (e.g. XMLForms™ and Oracle E-Business Suite™) and Open Source solutions. In the standards area, the W3 consortium has published the *XForms* specification, an XML-based standard for specifying Web forms, while user-centric approaches for interfaces are discussed in the ISO 13407 standard.

Proposed approach

Although the state-of-the-art provides sufficient tools for tackling the various phases of e-service lifecycle, these phases are still handled in isolation. To provide an alternative solution for the above-described problems one has to adopt a layered approach that introduces higher levels of abstraction (enhancing maintainability and re-usability), isolate knowledge from code, allow asynchronous development of modules and enable people with domain expertise to implement new services.

This approach shifts implementation focus from programmers to domain experts, who possess the knowledge needed to process the data gathered through the e-service. In this view, an e-service consists of the e-form and the processing of the submitted data. Processing has two phases: validation and information extraction. In the first phase submitted data is checked against a set of rules and, depending on the result, the user may be asked to change and re-submit the form. Although some simple error checks (e.g. date and number checking) are usually coded by programmers, for more complex dependency checks domain expertise is of the essence. For example, a check "if user selects field 170 and 180 and has less than two children, then field 190 has to be completed according to the law 1256/98" requires deep domain knowledge. So far, either the programmer worked closely with domain experts to program these checks, or the e-service gathered data that were processed manually, resulting in late error detections. The proposed approach enables experts to play a much more active role and carry out a much greater percentage of the implementation effort.

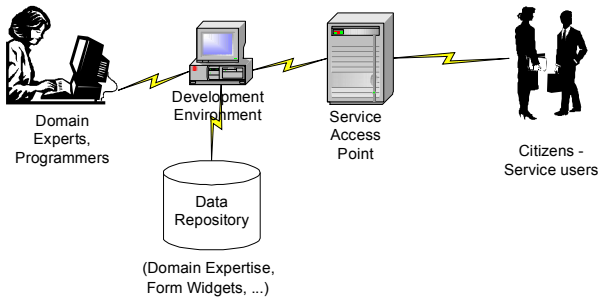

Fig. 1. System Architecture

As illustrated in Fig. 1., the development environment employs a data repository, where the ingredients of an e-service can be stored. These can be the widgets that compose the e-form along with their properties (type, multilingual labels, etc.) and the domain expertise needed to implement the service. Domain experts and/or programmers can use the development environment to implement and activate new services. The development environment has been designed to be friendly and intuitive, since a significant portion of its expected users (i.e. the domain experts) is not usually too familiar with the programming issues concerning the implementation of an e-service. To this end, an appropriate set of tools is provided.

Service development begins with the definition of the *forms* that comprise the service, and each form is then populated with the respective fields and labels. Fields and field groups may be directly drawn from the data repository, promoting thus

uniformity between various services and enhancing reusability. For instance, the area displaying the user's personal data may be defined once, and then be used in all taxation e-forms. If no suitable component exists, the developer may create a new one, by entering information such as the description, its type (string, arithmetic, etc.), whether it is editable or not, its appearance on the screen etc. Additionally, a high-level specification of data interchange with the organisation's IT systems is provided, while *validation rules* may be attached to fields. A second level of validation rules may be attached at *form level* and *service level*, to cater for checks involving multiple components. Validation rules are defined using *semantically rich elements*, e.g. "the total husband's income from salaries", rather than implementation-oriented terminology, such as *table4_field22*. This enhances readability and facilitates maintenance, since the level of abstraction remains high and semantic information in retained in this representation. Finally, domain experts would also provide the test cases to verify the validation process.

The adoption of the proposed approach offers significant advantages: Firstly, development is faster since previously developed and tested e-service components can be directly used in new services; this allows for cost reduction as well. Secondly, maintenance is facilitated , since it is easy to locate the components and services affected by some change either in legislation or policy. Moreover, changes are now mainly conducted by domain experts. Thirdly, consistency is increased, through the introduction of a central repository for knowledge storage. Uniformity in the look and feel may be also achieved through usage of service templates. Finally, *multiple dissemination platforms* may be supported, through appropriate content generators that create content suitable for a variety of platforms (Web, WAP, etc).

Conclusions

In this paper we presented experiences from developing and maintaining a set of e-services for the Greek Ministry of Finance. The traditional software engineering approaches employed in the first development phases proved to be inadequate in handling all aspects related to the lifecycle for electronic services. In the second phase we used a new approach, together with appropriate software tools, which allowed for using higher levels of abstraction, enhancing thus the maintainability, portability and reusability of the project's results, and reducing overall development time.

References

1. eEurope, "Common list of basic public services"
2. eEurope, "eGovernment indicators for benchmarking eEurope".
3. Vivienne Jupp, "eGovernment – Lessons Learned, Challenges Ahead", eGovernment Conference: From Policy to Practice, 29-30 November 2001, Charlemagne, Brussels.
4. Frank Robben "(Re)-organising for better services", eGovernment Conference: From Policy to Practice, 29-30 November 2001, Charlemagne, Brussels.
5. Know-Net Consortium, "Manage Knowledge for Business Value"
6. DECOR Project, "Delivery of context-sensitive organisational knowledge"