Integrating e-Government Public Transactional Services in the Public Authority Workflow

Costas Vassilakis

Dept. of Informatics and Telecommunications, E-Gov lab, University of Athens, Panepistimiopolis, 15784, Athens, Greece, costas@e-gov.gr

George Lepouras

Dept. of Informatics and Telecommunications, E-Gov lab, University of Athens, Panepistimiopolis, 15784, Athens, Greece, gl@e-gov.gr

Stathis Rouvas

Dept. of Informatics and Telecommunications, E-Gov lab, University of Athens, Panepistimiopolis, 15784, Athens, Greece, rouvas@e-gov.gr

Panagiotis Georgiadis

Dept. of Informatics and Telecommunications, E-Gov lab, University of Athens, Panepistimiopolis, 15784, Athens, Greece, p.georgiadis@e-gov.gr

Abstract. Documents submitted by citizens through electronic services deployed in the context of e-Government must usually undergo processing by some organisational information system, in order to complete the citizens' requests and for the reply to be returned to the citizen. The integration, however, of the e-service delivery platform and the organisational information system is often hindered for a number of reasons, including security considerations, platform diversity or idiosyncrasies of legacy information systems. In this paper we present a generic method for providing seamless communication between the two platforms, enabling the full integration of documents submitted through electronic services into the organisational workflow, leveraging thus the quality of services offered to the citizens and facilitating e-service development and operation.

Keywords: Electronic services, e-government, integration, communication, security, scheduling

Reference to this paper should be made as follows: Vassilakis C., Lepouras G., Rouvas S. and Georgiadis P. (2003) "Integrating e-Government Public Transactional Services in the Public Authority Workflow", Electronic Government J., vol. 1, Inderscience Publications

Biographical notes: Costas Vassilakis is a visiting associate professor in the University of Peloponnese and a researcher in the e-Gov lab of the University of Athens. George Lepouras is a Costas Vassilakis is a visiting assistant professor in the University of Peloponnese and a researcher in the e-Gov lab of the University of Athens. Stathis Rouvas is a researcher and a Ph.D. candidate in the e-Gov lab of the University of Athens. Panagiotis Georgiadis is an Associate Professor in the

University of Athens, Secretary General for Public Administration and e-Government in the Ministry of Internal Affairs and head of the e-Gov lab.

1 Introduction

According to the European Commission [1] "*transaction services, such as electronic forms, are perceived as the future of electronic government*". Transaction services, in general, allow users to submit electronic documents containing the data they want to communicate to the public authority (PA) delivering the service. These data are then transferred to the PA's organisational information system, where they are processed according to the workflow defined by the PA's business rules. When the processing is complete, a completion notification should be returned to the citizen that initially submitted the document, containing information about the result of the processing and/or for further actions that must be performed by the citizen. The full processing cycle in the context of transaction services is illustrated in Fig. 1.

The procedure for forwarding electronic documents submitted by the citizens to the organisational back-end and for relaying the results of the processing from the back-end to the citizen through the service delivery environment is not always fully automated, leading to delays in the processing cycle and a need for human intervention. The main reasons for not automating this procedure are the following:

1 *Security*. An on-line connection channel between a publicly accessible service delivery environment and the organisational information system is a potential security hazard, since it may be exploited by attackers to penetrate the back-end systems. Often, proprietary and undocumented protocols are used to implement the communication, inhibiting thus the enforcement of strict firewalling rules (address/port-based and/or content-based [2]) that would alleviate the problem.

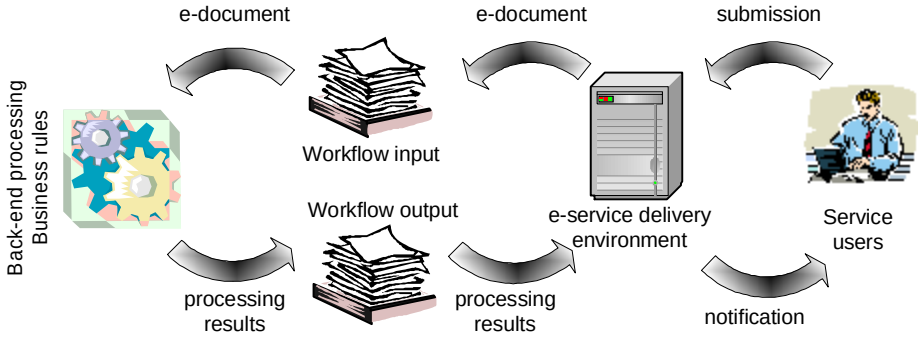

Fig. 1. Full processing cycle for transaction services

2 *Platform diversity*. In most cases the electronic services delivery platform is developed and managed separately from the back-end IT system, thus different software and hardware products are used in each environment. This may lead to

various compatibility problems, such as information modelling mismatch, incompatible national character set encoding, different settings in number precision etc.

3 *Inflexibility of legacy systems.* Most back-end systems are designed to ***deliver services only***, i.e. wait for incoming requests and execute the designated operation. In order to support the complete processing cycle for transaction services, these systems must act as ***clients*** as well, arranging for "pushing" the results obtained from processing the submitted e-documents to the service delivery environment, which will produce the appropriate notifications to the users.

4 *Task scheduling complexity*. Some e-documents must be immediately processed by the back-end systems, since the user is waiting for an on-line response; in other cases document processing may require human intervention or it would be preferable to be deferred for some off-peak time frame. Legacy systems often support only one type of task scheduling (immediate or batch), impeding the realisation of optimal scheduling policies.

5 *Change management.* When changes in the delivered electronic services occur, usually the service delivery environment is readily updated whereas the back-end IT system is adapted at a slower pace. This lack of cohesion would introduce problems in an automated communication scheme.

In this paper we present a communication scheme between electronic services delivery platforms and back-end IT systems that fully supports the processing life cycle of transaction services, allowing documents that have been submitted electronically to be integrated into the organizational work flow. The proposed communication scheme tackles all issues that hinder the establishment of direct communication between the two platforms, facilitating thus fully automating processing, smaller turnaround times and removing the need for human intervention.

The rest of the paper is structured as follows: Section two presents related work in this area. Section three outlines the architecture for the proposed communication scheme, while section four evaluates the approach by illustrating how the issues discussed above are tackled. Finally, in section five conclusions are drawn.

2 Related work

Insofar, work targeted to the provision of a generic framework for the integration of web services and organisational workflows has not been reported. Organisational needs have been addressed in a case-per-case basis by system implementers and integrators, who employ a number of paradigms facilitating communication between (possibly diverse) platforms.

One of the first approaches used for implementing communication between different platforms has been the *remote procedure call* paradigm (RPC [3]), which allows programs running on specific machine to execute code fragments on a different system, in a fashion similar to local procedure calls. The RPC paradigm, however, does not incorporate strong security mechanisms and TCP/IP port assignment is not fixed, hindering the use of strict firewalling rules. Moreover, resilience against

transient failures is minimal, no scheduling facilities are provided and both systems must be "in sync". CORBA ([4], [5]) is a more complete approach providing a middleware layer with features addressing platform heterogeneity, reliability and failure resilience; however CORBA-based systems do not generally support scheduling policies and change management, while most versions tend to be more complex and resource-consuming than is actually needed in an electronic service delivery environment. Additionally, CORBA-based systems are difficult to protect using firewalls and a compatible ORB is required on both ends.

Recent developments for platform communication include the XML language [6], the SOAP protocol [7], and the XML-RPC paradigm [8]. The XML language is the emerging standard in platform information exchange, since it allows for incorporation of structure and semantics in the exchanged messages, but this standard only defines the syntactic language rules, and does not deal with issues such as security, scheduling, or –more generally- message handling. SOAP is an XML-based messaging protocol, defining a set of rules for structuring messages that can be used for simple one-way messaging and for performing RPC-style request-response dialogues. The SOAP approach involves a listener, but security provisions are minimal, whereas no scheduling alternatives are provided (requests are always executed immediately) and no facilities are provided for handling change management. Finally XML-RPC is a variant of the RPC paradigm, where request parameters and responses are encoded as XML messages instead of using the marshalling and unmarshalling procedures of standard RPC. XML-RPC may be also tunnelled through the HTTP protocol. These enhancements, however, do not address any of the aforementioned issues regarding the use of the RPC mechanism in the context of electronic services.

3 The Communication Scheme

The proposed communication scheme encompasses two software modules, namely the *Service Delivery Environment Agent* and the *External Information System Agent.* The *Service Delivery Environment Agent* is an autonomous software module running on the platform hosting the e-service delivery environment. This software module enables the submission of ***requests*** to the PA's back-end system (or, more generally, to any system external to the service delivery platform) and the retrieval of the respective results. The External Information System Agent is again an autonomous software module running in the environment of the PA's IT system (or in the environment of any system that the service delivery environment needs to communicate with) and arranges for interception of the requests originating from the Service Delivery Environment Agents, and placing the e-documents contained in these requests in the input queue of the organizational workflow. Moreover, the External Information System Agent undertakes the responsibility for collecting the responses from the output queue of the organisational workflow and delivering them to the service delivery environment for further actions to be taken. Two ***pending action queues*** provide facilities for deferred scheduling of non-time critical requests, and for providing resilience against temporary communication failures. The overall

architecture of the proposed communication scheme is illustrated in Fig. 2. The operation of the communication scheme is described in the following paragraphs.

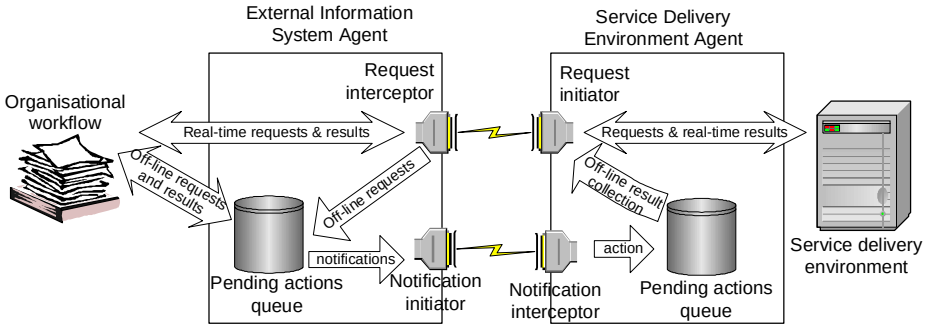

Fig. 2. Overall communication scheme architecture

3.1 Communication scheme functionality

The proposed communication scheme allows for services running within the service delivery environment to submit requests for e-document storage and retrieval from the back-end system. In order to submit a request, the electronic service should use an appropriate API, which is implemented in a library that must be loaded by the electronic service. Currently a Java [9] implementation for this library is provided, and implementations in the PHP [10] and ColdFusion [11] scripting languages are underway.

Each request is characterised either as *real time*, if it must be instantaneously processed and the results immediately returned[1], or as *off-line*, if the processing of the request may be deferred. An example of a real-time request is the event that a service user requests to view a submitted document; this request should be executed immediately, since the user is waiting at his client device for the document to appear. If completion of the request is not immediately possible, an appropriate error should be reported. On the contrary, when the user has filled in and submitted a new e-document, the organisational policy may allow of a two-day time window for the e-document to be processed in the organisational back-end and the reply to be returned; in this case, the execution of this request within the organisational back-end may be deferred. The characterisation of a request as *real-time* or off-line is dependent on the request semantics and organisational policies.

The *request initiator* module of the Service Delivery Environment Agent arranges for forwarding these requests to the External Information System Agent, where it is received by the *request interceptor* module. This communication is performed using either RMI [12] or XML [6] messages. In both cases, transferred messages may be encrypted by employing SSL or TLS [13].

Upon receiving an incoming request, request interceptor inspects the associated real time characterisation. Requests tagged as *real-time* are immediately forwarded to

[1] In this case the related procedure in the organisational workflow must be fully automated

the organisational workflow for processing, the results are collected and returned to the service that submitted the original request. This mode of operation is equivalent to the on-line services offered by databases, transaction monitors etc.

The actual method of communicating with the organisational workflow for the purposes of submitting or extracting electronic documents is heavily dependent on the implementation of the workflow. In some cases, database tables must be appropriately queried to retrieve documents or populated in order to store documents; other implementations require the execution of a custom program (usually provided with the workflow environment) that will read or modify the workflow e-document repository. Certain workflow engines, finally, provide programming interfaces (APIs) through which programs may interact with their electronic document repository. The *request interceptor* handles these cases through appropriate configurations, as detailed later in this section.

When an incoming request characterised as *off-line* is received, the request interceptor places it in the *pending actions queue*, a system repository residing in the environment of the organisational information system, and notifies the requesting application that its request has been successfully received and stored. A dedicated procedure, the *pending actions queue dispatcher*, periodically scans the pending actions queue to determine if there exist jobs that can be executed. The decision on whether a specific job in the pending actions queue will be scheduled for execution depends on the following parameters:

- *current system load*. System administrators may configure the pending actions queue dispatcher so as not to spawn new tasks when the system load has exceeded a certain limit. This facility is provided for enabling system administrators to favour the execution of real-time tasks (which are more time critical) and to guard the system against overloading.
- *organisational policies for batch job execution*. The organisational policy may specify certain time slots for batch job execution, such as weekends, night hours or off-peak periods. In a more refined environment, different policies may apply to different tasks; for instance a request to simply store a tax return form document may be allowed to run any day in the week, but requests to process such documents and compute the tax due may only be allowed to run during weekends. The pending actions queue dispatcher can be configured to take into account such policies and appropriately schedule off-line requests.

Once the pending actions queue dispatcher has determined that a specific request in the pending actions queue can be scheduled within the organisational workflow, the suitable method is employed for storing or retrieving the relevant document, similarly to the case of real-time requests, described above. An issue raised in this case is that when the processing of a request has been completed, the electronic service that originally submitted the request may not be active, being thus unable to collect the response. For example, if a citizen submits a tax return form on Monday and the computation of the tax due is performed on weekends, it is unrealistic to assume that the citizen will remain on-line for this period to collect the reply. Therefore, replies to off-line requests should be spooled and returned to the service delivery environment in an asynchronous manner. Delivery of replies to off-line requests is facilitated by employing the *pending actions queue* in cooperation with a signalling mechanism. More specifically, when the pending actions queue dispatcher forwards an electronic

document to the organisational workflow, it arranges for collecting the reply and for signalling to the service delivery environment that the request has been processed and the reply is ready. The reply is stored with appropriate tagging in the pending actions queue, and a *notification event* is posted to the service delivery environment to signify that the result is ready.

A *notification event* contains all information that is required by the service delivery environment to determine which result has been prepared and its mapping to the initial request; special provisions have been made for the case that a single "prepared result" contains the replies to multiple requests, which have been packaged into a single container for practical or optimisation purposes. The delivery of the notification event to the service delivery environment is undertaken by two software modules, namely the *notification initiator*, running on the installation hosting the organisational information system and the *notification interceptor*, running on the premises of the service delivery environment. The *notification initiator* module accepts requests for posting notifications to the service delivery environment, verifies their correctness and arranges for their delivery to the *notification interceptor*. Upon receipt of a notification event, the notification interceptor places a suitable entry in its local pending actions queue (different than the pending actions queue residing in the premises of the organisational information system). A local *pending actions queue dispatcher* examines this queue and processes its entries are processed in order to fetch the results of off-line requests in the service delivery environment, completing thus their processing cycle. Processing of an entry of this pending actions queue usually maps to the execution of a real-time request, specifically crafted for the purpose of collecting the specific type of results.

This mode of operation is the counterpart of batch process execution offered by most legacy systems. For requests serviced in this mode, there is no need for the organisational workflow to provide fully automated procedures for their execution, since the results are not expected immediately in the context of the delivered service.

The pending action queues in the premises of the service delivery environment and the external information system are also used for providing resilience against transient errors. More specifically, when an off-line request cannot be forwarded from the service delivery environment to the external information system due to a communications failure, the request initiator stores the request in the pending actions queue and its transmission is retried after a period of time. Real-time requests, on the other hand are not stored in the pending actions queue, since their semantics dictate that immediate execution is called for. On the side of the external information system, notification events that cannot be posted due to communications failures are similarly stored in the local pending actions queue and their transmission is retried after a time interval.

3.2 Configuring the communication modules

In order to provide a generic and flexible communication framework fulfilling all requirements for information exchange in a service delivery environment, each module of the Service Delivery Environment Agent and the External Information System Agent is fully configurable through a set of parameters:

1 The request initiator may be configured to selectively forward some request types to a specific back-end system (through the associated External Information System Agent), while some other request types may be forwarded to another back-end system. For instance, in an environment delivering taxation services, requests for storing documents containing VAT statements may be forwarded to different back-end system than requests for storing documents containing tax return forms. Moreover, it is possible for system administrators to define *alternative execution methods* for a specific request. In such a case, the request initiator module will sequentially try the defined methods, until an attempted method succeeds. This feature is provided to facilitate usage of *failover systems*, i.e. systems that organisations install to undertake the provision of specific functionalities when the "normal" service provider is not functioning or unreachable. For example, a municipality offering electronic services may install two systems *S1* and *S2* hosting the citizen registry, and configure the request initiator to forward registry-related requests *first* to the system S1 and, if this attempt fails, to system S2. Under this configuration, normal operation requests are served from system S1 (the *primary system*); if system S1 is not operational (e.g. due to malfunction, maintenance or communications failure), and thus the request initiator will fail in its attempt to contact it, requests will be forwarded to system S2. It should be noted that the request initiator does not undertake any responsibility for synchronising the information repositories of multiple systems providing the same service, in the presence of update requests; information repository synchronisation should be provided by the systems themselves. Finally, the ability for defining alternative execution methods *without any order* is provided; in such a case, the request initiator will forward the request to a randomly selected service provider from within the available pool. This configuration setup may be used for implementing load balancing.

2 The request interceptor may be configured regarding the details of how each incoming request is integrated in the workflow of the organisational information system and how the relevant results may be retrieved. Since workflow implementations radically differ in this aspect, the request interceptor can be configured to support different methods of interacting with the workflow engine:

- *database access*. When this option is used, the notification interceptor will store incoming electronic documents in designated database tables or will query database tables to produce electronic documents as results. An option for executing stored procedures is provided as well.
- *external program invocation.* This option provides the facility for executing an operating system level command, possibly complemented with command line parameters. The electronic document in this case is fed as the input to the invoked program, and the program output is collected as a reply to a request. Wrappers are also provided for external programs requiring to read or store the e-document to an operating system file.
- *procedure invocation.* For workflow engines providing a concrete API for querying or populating the electronic document repository, the request interceptor provides the facility for invoking a procedure, written in the appropriate programming language, which will arrange for appropriately interacting with the information repository to store or retrieve electronic

documents. This feature may only be exploited in environments supporting dynamic code loading, such as the Java platform and operating systems with provisions for dynamic library loading.

- Ø *file storage*. If none of the above methods is applicable to the workflow engine at hand, the request interceptor may store the incoming request to an operating system file, for further processing in semi-automatic or manual manner.

The request interceptor may be additionally configured to support *alternative execution methods* for each type of incoming request. Similarly to the case of the notification initiator, alternative execution methods are tried sequentially until one of them succeeds. This provision allows for exploiting multiple access paths to the workflow repositories that may be provided.

3 The notification interceptor may be configured to designate which action(s) should be scheduled for execution, as a response to each kind of notification event received. Action designation is performed by placing relevant entries in the Service Delivery Environment pending actions queue, which is then read by the pending actions queue dispatcher.

4 For the pending actions queues (both for the one hosted in the services delivery environment and for the one hosted in the external information system) the system administrators may configure the period that entries are examined, the clock hours that each type of entry may be processed and the system load over which execution of pending tasks should be precluded.

4 Assessment of the Proposed Scheme

The proposed scheme has been installed and tested in the context of two public authorities offering electronic services. A mixture of electronic services with diverse requirements was selected, in order to verify that the scheme meets the needs posed by organisations. In more detail, the selected services included the following:

1 *submission of informational documents*, i.e. documents that needed to be submitted by the citizens in order to notify the PA authority of certain facts. These documents have no automated or semi-automated process associated with them and produce no reply to the citizen. These documents are used by PA inspectors for crosschecking purposes in samples of the population.
For these cases, three generic services were required: creation of a new document (blank, except for the user data which were filled in), submission of a document and viewing of a document. In one case, document editing was also required. All services were coded as synchronous requests, since users should be able to view the documents they submitted immediately. If this requirement was relaxed, the storage request could be coded as an asynchronous one.

2 *submission of instantly processed documents*. This case includes documents that are submitted by the citizens and are instantly processed to produce a reply for the submitting citizen. Submitted documents are synchronously forwarded to the back-end where they are processed and the reply is returned as the response to the "submit document" request. The collected reply is presented to the citizen. In all

cases, the back-end procedure that processed the document arranged for its storage as well, saving thus one call from the service delivery environment to the organisational back-end. This optimisation step requires that the system responsible for processing the document can access the repository into which submitted documents are persistently stored; if this is not the case, two separate requests (for processing and storage) should be coded in the service delivery environment.

3 *submission of batchly processed documents.* In this case, documents submitted by citizens are not immediately stored or processed; rather they are spooled and processed in a batch manner, either periodically or when a certain document volume has been amassed. For these cases, document processing requests have been coded as asynchronous calls, whereas document storage tasks have been coded either as synchronous or asynchronous requests, depending on the viewing policy: for the cases that a submitted document should be immediately retrievable by the citizen (for viewing/confirmation purposes), storage requests were synchronous (and consequently separated from processing requests). For the cases that submitted documents should be retrievable only after the processing has been completed, only a single request was coded and the back-end process arranged for the storage of the citizen's document in the PA's document repository. When the processing stage is complete, a notification is sent to the service delivery environment, initiating a synchronous procedure that collects the replies batch, which is then processed to produce suitable e-mail messages to the citizens.

4 *submission of manually processed documents.* In this case, submitted documents are processed manually by the PA's staff; the processing stage possibly involves some extra-systemic procedure such as an on-site inspection, an interview etc. For these cases, the back-end procedure typically involves storing the document in the persistent storage and appropriately notifying (via an e-mail message or by raising a database flag subsequently inspected by the organisational applications) the employee in charge of this work. When the process has been completed, the PA worker enters the relevant data to the organisational application, via a procedure that has been enhanced to produce a notification to the service delivery environment. As a response to the notification, the service delivery environment spawns a process for collecting the relevant portions of the data, which are communicated to the citizen.

The IT staff in both PAs has found the communication scheme to be "manageable" and "easy to maintain in cases of changes". The separation of service code from the code implementing communication with the back-end system, in particular, was very positively commented, since it provides "a concrete task separation". Debugging has been found to be somewhat tedious, since in order to pinpoint an error, data from the communication scheme queues and the error logs have to be extracted and combined; this however has been characterised as a "typical problem in electronic services". A dislike for XML configuration files has also been recorded, particularly from staff that was not familiar with the XML language.

Regarding the issues outlined in section 1, the proposed scheme tackles them as detailed in the following paragraphs:

1 *Security*. All communication between the service delivery platform and the organisational information system is performed (a) between the request initiator and the request interceptor and (b) between the notification initiator and the notification interceptor. These software entity pairs communicate in known TCP/IP ports and exchange messages of well-documented types (XML or RMI), thus strict firewalling policies may be enforced (both address/port-based and content-based), augmenting the overall protection level of the organisational back-end system.

2 *Platform diversity*. The divergences owing to differences in hardware and software environments or related to different information schemata are handled by mappings encapsulated in the request initiator and the request interceptor. Messages are exchanged in XML or RMI, which provide rich data types, allowing for transferring of any type of data. In particular, XML messages may contain descriptions of the required conversions, facilitating thus dynamic mappings.

3 *Legacy systems act only as servers*. The proposed communication scheme always assigns the role of the server to the back-end system and the role of the client to the service delivery environment, being thus in tune with the functioning of the back-end systems. Bi-directional information exchange is handled through notifications, which are undertaken by the proposed communication scheme, not affecting the organisational information system.

4 *Task scheduling complexity*. The proposed communication scheme includes configurable scheduling engines, removing thus the need for schedulers provided by the operating system or customly written by the IT staff. The schedulers built in the communication scheme support both real-time transactions and deferred (off-line) processing.

5 *Change management*. The proposed communication scheme supports flexible mappings between the front-end information schema and the workflow information schema (within the configuration of the request interceptor), facilitating change management when the two interconnected systems are separately maintained. When a change takes effect in the service delivery environment and the respective portion of the back-end system has not been yet updated, it is possible for the administrators to configure the request initiator so that requests affected by this change are *spooled* to a local repository (e.g. a database or an operating system file), rather than forwarded to the back-end system. Spooled requests may be transferred to the back-end system when the changes in question have been incorporated into it.

5 Conclusions

In this paper we have presented a generic communication scheme for integrating an electronic services delivery environment with the organisational workflow. The proposed scheme screens service delivery platforms from legacy system idiosyncrasies, supports both immediate and deferred task execution, provides for flexible task scheduling and allows for enforcement of strict firewalling rules. The

proposed scheme has already been evaluated, in terms of functionality and performance, in the context of the one installation hosting five simple services needing to interact with the organisational workflow. Stress tests will be also performed, in order to assess the scalability of the platform; tight integration with tailorability providers (e.g. [14]) will be also addressed.

References

1 European Commission (2000) *Public Sector Information: A Key Resource for Europe*, Green paper on Public Sector Information in the Information Society, ftp.echo.lu/pub/info2000/publicsector/ gppublicen.doc

2 Northcutt S., Zeltser L., Winters S., Fredrick K., Ritchey R. (2002) *Inside Network Perimeter Security: The Definitive Guide to Firewalls, Virtual Private Networks (VPNs), Routers, and Intrusion Detection Systems*, New Riders Publishing; ISBN: 0735712328

3 Sun Microsystems (2002) *ONC+ Developer's Guide*, available at http://docs.sun.com/db?q=RPC&p=/doc/805-7224

4 Object Management Group (2002) *CORBA Basics*, http://www.omg.org/gettingstarted/corbafaq.htm

5 Bolton F. (2001) *Pure CORBA*, SAMS Publications, ISBN: 0672318121

6 World Wide Web Consortium (2001) *The XML Specification*, available at http://www.w3.org/xml

7 Newcomer E. (2002) *Understanding Web Services: XML, WSDL, SOAP, and UDDI*, Addison Wesley Professional, ISBN: 0201750813

8 Laurent S., Dumbill E. and Johnston J. (2001) *Programming Web Services with XML-RPC*", O'Reilly & Associates, ISBN: 0596001193

9 Arnold K., Gosling J. and Holmes D. (2000) *The Java Programming Language (3rd edition)*, Addison-Wesley Publishing Company, ISBN: 0201704331

10 Lerdorf R., and Tatroe K. (2002) *Programming PHP*, O'Reilly & Associates, ISBN: 1565926102

11 Hewitt E. (2001) *Core ColdFusion 5*, Prentice Hall, ISBN: 0130660612

12 Sun Microsystems (1999) *Java™ Remote Method Invocation*, available at http://java.sun.com/j2se/1.3/docs/guide/rmi/index.html

13 Thomas S. A. (2000) *SSL & TLS Essentials: Securing the Web*, John Wiley & Sons, ISBN: 0471383546

14 Loverdos C., Saidis K., Sotiropoulou A., Theotokis D. (2002) *Pluggable Services for Tailorable E-content Delivery*. OOIS, pp. 6-18