60 GOVERNMENT-WIDE WORKFLOW INFRASTRUCTURE - ENABLING VIRTUAL GOVERNMENT ORGANIZATIONS[1]

Olumide Oteniya, Tomasz Janowski, Adegboyega Ojo[2]
The United Nations University
International Institute for Software Technology
(UNU-IIST), MACAU
{gab,tj,ao}@iist.unu.edu

Many governments worldwide are establishing one-stop portals to provide access to various public services based on the needs of citizens or businesses and not the internal structure of the government. A critical support for such one-stop portals is a workflow infrastructure, supporting the matching of the needs against provided services and coordination of the implementing processes, often spanning several government agencies. This paper describes a generic workflow infrastructure for one-stop government – GovWF. GovWF supports the operations of a Virtual Government Organization – a hierarchy of agencies providing collectively a set of public services, while offering a uniform one-agency view to its customers. Conceptual and formal models are provided to rigorously describe the operations of GovWF. We describe how GovWF is implemented and also present a case study for illustration.

1. INTRODUCTION

e-Government is one of defining features of modern public administration (PA). Traditionally, e-Government efforts are targeted at improving internal efficiency in the delivery of public services, with back-office integration as the holy-grail. Lately, there has been significant shift in focus from the supply-side of public service delivery to the demand-side. This is largely due to the poor take-up of online public services. Addressing this problem requires an organizational model which simplifies the highly fragmented view of the public sector consisting of delineated agencies and possibly some private organizations providing public services (Kubicek, 2000).

Government customers, like their private sector counterparts, are demanding better services. In response, governments are adopting new practices, policies and technologies. One-stop government is one of the dominant practices for integrating public services across agencies from stakeholders' viewpoint (Wimmer 2001). A one-stop government portal organizes public services into life events or business episodes targeted at specific customers at particular times in their life.

[1] This work is funded by Macao Foundation under the e-Macao project.

[2] On leave of absence from the University of Lagos, Nigeria.

We consider a one-stop government providing seamless services as a virtual government organization (VGO). Each instance of request by a customer leads to a dynamic selection of VGO members forming an alliance to collectively satisfy the business process associated with the request. Dynamic service composition is the enabling technology for service invocation and execution within the VGO.

This paper describes a workflow infrastructure (GovWF) which coordinates service provisioning within a VGO. It dynamically binds concrete services provided by agencies to business process schemas based on user requests and other constraints, e.g. government policies. GovWF also executes business processes and propagates service outcomes to the relevant agencies to ensure internal consistency.

The rest of the paper is as follows. The notion of a VGO is defined in Section 2. Section 3 presents the workflow infrastructure for VGOs - GovWF, from concepts (Section 3.1), to formalization (Section 3.2), to implementation (Section 3.3). A case study illustrating the working of GovWF - requesting a license to open a restaurant business is presented in Section 4. The final Section 5 presents some conclusions.

2. VIRTUAL GOVERNMENT ORGANISATION

A Virtual Organization (VO) is a network of independent organizations which collaborate based on some common goal and share some resources on the basis of some policies to accomplish their goal (Witczynski et 2000). Virtual organizations are goal-oriented and dynamic in nature. The key concept underpinning VOs is the separation of functional goals and the means of satisfying them. A VO is characterized by (Mowshowitz, 1997): (1) formulation of abstract requirements, (2) analysis of concrete satisfiers and (3) assignment of satisfiers to requirements.

While a number of successful models of VOs exist in the private sector, there are only a few such examples in the public sector. The notion of VO in the public sector or VGO (and same as Virtual Government) underpins seamless or integrated public service delivery (Zhiyuan Fang 2002). A VGO typically consists of a group of government agencies and possibly some private enterprises, cooperatively providing public services on behalf of the government. These agencies may themselves consist of several departments or units that could be seen as agencies on their own.

We assume that only service specifications and process handles are published by the agencies in the VGO. Service implementation details are always hidden. To request for a service from a VGO, a customer describes its need through a service specification, if a service to fulfill the need is unknown, or through a process handle, if known. In the former case, an intermediation infrastructure is required to match the specified needs against the services provided by the agencies (Legal 2002). In the latter case, the requested process is initiated within respective agencies. In both cases, coordination of the implementing processes is necessary. A workflow infrastructure is typically used for such coordination.

3. WORKFLOW INFRASTRUCTURE FOR VGOs

We describe here a workflow infrastructure for orchestrating and coordinating services across agency boundaries - GovWF. The concepts, model and implementation of this infrastructure is described.

3.1 Concepts

The one-stop-government is synonymous with our concept of a VGO. A government portal organizes services by different agencies in a user-oriented way, according to the life events or business episodes. Each service includes a specification describing what it has to offer and a sequence of execution steps - sub-services or sub-process. Both can refer to the agency's own services/processes or to the services/processes offered by other agencies. Given a service request, the selected agency's catalogue is searched for the process which matches the specification of the requested sub-service. In general, an agency may consist of several sub-agencies and be part of a super-agency. It may invoke functions of both sub- and super-agencies; see Figure 1.

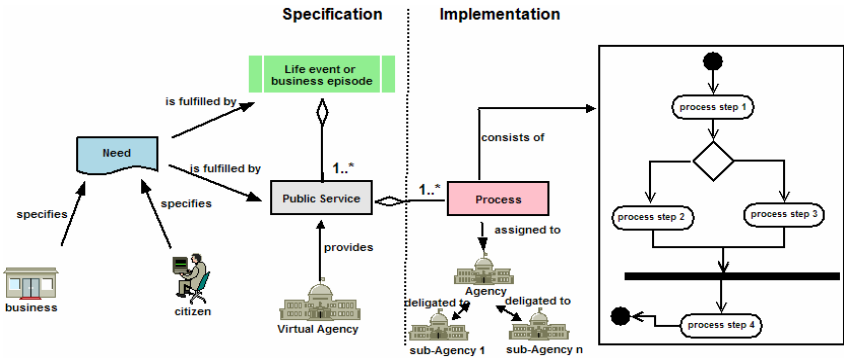

Figure 1: Workflow Infrastructure for VGOs - Conceptual Model

The workflow component GovWF connects the users and providers of services. It receives service/process requests, invokes processes at the agencies and coordinates cross-agency executions. Figure 2 depicts the sequence of interactions between a user, GovWF and agencies, taking place between requesting and receiving a service.

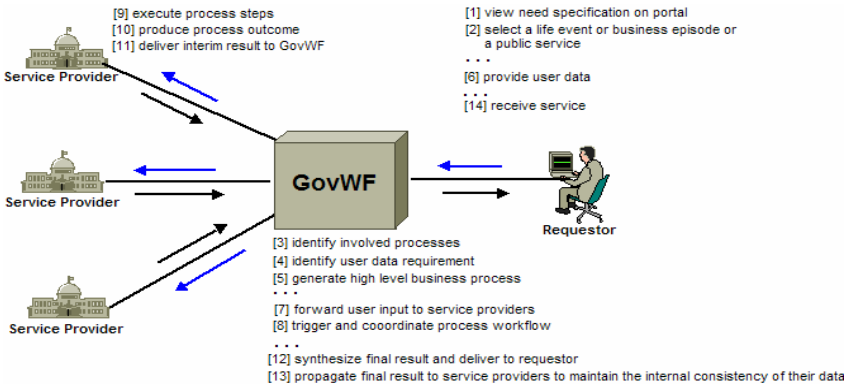

Figure 2: Workflow Infrastructure for VGOs - Usage Scenario

3.2 Formal Model

The aim of this section is to present a fragment of the formal model for GovWF, divided into Services, Agencies and Service Execution sub-sections.

3.2.1 Services

First, we introduce the abstract type to represent names in the model.

```
type
  Name
value
  this, super, default: Name
```

In order to describe both service requests formulated by agency users and the services offered by individual agencies, and allow matching of the requested and offered services, we introduce the abstract type Service, function `sat` to compare services (reflexive, anti-symmetric and transitive) and the weakest service for `sat`.

```
type
  Service
value
  weakest:
  Service
```

```
value
  sat: Service >< Service -> Bool
axiom all s, s', s'': Service :-
  (sat(s, s)) /\
  (sat(s, s') /\ sat(s', s) => s = s') /\
  (sat(s, s') /\ sat(s', s'') => sat(s, s''))
```

A process is defined through the type `Process` and two functions: `spec` returning the service delivered by the process and `impl` returning the steps executed by the process. There are two kinds of steps: invocation of another process by name and invocation of a service by executing any process satisfying the service according to `spec`. In both cases, a provider is specified and can be: `this` – the current agency, `super` – the supervising agency, or `sub` – one of sub-agencies given by name.

```
type
  Process
value
  spec: Process -> Service,
  impl: Process -> Steps
```

```
type
  Steps = Step-list,
  Step ==
    process(Provider, Name) |
    service(Provider, Service),
  Provider ==
    this | super | sub(Name)
```

For any process, we can calculate the execution context, which is the map from agency names to sets of process names executed within these agencies.

```
type
  Context =
    Name -m-> Name-set
value
  name: Provider -> Name
  name(p) is
    case p of
      this -> this,
      super -> super,
      sub(n) -> n
    end
```

```
value
  context: Step -> Context
  context(sp) is
    case sp of
      process(p,n) -> [name(p)+>{n}],
      service(p,s) -> [name(p)+>{}]
    end,
  context: Steps -> Context,
  context: Agency -> Context
```

3.2.2 Agencies

Two more types introduced are `Agency` to represent the behavior and structure of agencies, and `State` to represent the agencies' internal state. For a given agency we can obtain its: state (function `state`), set of currently executing process instances (`load`), set of all offered processes with unique names (`catalogue`), state transitions for all atomic processes (`operations`), and all sub-agencies of the agency with unique names assigned to each of them.

```
type
  State,
  Agency
```

```
value
  state: Agency -> State,
  load: Agency -> (Name -m-> Steps-list),
  catalogue:  Agency -> (Name -m-> Process),
  operations: Agency -> (Name -m-> State -> State),
  structure:  Agency -> (Name -m-> Agency)
```

Several axioms are defined to constrain this definition. Among them are:

1) Every internal process invoked by an agency is defined in its catalogue.

```
axiom[internal_processes_exist] all a: Agency :-
  this isin dom context(a) =>
  context(a)(this) <<= dom catalogue(a),
```

2) Every process in the catalogue with empty implementation represents the agency's internal operation and has the associated state transition.

```
axiom[operations_have_transitions] all a:Agency, n:Name :-
  (n isin dom operations(a)) is
  (n isin dom catalogue(a) /\ impl(catalogue(a)(n)) = <..>)
```

3) Every sub-agency of an agency whose processes or services are invoked through its processes is recorded in the agency's internal structure.

```
axiom[agencies_invoked_exist] all a: Agency :-
  dom context(a) \ {this, super} <<= dom structure(a)
```

4) Every sub-agency process invoked directly through an agency process exists in the catalogue of this sub-agency.

```
axiom[sub_processes_exist] all a: Agency, n: Name :-
  n isin dom context(a) /\ n ~isin {this, super} =>
  context(a)(n) <<= dom catalogue(structure(a)(n))
```

5) Every super-agency process invoked directly through a sub-agency process exists in the catalogue of the agency.

```
axiom[sup_processes_exist] all a: Agency, n: Name :-
  n isin dom structure(a) /\
  super isin dom context(structure(a)(n)) =>
  context(structure(a)(n))(super) <<= dom catalogue(a)
```

3.2.3. Service Execution

After defining agencies, we turn to operations. Invoking an agency process means to insert the instance of the process into the agency's load, provided the process exists.

```
value
  invokeProcess: Name >< Agency -~-> Agency
  invokeProcess(n, a) as a' post
    load(a')(n) = load(a)(n) ^ <.impl(catalogue(a)(n)).> …
    pre n isin dom catalogue(a)
```

Invoking a service means: (i) selecting any process in the agency's catalogue satisfying the specification of this service using `sat` and (ii) invoking this process.

```
value
  selectProcess: Service >< Agency -> Name
  selectProcess(s, a) as n post
    n isin dom catalogue(a) /\ sat(spec(catalogue(a)(n)), s) …
  invokeService: Service >< Agency -> Agency
  invokeService(s, a) is
    let n = selectProcess(s, a) in invokeProcess(n, a) end
```

After the process is loaded, its execution is carried out. For internal operations, the corresponding state transition is invoked and the process is removed from the load.

```
value
  execOperation: Name >< Nat >< Agency -~-> Agency
  execOperation(n, i, a) as a' post
    state(a') = operations(a)(n)(state(a)) /\
    load(a') = load(a) !! [ n +> remove(load(a)(n), i)] …
    pre … load(a)(n)(i) = <..>
```

For the remaining steps, the nature of the first step of the process is examined. If the step is provided by the current agency, the process/service is invoked as described above. Otherwise, the invocation is carried out within a given sub-agency.

```
value
  execStep: Name >< Nat >< Agency -~-> Agency
  execStep(n, i, a) is
    let s = hd load(a)(n)(i), a' = removeStep(n, i, a) in
      case s of
        process(=this, m)   -> invokeProcess(m, a'),
        process(=sub(n), m) -> execSubProcess(n, m, a'),
        service(=this, s)   -> invokeService(s, a')
        service(=sub(m), s) -> execSubService(m, s, a')
      end
    end pre … load(a)(n)(i) ~= <..>
```

3.3 Implementation

The implementation of GovWF consists of five major components as follows:

1) *Client* - receives requests from the portal and hands them over to the Filter component for processing. Client also delivers responses back to the portal.
2) *Filter* – this component is responsible for message transformation and security. It performs specific functions such as validation, logging, encryption, decryption, authentication and others, upon receiving service requests.
3) *Processor* - identifies required processes and generates process steps required for coordination. It also forwards requests to the corresponding service providers through so-called Service Agents, and synthesizes the final outcome. It also triggers the delivery of the final outcome and propagates the necessary side effects to the relevant agencies to ensure internal consistency of the VGO.
4) *Global Service Repository* - provides online data related to service offerings by government to its stakeholders and describes how the services will be invoked.
5) *Local Service Repository* - located at individual agencies, it comprises online data describing the services provided by each agency or service provider.

Service descriptions and data exchange from the portal through GovWF to the various agencies participating in the VGO will be implemented using XML. This is to enable interoperability among government agencies which may store information in proprietary data formats and systems. Governmental Markup Language (GovML) is a potential framework for implementing this functionality (Kavadias, 2003).

4. CASE STUDY

We demonstrate how GovWF supports the operations of a VGO providing services relevant to establishing a restaurant business (see Figure 3). The VGO comprises four agencies: Legal Affairs (`A1`), Municipal Services (`A2`), Public Works (`A3`) and Health (`A4`). Each agency provides one or two services and the VGO itself offers five services. Each service includes an identifier (e.g. `P1`), specification (e.g. `register`) and implementation (e.g. `register[A1]`). An implementation is a sequence of steps carried out to produce the service. For instance, `P2 P3` invokes sequentially the processes `P2` and `P3`, `register[A1]` invokes any process to satisfy the `register` service within `A1`, and `internal` is done internally.

Suppose the VGO receives a request to open a restaurant business. The request is received as a business episode `P1`, implemented by a sequence of two processes: `P2` (register business) and `P3` (issue restaurant license). `P2` is implemented by calling the `register` service of the agency `A1`, matching exactly the `register` service of `A1`. `A1` implements this service internally. Process `P3` of the VGO requests the `restaurant` service within the agency `A2`, which is matching the more general `catering` service of this agency. This service is implemented by requesting two sequential services `inspect` and `P5` from the VGO itself, identified by "`. .`". The former matches exactly the `inspect` service, which in turn requests the `inspect` service from the agency `A3`, while the latter is the `sanitation` service, which in

turn requests the `check` service from the agency `A4`. Both `A3` and `A4` perform the requested services internally. GovWF supports all interactions in this scenario.

VGO – Business Support Services			
id	spec	impl	description
P1	–	P2 P3	open restaurant
P2	register	register[A1]	register business
P3	issue	restaurant[A2]	issue restaurant license
P4	inspect	inspect[A3]	facility inspection
P5	sanitation	check[A4]	sanitation service

VGO – Sub-Agencies	
id	name
A1	Legal Affairs
A2	Municipal Affairs
A3	Public Works
A4	Health

Agency A1 – Legal Affairs			
id	spec	impl	desc
P1	register	internal	registration

Agency A2 – Municipal Services			
id	spec	impl	desc
P1	catering	inspect[..]P5[..]	catering

Agency A3 – Public Works			
id	spec	impl	desc
P1	inspect	internal	inspect

Agency A4 – Health Bureau			
id	spec	imp	desc
P1	check	internal	check

Figure 3: VGO Case Study – Opening a Restaurant Business

5. CONCLUSIONS

We described the concept of a Virtual Government Organization as a network of public agencies and private organizations involved in the delivery of government services. A VGO separates service specifications and implementations. To facilitate the dynamic matching of user needs against available services and the execution and coordination of cross-agency processes, we presented a workflow infrastructure to underpin VGOs - GovWF. The concepts and operations of GovWF were described informally and formally. A possible implementation was presented with a case study showing how a cross-agency request is handled within the GovWF-enabled VGO.

Following this work, we plan to detail the properties of GovWF and describe the protocols for coordination and monitoring. We are also working on building an implementation of GovWF within the context of an ongoing e-government project.

REFERENCES

1. Abbe Mowshowitz, Virtual Organization, Communications of ACM, Sept 1997, Vol. 40. No 9.
2. Herbert Kubicek, Martin Hagen, One-Stop-Government in Europe, http://www.e-gov.gr/local/ ism-egov/resources-egov/COST%20Project%20-%20One-Stop-Government%20in%20 Europe.pdf, 2000
3. Gregory Kavadias and Efthimios Tambouris, GovML: A Markup Language for Describing Public Services and Life Events, KMGov 2003, 106-115
4. Maciej Witczynski and Adam Pawlak, Virtual Organizations Enabling Net-based Engineering, [online], http://www.ecolleg.org/DISSEMINATION/VO-Dec2000-SUT.pps
5. Maria Legal, Gregoris Mentzas, Dimitris Gouscos and Panagiotis Georgiadis, CB-Business: Cross-Border Business Intermediation through Electronic Seamless Services, Rolland Traunmuller, Kluas Lenk (Eds.), Electronic Government 2002, LNCS 2456, pp. 338-343
6. Maria Wimmer and Johanna Krenner, An Integrated Online One-Stop Government Platform: The eGov Project, IDIMT-2001, Universitatsverlag Trauner, Linz, pp. 329-337, 2001.
7. Zhiyuan Fang, E-Government in Digital Era: Concept, Practice and Development, International Journal of the Computer, The Internet and Management, Vol. 10, No. 2, 2002, pp. 1-22