A Flexible, Event-Driven, Service-Oriented Architecture for Orchestrating Service Delivery

Sietse Overbeek, Bram Klievink, and Marijn Janssen, *Delft University of Technology*

This event-driven architecture uses decentralized intelligence to orchestrate services across a network of public agencies, without compromising the individual agencies' autonomy.

Government functions and roles are divided among many organizations, so creating processes and tasks to realize integrated service delivery is difficult. Our research is based on an immigration case study; an example illustrates the level of complexity involved. When coming to the Netherlands, an immigrant must apply for a number of permits by invoking services from several organizations. At the very least, the immigrant must request a temporary residence permit at the Dutch Immigration and Naturalization Service and a tax number or a tax exemption at the Inland Revenue Service. Immigrants often need additional permits from other public agencies—a permit to bring a spouse and children into the country; a permit to import a car or household goods; applications for a study grant, a loan, or daycare for the children; and so on. Once they've moved to the Netherlands, immigrants must register their new address in the citizens' registry, which is operated by the municipality in which the immigrant is living. There are many possible variations, and the exact course of services, processes, and tasks is hard to specify in advance. Nevertheless, several dependencies and constraining rules apply; for example, immigrants must have a temporary residence permit before they can apply for any other permit. Existing architectures don't support integrated delivery of services for such a highly complex and dynamic situation involving autonomous or semiautonomous organizations.

Nonetheless, combining and orchestrating the various services and tasks involved is high on governments' agendas, because individuals and businesses increasingly expect integrated and customized service delivery. Many past initiatives have focused on providing a detailed description of business processes and

1541-1672/09/$26.00 © 2009 IEEE
Published by the IEEE Computer Society

on defining interfaces, which often are "thick" with many information elements. Streamlining these interfaces requires standardization of the processes involved, because otherwise a system would have to analyze and predefine thousands of processes to enable cross-organizational service delivery, which simply isn't feasible. Therefore, many initiatives targeting integration focus on recurring questions rather than on incidental, nonstandard, or other requests that are difficult to predict. Furthermore, the introduction of new laws and regulations and changes to existing ones require continuous modification of processes and interfaces. This is an overwhelming task, requiring new analyses and often a complete redesign, which hampers modification. Organizations therefore need process execution that is flexible, that can easily adapt to changing circumstances, and that can create customized cross-organizational processes to accommodate complex service delivery.

Rather than looking to standardize and create interfaces, the architecture we discuss in this article relies on decentralized intelligence, which agencies use to process events. The architecture replaces thick interfaces with events, which trigger organizational activities. This creates the flexibility necessary to adapt to changing circumstances and makes it possible to generate new processes by a sequence of events. Nevertheless, the system still needs to share information for managing and orchestrating the dependencies among organizations. An ontology describes this information; its semantics create a database containing and capturing the necessary business information about the agencies. In this database, government organizations can register, modify, and remove their business information and other characteristics described in the ontology. The information stored in the ontology can be used to manage a cross-organizational process that matches specific requirements, thus enabling customization. After each step, the ontology makes it possible to decide on a possible next step, which increases adaptability compared to hard-coded, rigid business processes. The ontology captures the information needed to identify which services are needed where in the cross-organizational process, so that the various agencies can generate new events on the basis of the previous steps. The information can potentially aid in the analysis of dependencies or relationships among elements, and in this way it can improve cross-organizational processes.

Integrated Service Delivery

Governments often have highly fragmented structures, with many local and national agencies performing their own specific part of a more general task. Although government organizations often operate in silos, nowadays they must collaborate with other government agencies as well as private-sector partners. When we look at cross-organizational service processes such as our immigration example, it's clear that clients often must deal with various organizations, which creates a nested structure of service delivery, responsibilities, and operations, with various processes running in parallel and interacting with other organizations' processes as well as with the client. This involves a complex set of interactions between the various partners in a network. What's more, the organizations involved often have a relatively large degree of autonomy and may be reluctant to give up any of it, which makes coordinating these kinds of service networks challenging.

Standard service-oriented architecture (SOA) solutions for coordinating interorganizational processes fail to address the distributed nature of service networks, and often focus exclusively on coordinating standardized processes that can be fully specified in advance. In this article, we investigate the possibility of using events to coordinate demand-driven services across a network of organizations by developing an *event-driven, service-oriented architecture* (EDSOA). The architecture communicates events to allow flexible and loosely coupled interactions between autonomous organizations. Organizations can subscribe to certain types of events and can be informed when these events take place.

An *event* is typically a service request by an individual or business, but it can also be initiated by other organizations (for example, when an object's state changes), or it can take the form of an annual trigger. When an event occurs, organizations must determine whether and how to deal with it. If they respond, they must identify the activities and processes that are necessary to process the event. The distributed architecture integrates the activities of several public (and possibly private) organizations and provides the flexibility to deal with unanticipated questions. In other words, the architecture gives organizations a flexible way of supporting service delivery processes driven by the demands of individuals and businesses.

Event-Initiated Interactions

Legislation deals with the "what" question and doesn't address "how." Processes running across multiple organizations must typically deal with numerous laws and regulations originating from various sources (such as the European Union, ministries, and local governments); these are often created without other legislation in mind. Each organization executes and

implements these laws and regulations in its own way. It is nearly impossible to specify in advance the flow of the entire process through a network of service partners, because it's not always clear which steps such processes involve, at which point in time they will occur, or where they are enacted. Also, autonomous or semiautonomous organizations often don't want to depend too much on other organizations, so the interaction process requires a mechanism to coordinate the organizations' technological responses flexibly and in a way that matches the service network's fragmented structure. The complex set of interactions requires a very loose coupling between various parties, which means that we must expand the customary linear request-response interactions to facilitate events—for example, via a publish-subscribe mechanism.[1]

In such a mechanism, the basic building blocks are events, which can be defined as a state change resulting in a notable occurrence at a particular point in time. Events serve as a communication vehicle to inform or instruct: Individuals or businesses generate some events (by marrying, for example, or applying for a residence permit). Organizations involved in service delivery generate others (by completing a process, for example). Events can notify relevant parties of a change in information, but they don't contain the information itself.

An event's purpose is to coordinate a high-level business process by adding high-level, thin information flows and simple interactions on top of regular process execution. In this setup, there are two layers: The first contains the overall business service that coordinates the parts at several organizations by events and agreements. This way, the interactions among and between service partners and clients take place through events. The second layer includes processes within organizations that disseminate events and interact with other organizations at particular points in time or during the process. Generating events is the responsibility of the organizations themselves.

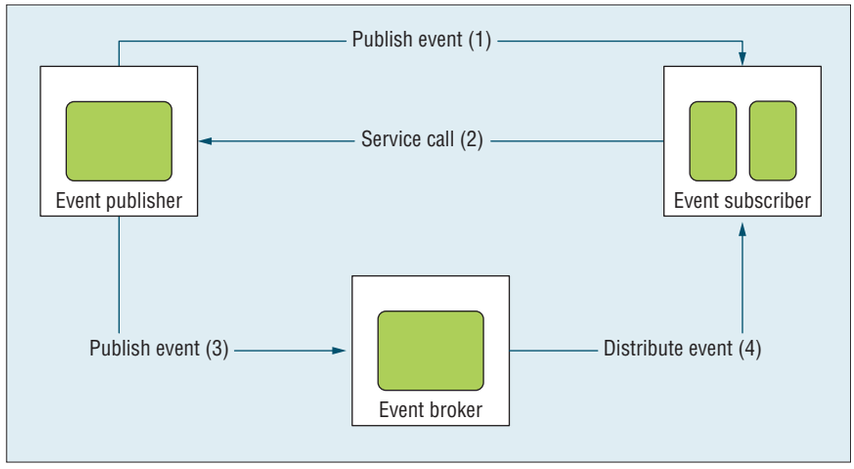

Figure 1. Event-initiated interactions, with and without a broker. The organization publishing the event either distributes the event among the organizations that have subscribed to the event type (1), or sends it to a broker (3), which distributes it (4). The event subscriber determines whether it needs to act on the event; if so, it initiates a service call to the publisher (2).

A simple example of an event is a change of address. It is nearly impossible to predict in advance which government organization will learn of such an event first, which is why governments require citizens to register a change of address at a specific agency. However, if any one government organization receives and verifies such an address change, that organization could publish that event to inform other government organizations. The event doesn't contain much more than a notification that a person with a certain social security number changed his or her address. One subscriber to this type of event would be the central administration, which, upon learning of the event, activates a process requesting the full address from the agency that initially published the event.

The combination of events, the rules to process them, and the services designed to handle the follow-up creates a flexible mechanism for orchestrating cross-agency service delivery. Because this is only a technological tool enabling interaction, it must be accompanied by contracts and service-level agreements (SLAs) to create the necessary coherence at other layers. Even though events are discretionary in nature—which is one of the main ways this setup respects the autonomy of the organizations involved—follow-up may be mandatory. For example, the government organization that receives and verifies an address change may at its discretion notify other government organizations of this change. However, the follow-up that is mandatory for such an organization is to provide the central administration with the verified address so that it can update the address in its records. Nevertheless, organizations can continue to use their own systems and architectures. On top of that, an event bus connects the organizations, and portals or applications fire and signal events. Organizations connect to the event bus, register relevant events, and then act on them. This can involve processes that directly interact with other organizations.

Figure 1 shows two forms of disseminating events: with and without

a broker. The event serves to notify partner organizations of a resource or client status change or to issue a request for a service or information. The organization publishing the event (most likely the organization that receives the service request from the individual or business) either distributes the event among the organizations that have subscribed to the event type (action 1 in Figure 1), or sends the event to an entity (such as a third-party organization) that keeps track of publishers and subscribers of certain event types (action 3). The broker then distributes the events to the subscribers (action 4). Organizations can also use a broker to authenticate the source and recipient. In both scenarios, the event subscriber determines whether it needs to act on the event; if so, it initiates a service call to the publisher (action 2). This call can, for example, contain the information requested by the event or serve as a request for further information. To return to the previous example, the subscriber determines whether the person who has changed his or her address is in its records and, if so, starts a service request to gather the new address information.

One of this setup's main advantages is that all the steps don't need to be known or specified in advance. Orchestrating in EDSOA doesn't specifically dictate to each individual organization what to do and how, it only distributes the events. How to respond to that event is left to the decentralized organizations. The very loosely coupled nature of event interactions requires the orchestration and monitoring of the process execution to ensure that at least the necessary steps are taken and that the process is finished in time. This requires the development of an ontology that captures the main elements necessary for managing and orchestrating cross-organizational processes at an event level. The decentralized organizations can use the information stored in the ontology to create and track customized, cross-organizational processes.

Framing Integrated Service Delivery

Ontologies are vital semantic resources for many application areas that involve processing by machine.[2] An ontology is a shared understanding of a certain domain, formally represented as logical theory in the form of a computer-based resource. By sharing an ontology, complex software applications can meaningfully communicate to exchange data and thus make such data transactions interoperate independently of their internal technologies.

E-government applications commonly use models and ontologies to build generic, universal representations and models.[3] In public service, the ontology aims to capture the basic elements of cross-organizational processes, thus framing integrated service delivery. Having a description of each organization's business information allows an organization to determine its potential next steps, which organizations it should communicate an event to, and whether to monitor the execution. Our ontology includes semantics related to the public domain, creating a starting point for realizing integrated service delivery. We've used an ontology to create a shared understanding of the public domain and identify essential concepts and relationships between concepts. We use the ontology's concepts to create a database in the architecture, to which organizations can add information about their services, resources, processes, and events. The system can then use this information to investigate which organization a new event is relevant to and to monitor the provision of services.

Object-role modeling (ORM) is a conceptual data-modeling technique that can aid conceptual modeling of databases as well as modeling ontologies. Figure 2 shows the ORM model of the ontological framework we derived from the immigration case study. We developed our ontology by conducting interviews, and we validated it using two discussion sessions as part of our case study. This case study involves highly complex, dynamic processes with many variations that are hard to model using static diagrams.

In an ORM model, ovals represent object types (which are the counterparts of classes), and boxes represent relationships between object types. These relationships are labeled as fact types. (For more details on ORM, see for example previous research by Erik Proper and Theo van der Weide.[4])

The ontology's eight central concepts are *role*, *actor*, *service*, *process*, *resource*, *organization*, *event*, and *EDSOA*. The ontology's role concept lets us denote a specification of an actor enactment. An actor is an organization resource that enacts a role during process performance or, at a more granular level, task performance. An employee of the Immigration and Naturalization Service playing the role of registrar is an example of such an actor in a public organization. The immigrant is another example of an actor in the public domain. An actor can perform an organizational process and uses services that are required for organizational processes. For example, a registrar who registers a new residence permit for an immigrant who comes to the Netherlands might perform a process called "grant residence permit." During this performance, the registrar uses a service to create a new residence permit registration. Figure 2 also shows that organizations have resources they use to perform processes. A human or computer-based actor is a

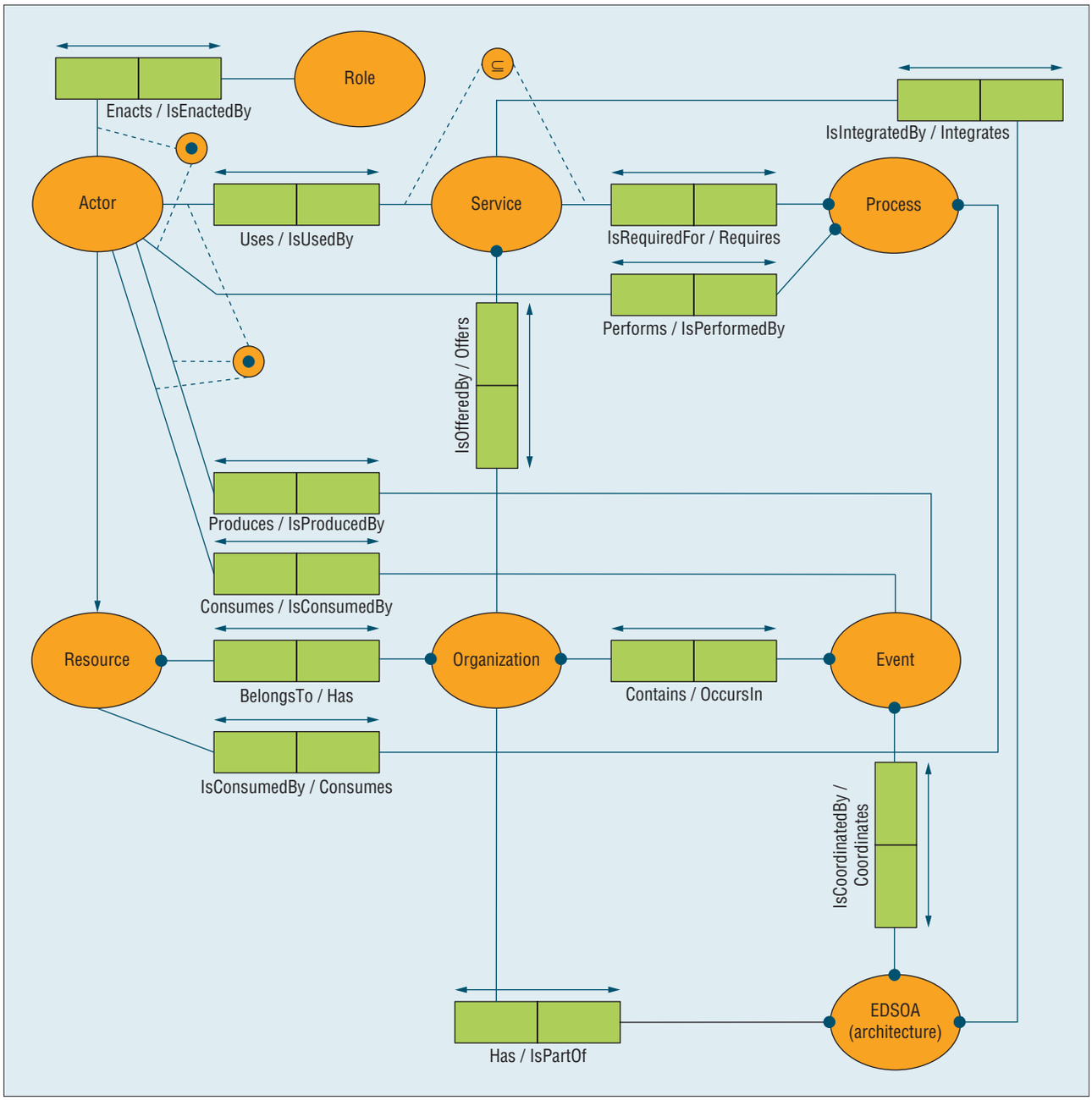

Figure 2. Ontology for creating and orchestrating cross-organizational processes. In this model, created using the object-role modeling (ORM) technique, ovals represent object types, and boxes represent relationships between object types. The ontology shows three different constraints: the mandatory, uniqueness, and subset constraints. The mandatory constraint, represented by a solid dot, indicates that each instance of an object type must play the role to which the mandatory constraint is related. The uniqueness constraint, represented by the double arrow, expresses that instances of object types may play a certain combination of roles at most once. We can use subset constraints, indicated by the subset symbol, to indicate that instances of an object type that play a certain role are also part of a set of instances that play another role.

specialization of the resource concept, because an actor belongs to an organization and is required for process performance.

Finally, Figure 2 also includes the notions of event and EDSOA. As previously mentioned, events occur during process fulfillment and are transmitted between integrated and interacting services and orchestrated by an EDSOA.[5] Getting a grip on organizational events is important in successfully

realizing integrated service delivery, because coordinating between services provides insight into how services depend on each other and which actors require which services. Implementing an EDSOA based on the ontological framework for integrated service delivery can enable integrated service delivery and orchestration of events in practice. Now that we've introduced the concepts of the integrated service delivery ontology, we can examine the ontological model.

Ontological Constraints

Constraints are important, because they show which relationships are possible. For example, customs cannot begin processing an application for importing a car before the immigrant has acquired a tax number. The ORM model of the ontological knowledge framework in Figure 2 shows three different constraints: *mandatory*, *uniqueness*, and *subset*.[6]

The mandatory constraint is sometimes called the *total role* constraint because each instance of an object type must play the role to which the total role constraint is related. For example, Figure 2 shows that every process is performed by an actor. This is shown by the solid dot on the *Process* object type, which is connected to the role *IsPerformedBy*. We can determine the population of elements in an ORM model—such as instances of object types, fact types (relationships between object types), and roles (one of the two parts that constitute a relation between object types)—by applying the population function, as discussed by Proper and van der Weide.[4] (The word *role* in this context—that is, as a building block of ORM models—should not be confused with the *Role* object type as part of Figure 2. The object type *Role* indicates a possible role that can be enacted by an actor participating in a cross-organizational process.) Using the population function makes it possible to reason about the constraints in an ORM model precisely. We model this function, referred to as Pop, as follows:

$$\mathtt{Pop}: \mathcal{OT} \rightarrow \wp(\Omega)$$

The set $\mathcal{OT}$ contains object types. (Roles and fact types are subsets of object types.) We can refer to the set Ω as the *universe of instances* or UoI. The UoI contains all possible instances of types found in an ORM model. The power set $\wp(\Omega)$ is the set of all subsets of the set Ω. We can now express mandatory or total role constraints found in the ORM model by applying the population function. For example, we can express the total role constraint connected to the role *OccursIn* as follows:

$$\mathtt{Pop}(\mathit{OccursIn}) = \mathtt{Pop}(\mathit{Event})$$

This means that every instance of the object type *Event* needs to play the role *OccursIn*. Suppose that a Chinese national requests a residence permit at the immigration service and then makes a similar request for his spouse. These actions generate two events at the immigration agency: first, a "request residence permit" event, then a "request permit to reunite family" event. These events play the role *OccursIn* because of the total role constraint connected to that role. This implies that the events should occur in an organization, in this case the immigration agency where these events arise.

It's now trivial to express each total role constraint that spans one role by using a population function like the one above. However, we can identify two total role constraints spanning various roles in the ORM model. First, note that if an actor uses a service, that actor also produces and consumes an event. This is expressed by the total role constraint spanning the roles *Uses*, *Produces*, and *Consumes*. Formally, we express this as

$$\mathtt{Pop}(\mathit{Actor}) = \mathtt{Pop}(\mathit{Uses}) \cup \mathtt{Pop}(\mathit{Produces}) \cup \mathtt{Pop}(\mathit{Consumes})$$

We can exemplify the formalization as follows: the immigrant that has requested a residence permit and a family reunion permit uses a residence permit service and a family reunion service at the immigration agency. When this is the case, it's mandatory that the immigrant produces and consumes events related to this service, such as the two aforementioned events, because of the total role constraint spanning the three roles.

Another complex total role constraint in the ORM model spans the roles *Enacts* and *Performs*. This is to make sure that an actor enacting a role also performs a process and vice versa.

We use uniqueness constraints in an ontology to express the fact that instances of object types may play a certain combination of roles at most once.[4] We can generalize this restriction as follows: If a certain combination of object type instances occurs in a set $\mathcal{RO}$ of roles, this combination should occur at least n and at most m times in this set. Note that $\mathcal{RO} \subseteq \mathcal{OT}$. We can express the uniqueness constraint using the following frequency function:

$$\mathtt{Frequency}: \mathcal{RO} \times \mathcal{N} \times \mathcal{N} \rightarrow \wp(\Omega)$$

Related to Figure 2, the expression $\mathtt{Frequency}^{(\{\mathit{Has},\mathit{IsPartOf}\}0,1)}$ shows that a combination of instances in the set of roles *Has* and *IsPartOf* should occur at least 0 and at most 1 time in this set. Instances of the object type *Organization* play the role *Has,* and instances of the object type *EDSOA* (*architecture*) play the role of *IsPartOf*. A combination of an architecture instance

playing the role of *IsPartOf* and an organization instance playing the role of *Has* cannot occur more than once. If the immigration agency uses an EDSOA, the service requests for the residence permit and the family reunion permit we mentioned earlier are referred to as "permit SOA." This architecture deals with the correct settlement of events related to permit requests. Because of the uniqueness constraint, this agency can't use two of exactly the same architecture implementations.

We can use subset constraints to indicate that instances of an object type that play a certain role are also part of a set of instances that play another role. In Figure 2, the model uses a subset constraint to indicate that each service that an actor uses should be required for some process. Formally, we describe this as follows:

$$\mathrm{Pop}(\mathit{IsUsedBy}) \subseteq \mathrm{Pop}(\mathit{IsRequiredFor})$$

The ontological constraints we've specified restrict what information can be registered, modified, or deleted by organizations that adapt the ontological model for integrated service delivery.

To increase usability for public organizations that wish to adopt the ontological framework in Figure 2, the ontology can be specified in multiple specification languages—XML, RDF, RDF-S, OWL, and so on.[2] These languages are specifically designed for computer-based applications that need to process the content of information instead of just presenting information to human actors. However, because it provides additional vocabulary along with a formal semantics, the Web Ontology Language (OWL) facilitates greater machine interpretability of Web content than languages such as XML, RDF, and RDF Schema (RDF-S).[2] To increase the successful adaptation and machine interpretability of our ontological framework, we turn to an OWL specification of the ontological framework that we have to this point visualized in ORM.

```
<owl:Class rdf:ID = "Resource" />
<owl:Class rdf:ID = "Actor">
  <rdfs:subClassOf rdf:resource = "#Resource" />
</owl:Class>
<owl:Class rdf:ID = "Role" />
<owl:Class rdf:ID = "Service" />
<owl:Class rdf:ID = "Process" />
<owl:Class rdf:ID = "Organization" />
<owl:Class rdf:ID = "Event" />
<owl:Class rdf:ID = "Architecture" />
<owl:ObjectProperty rdf:ID = "Enacts">
  <owl:inverseOf rdf:resource= "IsEnactedBy" />
  <rdfs:domain rdf:resource = "#Actor" />
  <rdfs:range rdf:resource = "#Role" />
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID = "Uses">
  <owl:inverseOf rdf:resource= "IsUsedBy" />
  <rdfs:domain rdf:resource = "#Actor" />
  <rdfs:range rdf:resource = "#Service" />
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID = "Produces">
  <owl:inverseOf rdf:resource= "IsProducedBy" />
  <rdfs:domain rdf:resource = "#Actor" />
  <rdfs:range rdf:resource = "#Event" />
</owl:ObjectProperty>
....
<owl:Restriction>
  <owl:onProperty rdf:resource="#Coordinates.Event" />
  <owl:minCardinality rdf:datatype=
   "&xsd;nonNegativeInteger">1</owl:minCardinality>
</owl:Restriction>
```

Figure 3. A partial Web Ontology Language (OWL) representation of the ontological framework for integrated public service delivery. This representation verbalizes the concepts, relationships, and constraints of the ORM model presented in Figure 2.

OWL Description of the Integrated Service Delivery Ontology

OWL, a language for publishing and sharing data using ontologies, represents the meanings of terms in vocabularies and the relationships between those terms in a way that is suitable for software processing. Figure 3 shows a partial OWL representation of the ontological framework for integrated public service delivery. This representation verbalizes the concepts, relationships, and constraints of the ORM model presented in Figure 2.

Using OWL to represent the ontological concepts and relationships between those concepts yields differences compared to modeling it in ORM. However, both languages intend to express the same meaning. For example, the ORM uniqueness constraint that is displayed as a double arrow and which spans *Enacts* and *IsEnactedBy* cannot be expressed in OWL, as it is implied by definition.[2] That is, the formalization of ObjectProperties in OWL

doesn't allow the same tuple to appear twice in the same set, such as Enacts = {<actor1,role1>,<actor1,role1>}. This is because the *domain* and *range* (or *codomain*) of *Enacts* are already specified in OWL. As Figure 3 shows, the domain of *Enacts* is the set *Actor*, and the range is the set *Role*. This immediately implies that *Enacts* is a total function, which means that every actor is uniquely associated with a role. The ORM model visualizes this through the total role constraint on the *Actor* object type and the uniqueness constraint on the *Enacts*/*IsEnactedBy* fact type. OWL expresses all the other uniqueness and mandatory constraints as cardinality restrictions. For instance, in OWL we express the mandatory constraint on *Coordinates* by the constraint owl:minCardinality. An owl:minCardinality constraint of one or more means that all instances of the class must have a value for the property.

Our ORM and OWL models illustrate different ways of characterizing the ontology. The contrast in formalizations and constructs of the two languages causes such differences. Which language is more suitable for specifying an ontology depends on the application scenario and the ontology's perspectives.[2] For example, ORM and EER (enhanced entity-relationship model) are suitable for database and XML-based application scenarios because of their extensive treatments of data set integrity. Languages based on description logic, such as OWL, seem more applicable to deductive and reasoning-based application scenarios, because they focus on the expressiveness and the decidability of axioms. The ORM model of the ontology for integrated service delivery and the OWL translation to enable the ontology's machine readability can together serve as the foundation for an EDSOA for integrated service delivery: the ORM translation is especially suitable as a database model for an ontology database, and the OWL translation is especially suitable to enhance the ontology's machine readiness.

EDSOA for Integrated Service Delivery

Figure 4 shows the elements from the ontological framework that constitute an EDSOA for integrated service delivery. At the top left of the figure, we see that clients request services from organizations by accessing an information portal on the Web. A service request produces an event. The event broker registers the event in the ontology database and communicates it to the relevant parties. To supply services to clients, organizations execute processes together. During the execution of such cross-organizational processes, actors that participate in process execution produce and consume events. The event broker tracks which actor performs which task as part of the process, and which events are being produced and consumed. This broker should, therefore, manage which available actors can consume events that are produced by other actors in the process. The process information that the event broker retrieves in fact comprises information that can populate the ORM model shown in Figure 2. Real-life instances and relationships between instances are extracted from the organizational process and can be stored in the ontology database. That is, the object types and fact types as part of the ontological model for integrated service delivery are instantiated by information extracted from real-life processes. For example, the system can store data related to services that organizations can offer as well as data about which actors are involved in which processes.

Furthermore, the ontology database contains data that can be used for process monitoring. Clients who request services from organizations produce events that indicate what they want, and these events are also stored in the database. An organization can read such client events from the database so as to correspond and supply a service that will resolve the client's need. The implementation of the ontology database is based on the conceptual ORM model of Figure 2. The ontology modeler translates the ontology to human-readable or machine-readable verbalizations tailored to the actor that wishes to retrieve information to realize some part of integrated service delivery. In this way, events can be viewed and monitored.

Evaluation: Immigration Case Study

We evaluated the ontology (Figure 2) and the accompanying architecture (Figure 4) using a scenario derived from the immigration case study. In the scenario, a Chinese immigrant with a spouse and children requests a (temporary) residence permit, applies for a study grant for his children, and wants to import a car. We used this scenario to materialize the ORM ontology model shown in Figure 2. First, we instantiated the process object type in Figure 2 as the temporary residence permit (TRP) process. We instantiated the ontology's organization object type as the Immigration and Naturalization Service, the Inland Revenue Service, the Student Loan Administration, and the Customs Authority. Events that occur in these organizations are in fact the results of the process steps as shown in Figure 4. For instance, the initial request results in an event that triggers all the organizations involved; events that occur at the Immigration

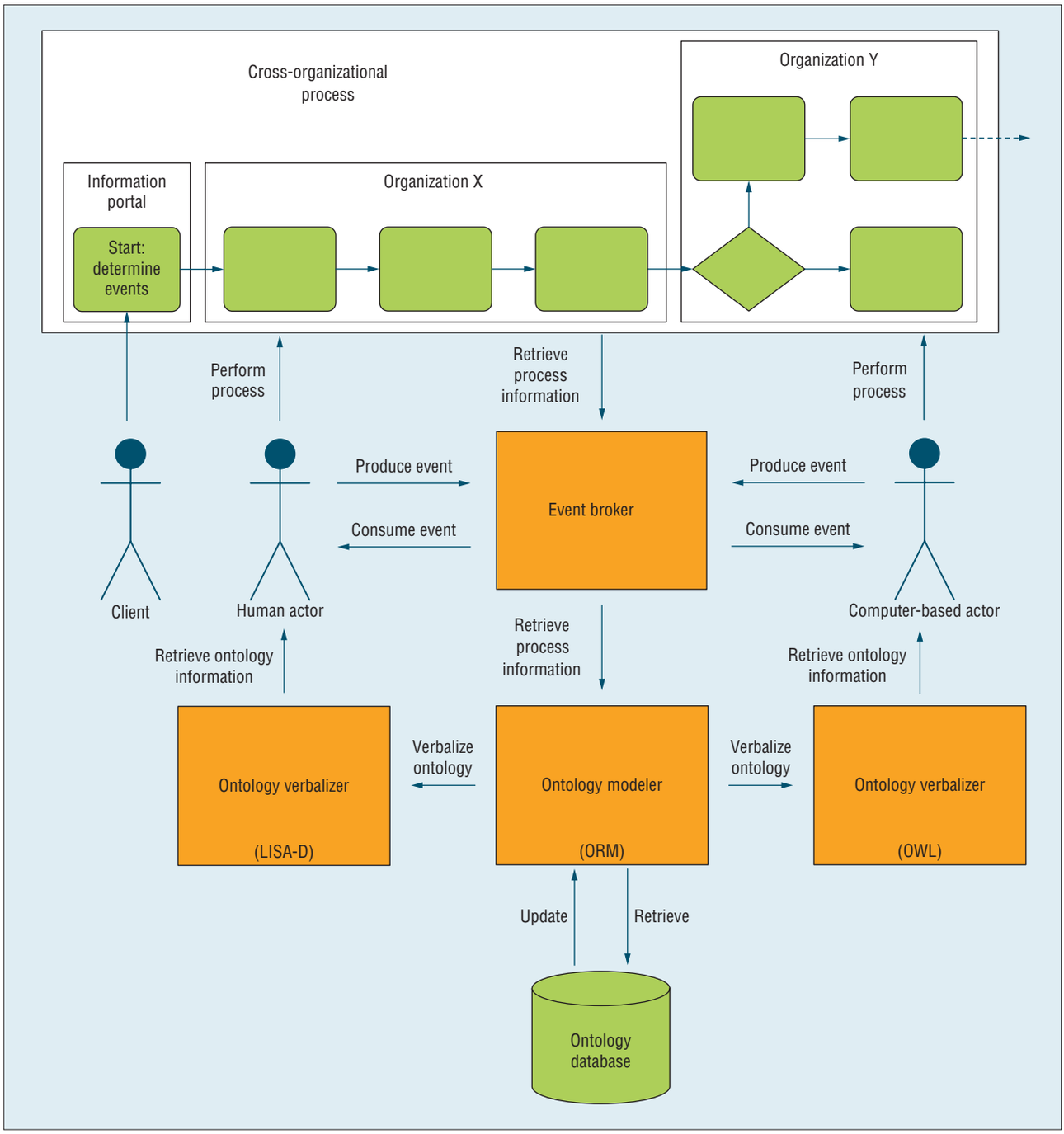

Figure 4. Overview of the architecture for integrated service delivery in relation to cross-organizational business processes. A client's service request produces events, which are registered in an ontology database and communicated to the relevant parties by an event broker. Real-life data is extracted from the organizational process and stored in a database that uses the ontological model for integrated service delivery as the underlying data model.

and Naturalization Service are "residence permit granted" and "acknowledge that TRP is ready." The service object type can be instantiated as the TRP provisioning service and labor contract service. Furthermore, several roles are enacted during the TRP process: for example, human resources employee at the immigrant's employer, immigrant, immigration officer, and permit advisor. The actual actors are the people

Related Work in Flexible Orchestration of Integrated Service Delivery

Generally speaking, two options exist for realizing integrated service delivery: centrally concentrating all intelligence in a single entity, and harnessing decentralized intelligence to accomplish integration. Often, top-down analyses are based on static processes, optimization of processes, and strict process control. Decentralized approaches are adaptive, with decentralized responsibilities of autonomous and networked parties. Sanjay Gosain, Arvind Malhotra, and Omar El Sawy propose advanced structuring and dynamic adjustment as main principles for flexibility:

> By appropriately structuring inter-organizational information flows and interconnected processes, enterprises can reduce the effort involved in adjusting to changing business environments. Through IT-supported learning and adaptation, enterprises can effectively and quickly reconfigure a set of inter-organizational processes that are appropriate for a changed business environment.[1]

Furthermore, many workflows focus on advanced structuring, which hinders easy modification and changes during workflow executions. This can complicate a workflow's implementation or a process-aware information system's configuration.[2] Adaptive decentralized workflows are more flexible, and organizations can modify or update them. As long as you don't violate ontological constraints, it doesn't matter how you compose an organizational workflow to integrate and deliver services. We can distinguish several approaches that contribute to achieving flexible workflows, and which we can modify during workflow execution:[3]

- *Dynamic model evolution.* Changes in dynamic workflows also require that changing process models are well managed. To support this, we can specify process model changes via a taxonomy of change modalities and a language for the unambiguous specification of procedural change.
- *Emergent process modeling.* A common approach to managing dynamic workflow tailoring is based on partially specified process models and depends on flexible workflow systems to refine and execute them at runtime.
- *Exception handling.* If workflows must be modified because processes change, workflow reliability can be maintained by using an exception-handling technique similar to exception handling in programming languages.
- *Flexibility by user selection.* Providing users a workflow system with some freedom, offering them multiple workflow execution paths, results in a more flexible workflow enactment.

Finally, we can use different ontologies on the Web to describe different services.[4] In this article, we use the conceptual modeling language ORM (object-role modeling) to describe the ontology for integrated service delivery. However, different modeling languages can describe the same ontology. When this is the case, interorganizational translations of the ontology are necessary. These translations, however, should also be organized in a decentralized way—that is, they should be made by actors who have the background knowledge and processing goals to perform these translations.[4]

References

1. S. Gosain, A. Malhotra, and O.A. El Sawy, "Coordinating for Flexibility in e-Business Supply Chains," *J. Management Information Systems,* vol. 21, no. 3, 2005, pp. 7–45.
2. M.T. Wynn et al., "Reduction Rules for YAWL Workflows with Cancellation Regions and OR-Joins," *Information and Software Technology,* vol. 51, no. 6, 2009, pp. 1010–1020.
3. W.M.P. van der Aalst et al., "Business Process Management: Where Business Processes and Web Services Meet," *Data & Knowledge Eng.,* vol. 61, no. 1, 2007, pp. 1–5.
4. M.H. Burstein, "Dynamic Invocation of Semantic Web Services that Use Unfamiliar Technologies," *IEEE Intelligent Systems,* vol. 19, no. 4, 2004, pp. 67–73.

or computer-based entities that enact such roles and can be identified by their organizational names. Finally, the EDSOA (architecture) object type in Figure 2 can be instantiated as an architecture that orchestrates and controls the cross-organizational TRP process. This architecture should then be adopted by organizations that participate in the TRP process.

Next, we evaluated the architecture using the four dimensions of flexibility;[7] our evaluation showed that there were improvements in the service delivery process on all these dimensions:

- *Robustness*. The architecture can endure variations and perturbations in external changes. The architecture for integrated service delivery shows that organizational workflows are coupled to offer integrated services in response to client needs. If organizational processes change, information about these changes can be stored in the ontology database, and the event broker can verify whether changes to processes satisfy the ontological constrains.
- *Modifiability*. Individuals and businesses can create, update, or delete instances of the objects that constitute the ontological model in the ontological database. For instance, an event "residence permit expired" can be submitted to the immigrant and stored in the Immigration and Naturalization Service database.
- *New capability*. As long as main concepts and relationships between concepts for integrated service delivery remain justified, the ontology is generic in the sense that it can support the processes in our case study (that is, to provide an integrated set of services to clients).
- *Partnering flexibility*. The architecture can accommodate new organizations if they add their organizational data to the ontology.

This provides the potential to include both public and private partners involved in the service delivery process.

To demonstrate the EDSOA's flexibility, consider a simple rule. As we've mentioned before, immigrants must register in the citizens' registry after arriving in the Netherlands (which in turn requires that they have a place to live). If this rule changed, the workflow for receiving a residence permit or registering in the citizens' registry would also be affected. Our ontology provides the basis for orchestrating events between services provided by public and private organizations that adapt the ontology. In our case, flexible reactions to changes in the (public) environment are based on emerging events and the settlement of those events through the delivery of suitable services. Overall, we generate this flexibility by creating an ontology and an EDSOA that all decentralized organizations can use to register their information. The type of information that must be registered is limited, and the decentralized organizations maintain it. At the same time, the system uses organizational information to create and monitor cross-organizational processes. This keeps the central overhead limited and fosters the advantages of decentralized execution.

Future research can take several directions. First, we haven't yet implemented support for version control in the architecture design. Thus, if changes are made to the ontological model, these changes apply to all processes. In practice, some processes might need to be executed on the basis of previous versions of the ontology, while other processes can take place on the basis of updated ontological information. Second, we could analyze the information captured in the ontology together with executed processes to identify possible process improvements. We could use the conceptual modeling language LISA-D[4] (language for information structure and access descriptions) to create such verbalizations. Third, we could use data-mining algorithms to generate information from the ontology and analyze where we can make improvements for integrated service delivery—for instance, we could determine whether client events are dealt with sufficiently by organizations that supply services and correspond to client events. At the organizational level, further research should be done on the mechanisms that must accompany the EDSOA when private parties are involved. Finally, we haven't yet investigated the reusability and generalization of this research. Future investigation can focus on further validation of the ontology through case studies in various domains and countries. ■

THE AUTHORS

Sietse Overbeek is a research associate in the Faculty of Technology, Policy, and Management at Delft University of Technology. His research interests include conceptual modeling of information systems, ontology modeling, and service-oriented architectures. Overbeek received his PhD in computer science from the Radboud University Nijmegen. He is a member of the ACM. Contact him at s.j.overbeek@tudelft.nl.

Bram Klievink is a PhD candidate in the Faculty of Technology, Policy, and Management at Delft University of Technology. His research interests include coordination mechanisms for public and private service networks. Klievink received his degree in political science from the Radboud University Nijmegen and a degree in business information systems from Avans University of Applied Sciences. Contact him at a.j.klievink@tudelft.nl.

Marijn Janssen is an associate professor in the Information and Communication Technology section of the Faculty of Technology, Policy, and Management at Delft University of Technology. His research interests include business engineering, information integration, agent-based and service-oriented architectures, and designing the coordination of networked public and private organizations. He is also the director of the interdisciplinary SEPAM Master program. Janssen received his PhD in information systems from the Delft University of Technology. Contact him at m.f.w.h.a.janssen@tudelft.nl.

Acknowledgments

AGILE (Advanced Governance of Information Services through Legal Engineering; www.jacquard.nl/?m=426) supported this work.

References

1. Q.Z. Sheng, B. Benatallah, and Z. Maamar, "User-Centric Services Provisioning in Wireless Environments," *Comm. ACM*, vol. 51, no. 11, 2008, pp. 130–135.
2. M. Jarrar and R. Meersman, "Ontology Engineering—The Dogma Approach," *Advances in Web Semantics I*, T. Dillon et al., eds., Springer, 2008, pp. 7–34.
3. V. Peristeras, K. Tarabanis, and S.K. Goudos, "Model-Driven eGovernment Interoperability: A Review of the State of the Art," *Computer Standards & Interfaces*, vol. 31, no. 4, 2008, pp. 613–628.
4. H.A. Proper and T.P. van der Weide, "A General Theory for Evolving Application Models," *IEEE Trans. Knowledge and Data Eng.*, vol. 7, no. 6, 1995, pp. 984–996.
5. S.-T. Yuan and M.-R. Lu, "A Value-Centric Event Driven Model and Architecture: A Case Study of Adaptive Complement of SOA for Distributed Care Service Delivery," *Expert Systems with Applications*, vol. 36, no. 2, 2009, pp. 3671–3694.
6. P. van Bommel, A.H.M. ter Hofstede, and T.P. van der Weide, "Semantics and Verification of Object-Role Models," *Information Systems*, vol. 16, no. 5, 1991, pp. 471–495.
7. C. Tan and S.K. Sia, "Managing Flexibility in Outsourcing," *J. Association for Information Systems*, vol. 7, no. 4, 2006, pp. 179–2006.