SOA in Variability-Intensive Environments: Pitfalls and Best Practices

Matthias Galster, University of Canterbury

Laurens Lapre, CGI

Paris Avgeriou, University of Groningen

// Different business processes cause variability in information system implementations. An architectural pattern based on best practices that also implements service-oriented architecture in Dutch municipalities provides a solution. //

WHEN THE DUTCH government approves new laws, all of its more than 400 municipalities must implement them. Owing to Dutch municipalities' autonomy, the business processes that implement these laws vary. Service-oriented architecture (SOA) projects in local Dutch e-government first began in 2004. Because SOA-based solutions link business processes to implementation, these solutions must take variability among the business processes into account. For example, one municipality might require steps that another municipality does not. In the case of the Dutch law for social support ("Wet Maatschappelijke Ondersteuning" or WMO law; www.rijksoverheid.nl/onderwerpen/wet-maatschappelijke-ondersteuning-wmo), one municipality might require a dedicated check from an approved healthcare practitioner to assess whether a citizen is eligible for a subsidized wheelchair, whereas another municipality might simply rely on the citizen's self-assessment. Variability among municipalities is too great to provide one solution for implementing a law for all municipalities, but developing and maintaining individual solutions for each municipality requires significant effort.

Here, we discuss insights from studying the introduction of SOA in variability-intensive environments and propose an architectural pattern to overcome issues in current practice. We chose e-government as an example of a variability-intensive environment for three reasons. First, variability

0740-7459/14/$31.00 © 2014 IEEE

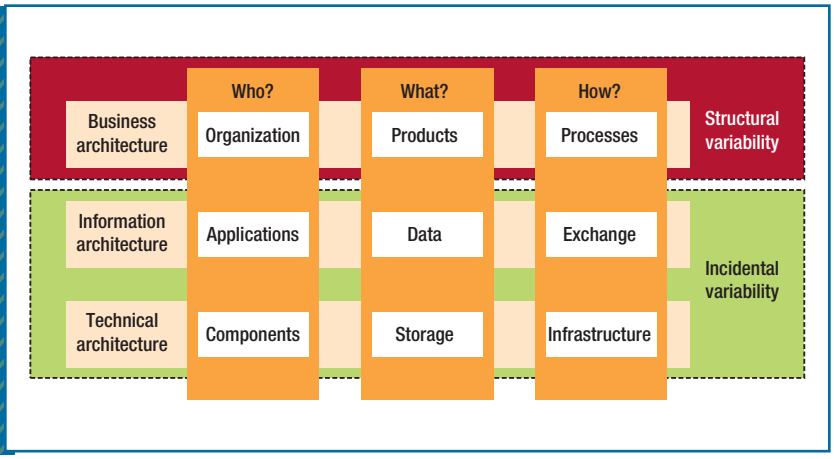

FIGURE 1. The Dutch Government Reference Architecture (NORA) matrix illustrates types of variability in the context of service-oriented architecture. Structural variability (in red) is often hidden and poorly understood and therefore our main concern. On the other hand, incidental variability (in green) is relatively easy to anticipate and thus not our concern.

between municipalities is a hidden but constant factor. It's often uncovered when introducing SOA in one municipality rather than anticipated when designing a SOA-based solution for all municipalities. The fact that we don't know the scope of variability up-front is also why we can't design product families for municipalities. Consequently, software solution providers must deal with variability when introducing a SOA-based solution in municipalities. Second, we found that e-government involves more variability than other domains (for example, retail). In e-government, many differences exist among municipalities that serve the same goals; although stores in a retail chain might differ in size, they aren't autonomous and typically follow processes defined for all branches. Finally, a strong drive exists toward SOA-based solutions in e-government,[2] making it a suitable domain to study such problems and solutions. Not many reports exist on SOA in variability-intensive environments in practice.

We were involved in implementing SOA-based solutions in Dutch municipalities for more than five years and had access to the stakeholders involved. We also interviewed representatives from software vendors and Dutch municipalities, studied documents, and consulted with experts. Our findings are applicable to projects from any variability-intensive domain that tries to implement standardized service-based solutions in multiple organizations. That said, extending our findings to other domains is subject to future investigation. We don't discuss general problems that occur when migrating to SOA (for example, see the work of Maryam Razavian and Patricia Lago[3]) although some pitfalls aren't specific to SOA-based solutions. The target audience of our findings are software solution providers who develop SOA-based solutions for multiple customer organizations.

Types of Variability

To classify types of variability in e-government and to focus our work, we used the SOA-based architecture framework for the Dutch Government Reference Architecture (NORA; www.e-overheid.nl/onderwerpen/architectuur-en-nora). NORA (see Figure 1) is widely used and provides architecture principles, views, roles, and so on. It has three layers:

- The *business architecture* describes organizations (municipalities) that deliver products and services (for example, building permits and fire inspections). Products and services are the result of business processes.
- The *information architecture* describes applications that organizations use (for example, information systems) to access and manipulate data. These data are shared and exchanged between organizations and the central government.
- The *technical architecture* describes the technical implementation of applications. Applications from the information layer consist of software and hardware components (for example, SOA services), data storage, and an information technology (IT) infrastructure.

Incidental variability (the green area in Figure 1) is relatively easy to anticipate. It occurs primarily in the information architecture and the technical architecture (for example, different databases in municipalities or IT infrastructures). Incidental variability is resolved by standardization (for example, laws, regulations from the EU, software reference architectures, standardized IT

infrastructures, standardized software services, or data standards). Thus, we aren't concerned about incidental variability.

Structural variability (the red area in Figure 1) is often hidden and only poorly understood up-front. It occurs in the business architecture and concerns product and process issues. Products are only partially centrally regulated and lack detailed descriptions. Consequently, organizations can modify them. Process variability refers to the variability in the business processes to provide these products. Structural variability will always exist and can't be resolved by standardization initiatives. Causes for structural variability include:

- organizational size—the larger the municipality, the more complex its processes, thus involving more employees or organizational units;
- economic status—wealthy municipalities have fewer budget constraints and provide more free services, which in turn affects their business processes;
- international appeal—municipalities that attract an international workforce have dedicated, detailed business processes for servicing an international audience; and
- demographic situation—municipalities with many senior citizens have well-defined processes provided for senior services, whereas municipalities with many young citizens focus on processes for young people.

This shows that we can't control structural variability. For example, we can't standardize the size of an organization. However, structural variability currently isn't well supported in practice.

Pitfalls

Structural variability has its pitfalls—this section highlights the ones that we observed in most organizations when dealing with structural variability. The introduction of service-based systems has its own general barriers,[4] but we focus here on pitfalls related to variability.

Pitfall 1 (P1): Data Diversity

Data needed to fulfill customer requests and to deliver a product to a customer vary. Consequently, data requirements among organizations differ.

Pitfall 2 (P2): Organizational Mapping

No concise mapping of processes to organizational units in different organizations exists. To implement one process, unit boundaries must be crossed, and these boundaries aren't the same in all organizations. Furthermore, processes for the same product or service offered by different organizations require different actors in different organizations. For example, some organizations have dedicated front-office staff for commercial customers, while others have one front office for both commercial and noncommercial customers.

Pitfall 3 (P3): Lack of Documentation

Customer organizations often don't have the resources—budget, staff, or expertise—to document their processes and variability. Although variability exists, organizations aren't aware of the extent of the variability or don't make it explicit. Consequently, the documentation of processes and variability among different organizations becomes a burden for software solution providers.

Pitfall 4 (P4): Quality Requirements

A standard process often doesn't meet quality requirements (for example, performance or privacy) of all the organizations that introduce it. If differences in a process's actual implementation are large, the degree to which quality requirements are achieved might differ.

Pitfall 5 (P5): External Parties

Different versions of a process implementation can require interactions with different external parties. The degree to which an application must be able to interact in an open-world setting isn't known in advance because external parties are different for each organization and for each implementation.

> Structural variability is often hidden and only poorly understood up-front.

Pitfall 6 (P6): Integration of Other Vendors

Coping with variability requires knowledge about the variability (for example, in what processes or products it occurs and what parts

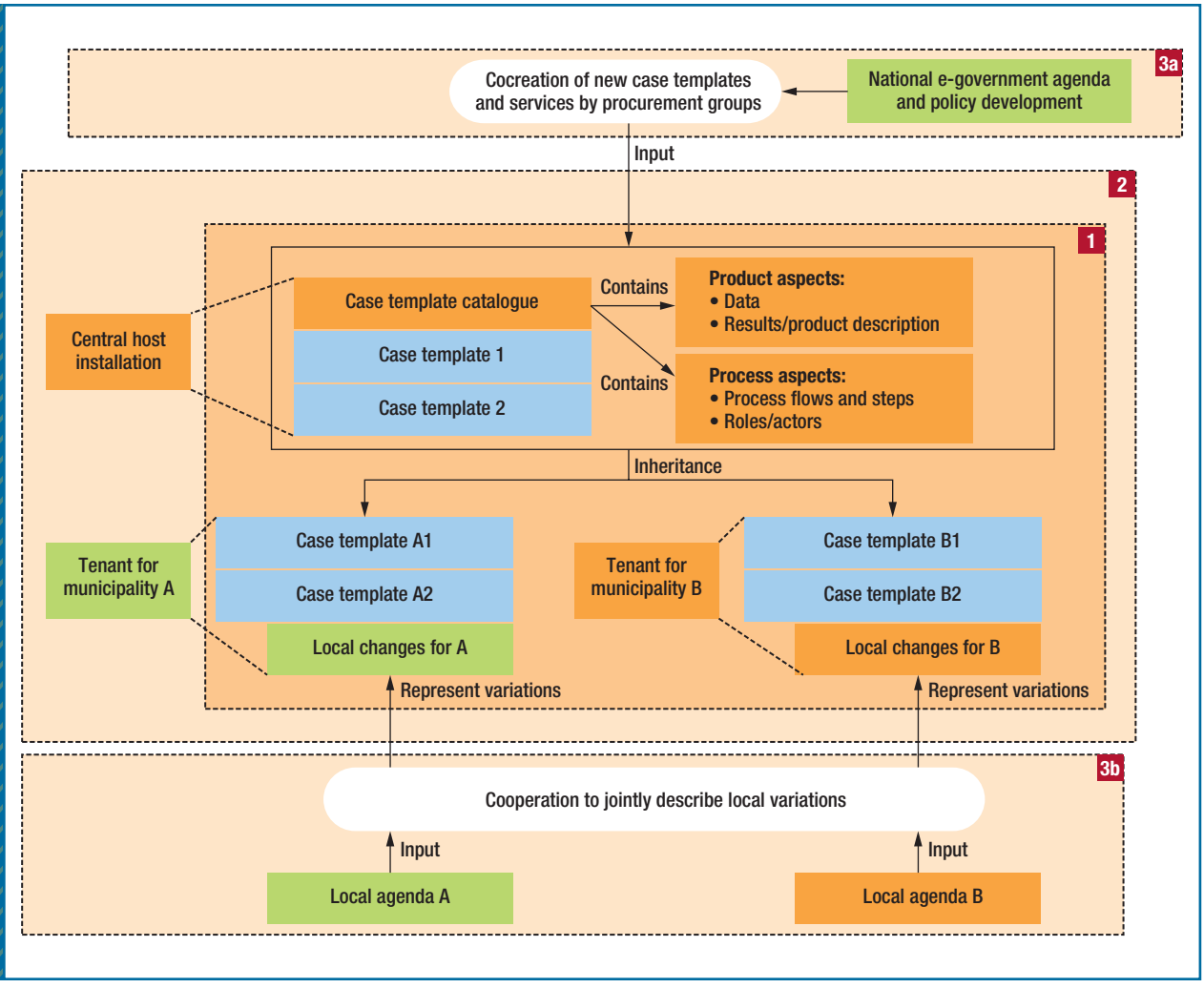

FIGURE 2. Architectural pattern to address product and process variability. The pattern integrates the three best practices case management, multitenancy, and cocreation and cooperation. Case management is used as part of multitenancy. Cocreation and cooperation are applied top-down and bottom-up to generate case templates used for case management. Multitenancy helps distribute these case templates.

of a system are affected by it). Such knowledge often resides with one software vendor and causes vendor lock-in because other software solution providers can't easily integrate their own solutions with those of the "knowledgeable" vendor.

Pitfall 7 (P7): Ripple Effects

As variability in processes causes ripple effects to other layers of service-based systems (the service layer, application layer, and so on), consistency among variability in the different layers is difficult to ensure.

Pitfall 8 (P8): Updates

In cases with a large number of organizations and lots of variability among them, updating systems becomes problematic. If each system is independently implemented to accommodate local specifics, developers must make global updates to each implementation of the system separately rather than making updates to a core system and propagating the updates to all instances of it.

Best Practices

Research has addressed variability in service-based systems mostly in the

context of product-line engineering.[5] However, to address the aforementioned pitfalls, we identified three best practices and organized them in an architectural pattern: *case management*, *multitenancy*, and *cocreation and cooperation*. Considered in isolation, these best practices aren't new. However, in our pattern, they're interrelated (see Figure 2). Furthermore, the best practices can be applied specifically to address variability. We call this solution a pattern because it isn't a one-time solution but a recurring one for the same problem that we've observed in e-government systems over the past five years.

Case Management

Case management (area 1 in Figure 2) serves the varying needs of individual organizations (municipalities). Here, a case can be a product provided to a customer (for example, a building permit issued for a citizen) or a process performed for the customer (for example, a WMO request performed for a citizen). To achieve standardization and local adaptation, case management focuses on process flows and products that are similar among organizations. Furthermore, we only document process and product aspects in case templates that are relevant to the customers involved. In contrast, process and product aspects relevant for the internal organization are neglected and left to local implementation details in municipalities. This partially addresses data diversity (P1) because a case template prescribes the data required to fulfill a process or to deliver a product. Furthermore, it helps achieve a first level of similarity for generic products and processes that are reused in case templates.

As Figure 2 shows, a case template catalog provides templates that include product aspects (data required for delivering a product and the results of doing so—for example, the details of a building permit) and process aspects, such as process flows and steps, and the actors and roles involved in that process. Based on these templates, vendors create local instances of products and processes (see "Inheritance" in Figure 2) that municipalities can then adapt to local process flows and product definitions. This addresses the pitfall related to organizational mapping (P2) because local instances can implement the organizational specifics (for example, actors and roles) of the municipalities, rather than assuming the same roles and actors in all organizations.

Case templates also define common properties of cases (products and processes) that different municipalities use. As Figure 2 shows, common properties include roles, actors, and so on, which ensures that, to a certain degree, the same types of external parties are involved in delivering a product or performing a process (P5). Furthermore, case templates reduce ripple effects across layers of a service-based system (P7) because case templates not only provide business process information but also implementation details. Finally, case templates facilitate updates because updates must only be applied to templates, not to individual implementations of processes (P8).

In Dutch e-government, national regulations are described and published as case templates in central case management repositories. These templates exist for frequently required services such as requesting a passport or building permit. Templates contain e-forms, data, and document templates as well as the roles and actors that handle citizen requests.

Multitenancy

Multitenancy (area 2 in Figure 2) creates a single central host installation at a software vendor site (online and cloud-based) that hosts all installations for municipalities ("tenants"). The main multitenancy provider is defined by the customer or municipality. The dotted lines between the boxes ("Central installation," "Tenant for municipality A," and "Tenant for municipality B") in area 2 and the elements in area 1 of Figure 2 (case template catalogs) indicate a refinement of multitenancy through case management. Thus, multitenancy isn't just a best practice but can also be used to distribute generic case templates. The central installation provides case templates, and each tenant uses instances of these templates, including local adaptations.

> Multitenancy isn't just a best practice but can also be used to distribute generic case templates.

As multitenancy is integrated with case management, one tenant

TABLE 1

Pitfalls and best practices.

Pitfall	Best practice		
	Case management	Multitenancy	Cocreation and cooperation
P1 – Data diversity	✓	-	✓
P2 – Organizational mapping	✓	-	-
P3 – Lack of documentation	-	-	✓
P4 – Quality requirements	-	✓	-
P5 – External parties	✓	-	✓
P6 – Integration of other vendors	-	-	✓
P7 – Ripple effects	✓	-	-
P8 – Updates	✓	✓	-

per customer is created, and each tenant inherits case templates. The tenant can make local changes to case templates while reusing large portions of the generic case templates. Because of the inheritance relation, most of the local changes can be kept, even when the central case template is updated to reflect new requirements (for example, new regulations or legislation). The reason is that changes are only made to the generic parts of the template but not instances. This addresses P8. Multitenancy also addresses the problem of quality requirements (P4): because multitenancy utilizes a cloud-based solution (at least in all the projects we were involved in), resources to accommodate differences in performance or privacy requirements can be added based on the needs of municipalities. Municipalities didn't care about technical issues for ensuring quality requirements because cloud infrastructures are built for extensions; they would simply sign service-level agreements with cloud providers.

An interesting side effect we noticed in Dutch e-government is that municipalities reconsider their own processes when a new generic case template becomes available. Municipalities then reassess the need for local variations and, if possible, adapt their physical processes to the new case template. Thus, generic case templates that can be adjusted to local needs help achieve more process uniformity across municipalities. This is a huge benefit of combining multitenancy and case management. Another side effect of using multitenancy is that having separate tenants allows changes for individual customers that don't affect the others.

Cocreation and Cooperation

Cocreation and cooperation (areas 3a and 3b in Figure 2) help generate case templates used in case management. Cocreation means that procurement groups create case templates and services. In the example of e-government, procurement groups consist of different organizations (for example, neighboring municipalities) and software solution providers. The development of case templates and services is driven by national e-government initiatives and policies. Cooperation means that several municipalities jointly document local variabilities. Each municipality considers its own local agenda when describing variability and contributes it to the total set of variabilities. Cooperation allows municipalities to pool resources to adapt to local needs. It also reduces data diversity as organizations use the same data in their processes (P1). As with case management, potential external parties can be unified to reduce the range of differences (P5). Involving municipalities in creating templates and services reduces vendor lock-in (P6). As a member of a procurement group, a municipality subscribes to the case management environment and gets a tenant and several case templates. Consequently, individual organizations need not define their own variability or implement an individual organization-specific solution (P3).

In the example of Dutch e-government, several procurement groups

exist (for example, GovUnited and Dimpact) to facilitate cocreation and cooperation. Competition among these groups drives innovation. Procurement groups also give small municipalities more power to negotiate with software vendors. The pressure from procurement groups in combination with knowledge sharing between municipalities through cooperation for both central case template development and documenting local variations is a key benefit of the presented architecture pattern.

Putting It All Together

Figure 2 shows how all the best practices form our proposed SOA-based architectural pattern for variability-intensive environments. Although this architectural pattern might address some incidental variabilities, we don't discuss that here. By combining the three best practices in one architectural pattern, we also create an ecosystem approach that lets software services from various vendors coexist and thus reduces vendor lock-in. In Table 1, we show which best practices address specific pitfalls. The best practices in Table 1 don't fully address each pitfall but do contribute to addressing them.

Implementation of the Pattern

We've applied our architectural pattern to approximately 20 municipalities in the Netherlands over the past five years. Sizes ranges from small (about 20,000 citizens) to large municipalities (up to 750,000 citizens). Common characteristics of the projects included the following:

- They're variability-intensive.
- They have a broad range of users (typically municipality staff and citizens).
- They're large-scale and nontrivial (for example, many components, integration with existing infrastructures and IT systems).
- The developed systems are enterprise systems (CRM, CMS, data warehousing, and so on) rather than consumer software.
- They aren't part of a product line.

Implementing information systems that follow the proposed pattern can take up to six months, depending on the organization's size. The customer organization usually involves around 20 to 50 people on the project, whereas the software solution provider usually involves two to five people. Depending on municipality size, 150 to 500 applications must be integrated. Cooperation and cocreation have primarily happened for processes related to citizen services, social security, and healthcare. For example, 40 standard templates currently cover 90 percent of the needs of the WMO law.

Experienced benefits (beyond the benefit of using SOA and addressing the pitfalls) include a higher software degree of reuse of old and new modules and cost reduction. Variability is documented and implemented not several times but only once, in a collaborative manner. Adapting generic processes is cheaper than keeping local instances of processes. It also requires less IT knowledge in the municipalities to implement changes or updates. Furthermore, we noticed a

ABOUT THE AUTHORS

MATTHIAS GALSTER is a lecturer in software engineering at the University of Canterbury. Previously, he worked as a senior researcher at the University of Groningen. His interests include requirements engineering, and software architecture. Contact him at mgalster@ieee.org.

LAURENS LAPRE has been active in the public sector since 1990 as a business consultant and enterprise architect at CGI. Originally a cognitive psychologist, he also did research on e-government at the TU Delft. Contact him at laurens.lapre@cgi.com.

PARIS AVGERIOU is professor of software engineering at the University of Groningen and a senior member of IEEE. He has published more than 110 peer-reviewed articles in international journals, conference proceedings, and books. His research interests lie in the area of software architecture, with strong emphasis on architecture modeling, knowledge, evolution, patterns, and link to requirements. Contact him at paris@cs.rug.nl.

reduction of vendor lock-in as vendors were "pressured" into using open instead of proprietary standards. Interestingly, the pattern also accelerates the adoption of e-government services by decentralized municipalities with limited IT resources.

Practical Problems and Liabilities

We observed several practical problems when implementing the pattern. On one hand, we faced general barriers that hold for any SOA implementation. Introducing the pattern in organizations with dominant back-office structures is difficult because these structures must adapt to accommodate the pattern. A strong back-office structure might hamper changes in existing processes due to political reasons. Furthermore, it might lead to cultural changes and require special training for staff. In addition, many organizations have legacy systems that, according to the architectural pattern, must communicate with other systems in the organization.

On the other hand, there are problems that come from the application of the proposed pattern. First, the pattern isn't applicable in situations where individual organizations are highly competitive and object to cocreation and cooperation. Second, the pattern might not be applicable if too much variability exists among different organizations. For example, when banks merge, many differences in processes exist, even to the extent that they have completely different IT landscapes.

These pitfalls and best practices apply beyond local e-government and are relevant for any software vendor that works with standardized business processes in SOA environments with a large number of customers. Likewise, the types of variability (incidental and structural) are generalizable to other domains. For example, multinational organizations have similar characteristics to national e-government because they're typical "head office organizations" and must often balance standardization with local variability: in multinational organizations, branches in different countries serve the same purpose but are subject to different cultures, regulations, laws, staff structure, and so forth. Our future research efforts are concerned with two issues: the integration of our pattern into processes for the design of reference architectures and the design of variability-intensive service-oriented reference architectures for e-government.

Acknowledgments

This work is supported by NWO SaS-LeG, contract number 638.000.000.07N07 and CGI. We thank Eljto Poort, Uwe van Heesch, and the reviewers for their valuable input.

References

1. M. Turner, D. Budgen, and P. Brereton, "Turning Software into a Service," *Computer*, vol. 36, no. 10, 2003, pp. 38–44.
2. A. Bouguettaya et al., "Service-Centric Framework for a Digital Government Application," *IEEE Trans. Services Computing*, vol. 4, no. 1, 2011, pp. 3–16.
3. M. Razavian and P. Lago, "A Frame of Reference for SOA Migration," *Proc. 3rd European Conf. towards a Service-Based Internet* (ServiceWave), Springer, 2010, pp. 150–162.
4. M. Razavian and P. Lago, "A Survey of SOA Migration in Industry," *Proc. 9th Int'l Conf. Service-Oriented Computing* (ICSOC), Springer, 2011, pp. 618–626.
5. J. Lee, and G. Kotonya, "Combining Service-Orientation with Product Line Engineering," *IEEE Software*, vol. 27, no. 3, 2010, pp. 35–41.