

# Scaling Business Process Platforms:

## Identifying and Meeting the Challenges

Business process platforms can help simplify the process of developing new software by enabling the creation of services, processes, and applications from existing reusable components. This article discusses the key challenges of scaling these platforms and outlines how elements such as metadata and software product lines can help meet these challenges.

BY DAVID S. FRANKEL

### Related Categories:

Application development

Business applications

Enterprise resource  
planning (ERP)

SAP

Visit [www.dell.com/powersolutions](http://www.dell.com/powersolutions)  
for the complete category index.

**B**usiness process platforms can help provide increased flexibility in business software systems by enabling developers to build new services, processes, and applications from existing reusable components. These platforms consist of two basic elements: a *technical software platform* and an *application platform*.

The technical software platform is a set of packaged technical capabilities designed to simplify the software development process; it includes database and network management systems as well as middleware that manages transactions, componentization, security, persistence, data transformation, Web services, and so on. The steady rise in the abstraction level of technical software platforms has enabled a similar rise in the abstraction level of programming languages and model-driven development tools.

The second part, the application platform, sits on top of the technical software platform and consists of frameworks of reusable, executable business-oriented services (such as a post-to-ledger service) and executable business processes composed of multiple services (such as an order-entry process). Application platforms also enable

a rise in the abstraction level of model-driven tools that combine simple services into complex services, processes, and applications.

### Identifying the principal challenges of scaling business process platforms

Several challenges can arise when scaling business process platforms, including three specific challenges related to the application platform: finding the appropriate components, building the appropriate components, and managing configuration.

#### Finding the appropriate components

Over time, business process platforms can offer increasingly large portfolios of reusable services and large numbers of processes composed from those services. Such an array of components can present several basic questions when building new services, processes, and applications from these components—for example, how can those who assemble solutions from components identify suitable components to reuse? And once they have identified

candidate components, how can they evaluate these candidates and determine whether they are compatible with one another?

### Building the appropriate components

Building software from reusable components has often not worked as well in practice as envisioned. Anticipating the needs of those using these components can be difficult, particularly for large platforms intended to support multiple business needs, such as enterprise resource planning, customer relationship management, and supplier relationship management.

### Managing configuration

Creating applications from reusable components does not solve all the traditional challenges of a software life cycle—in fact, some of these challenges can be exacerbated if not handled properly. Configuration management—including design-time configuration, deployment-time configuration, and version management—can be particularly complicated.

**Design-time and deployment-time configuration.** Each component may have a set of properties that the application modeler or developer sets during application design, such as a property of an accounts payable component that indicates whether to use cash or accrual accounting. A component might also have properties set during application deployment, such as a property of an accounts payable service that chooses a relational database source that the component uses for persistence, or a technically oriented property that determines the service communication protocol. Because each component can have its own set of configuration parameters, it can be daunting to debug errors that arise from subtle incompatibilities among multiple components' configuration properties.

**Version management.** Each application component has its own life cycle and version history, which can complicate software management. Installation procedures must deploy not only the correct application version at particular sites, but also the correct versions of all application components.

### Creating a metadata-rich environment for business process platforms

To help meet the challenges of scaling business platforms and to simplify the process of finding and managing large numbers of components, business process platforms must provide a metadata-rich environment. Metadata provides machine-readable information about the platform components. This section outlines the foundational types of metadata required by business process platforms—semantically rich service contracts, quality-of-service constraints, configuration invariants, and version information—and briefly discusses the goal of integrated metadata management.

### Semantically rich service contracts

A service contract consists of four fundamental elements:

- **Signature:** This defines the types of information that the service requires as input and produces as output.
- **Preconditions:** These are constraints that must be satisfied for the service to execute in a well-defined manner. A precondition of a money transfer service, for example, might be that the source and destination accounts must be owned by the same customer. Some money transfer services may not have this restriction, but for those that do, it is an important part of the service contract.
- **Postconditions:** These are constraints that must be satisfied when the service has completed execution; in essence, they describe the effects that result from that execution. A postcondition of a money transfer service is typically that the balance of the source account is decremented by the amount of the transfer, and the balance of the destination account is incremented by the same amount.
- **Information invariants:** These are constraints that apply to the information structures the service uses as its input and output parameters. An invariant for an ordinary checking account might require that the account balance never fall below zero (invalidating attempts to withdraw an amount greater than the balance), while an invariant for a checking account with overdraft protection might allow a balance of \$1,000 below zero.

Signatures have typically been the only machine-readable part of service contracts. For example, the Web Services Description Language (WSDL) allows developers to define an operation's signature. Supplementing the contract with preconditions, postconditions, and information invariants is central to the Design by Contract approach, which uses such constraints to define a contract's semantics (meaning).<sup>1</sup> The semantics of a service contract can often be obscured in code, without even being captured informally in English or some other human language. Expressing these semantics using formal constraint languages makes them machine-readable. In a metadata-rich environment, a full machine-readable contract—including both the signature and the semantics—is an integral part of a component's manifest.

The Unified Modeling Language (UML) includes the Object Constraint Language (OCL), which supports writing machine-readable preconditions, postconditions, and information invariants. The United Nations Centre for Trade Facilitation and Electronic Business (UN/CEFACT) Modeling Methodology is an example of a methodology that uses UML in this way.<sup>2</sup>

<sup>1</sup> For more information on Design by Contract, see *Object-Oriented Software Construction*, by Bertrand Meyer, 2nd ed. (Prentice Hall, 1997).

<sup>2</sup> For more information about the UN/CEFACT Modeling Methodology, visit [www.unece.org/cefact/umm/umm\\_index.htm](http://www.unece.org/cefact/umm/umm_index.htm).

There are other avenues besides UML, including various initiatives grouped under the name *Semantic Web Services*, which aim to use constraint languages in conjunction with WSDL and the Semantic Web languages RDF (Resource Description Framework) and OWL (Web Ontology Language). Emerging technologies can also represent machine-readable constraints in a human language, such as carefully structured English, so that nontechnical business analysts can read and write them.<sup>3</sup>

In addition, certain approaches to structuring business information can help reduce (but not eliminate) the need to express information invariants in constraint languages. The UN/CEFACT Core Components Technical Specification has an approach to organizing the definitions of *global data types* such that the very structure of a data element definition provides a map of the element's semantics.<sup>4</sup> This specification uses the joint International Organization for Standardization and International Electrotechnical Commission (ISO/IEC) 11179-5 approach to classifying data elements based on semantic structure—an approach that is also widely used in the Semantic Web community.<sup>5</sup> These semantic maps provide powerful metadata that can help enrich the service contract's semantic content when the data elements appear as service parameters. The industry is just beginning to learn how to exploit this type of metadata.

Including service contracts in the business process platform's metadata can provide several key benefits that can help address scalability problems:

- **Precise contract:** A precise contract specification with unambiguous, mathematically specified constraints can help simplify efforts to understand whether a component can satisfy specific requirements. Precision is particularly important when components must interact across organizational boundaries and human language barriers.
- **Constraint enforcement:** With some limitations, developers can use tools to enforce constraints. For example, they could generate code that checks whether a precondition is satisfied before invoking a service.
- **Collision detection:** Knowledge representation tools can detect some types of contract conflicts, such as mutually contradictory constraints—for example, a constraint on one component could require a customer's age to be 18 or under, while another requires it to be 21 or over. Such tools require that the languages used to define information structures be grounded in mathematical formalisms, as is the case with the

Semantic Web's RDF and OWL languages. Although a knowledge representation tool cannot detect certain types of collisions with certainty,<sup>6</sup> in such cases it might be able to raise a warning flag, allow the human to decide how to proceed, and store that decision. The next time the same warning occurs, the tool can display previous decisions and eventually could suggest a default decision.

- **Semi-automated service composition:** Emerging tools can partially automate the identification of candidate services that may satisfy a requirement. The searcher expresses the requirement in a machine-readable fashion, and the tool scans the available services to match the requirement against the service contracts. These tools also require formally grounded languages.

### Quality-of-service constraints

The service contracts discussed in the preceding section specify the functional behavior of a service; they do not specify nonfunctional, quality-of-service constraints. Yet quality-of-service constraints are important metadata too. A suitable service not only implements the necessary functional behavior, but also satisfies quality-of-service requirements.

Separating functional contract aspects from nonfunctional quality-of-service aspects can be useful. For example, standards bodies might want to codify the functional behavior contract for a common service, such as a post-to-ledger service, while leaving quality-of-service constraint specifications to negotiations among the service providers and clients. Implementers who provide the service can advertise the quality of service that the implementation offers.

### Configuration invariants

Configuration invariants are a special type of information invariant that specify constraints on the values of a component's design-time or deployment-time configuration parameters. The value of one configuration parameter may constrain the values of others. Tools can enforce these kinds of constraints, with some limitations, and detect collisions resulting from specific component combinations; again, these collisions cannot always be detected with certainty.

### Version information

A substantial amount of metadata can help developers effectively manage application and component versions, including helping anticipate the potential ramifications of a component change by tracking which applications use which components, down to the

<sup>3</sup> For example, see "Semantics of Business Vocabulary and Business Rules," by the Object Management Group, August 22, 2005, [www.omg.org/docs/bei/05-08-01.pdf](http://www.omg.org/docs/bei/05-08-01.pdf).

<sup>4</sup> For more information about this specification, see "Core Components Technical Specification – Part 8 of the ebXML Framework," by the United Nations Centre for Trade Facilitation and Electronic Business, November 15, 2003, [www.unecce.org/cefact/ebxml/CCTS\\_V2-01\\_Final.pdf](http://www.unecce.org/cefact/ebxml/CCTS_V2-01_Final.pdf).

<sup>5</sup> The ISO/IEC 11179-5 standard is available from the International Organization for Standardization Web site at [www.iso.org/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=35347](http://www.iso.org/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=35347).

<sup>6</sup> Although knowledge representation tools can typically detect with certainty collisions among the relatively simple constraints that make up description logics, certainty is generally unattainable for collisions among constraints using first-order logic.

version level. Such procedures can also help improve quality assurance processes.

**Integrated metadata management**

Using a wide variety of metadata can introduce its own challenges. Managing each type of metadata with different tools using different mechanisms can isolate metadata within separate silos. Integrated metadata management technologies such as the MetaObject Facility (MOF) specification, which is one of the core Model Driven Architecture standards, and the Eclipse Modeling Framework (EMF), essentially a type of MOF that underpins the Eclipse metadata management facilities, can help enterprises streamline metadata management.

**Building components with software product lines**

The Carnegie Mellon Software Engineering Institute (SEI), which produced the Capability Maturity Model (CMM) for Software, has defined an approach to organizing components for reuse called software product lines (SPLs). SPLs can help address one of the major problems with component-based development: scope. Constraining the scope to which a framework of components applies can help ease the task of creating truly reusable components.

The SEI defines an SPL as “a set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way.”<sup>7</sup> For example, an SPL could be a set of products that manage risk for portfolios of tradable financial derivatives or a set of products that provide role-based access security. A fairly narrow focus is typical of an SPL, because ensuring reusability is easier for a restricted, well-defined scope than for a broad, loosely defined scope.

The SPL approach divides software development into two distinct but related processes: core asset development and product

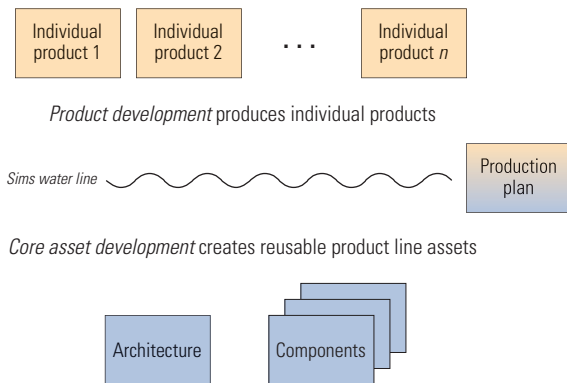


Figure 1. Software product line approach to development

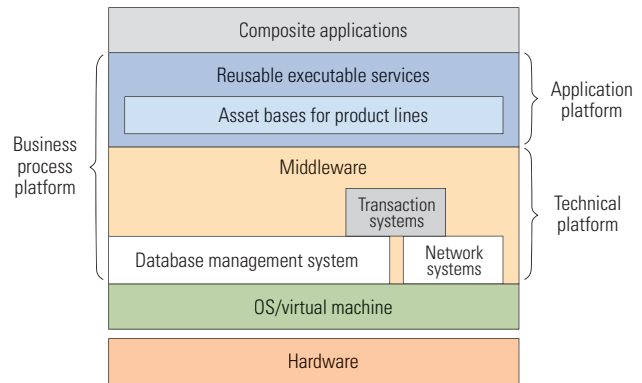


Figure 2. Business process platform with distinct but integrated product lines

development. Core asset development produces a framework of reusable assets for a product line and defines the framework architecture. Product development uses this framework to produce individual products. A production plan provides instructions for using the framework in accordance with the architecture to produce products. Figure 1 illustrates this approach, including how the two processes are divided by a *Sims water line*—an analogy developed by Oliver Sims to describe the separation, from the developer’s viewpoint, between aspects of a system that appear “above the surface” and aspects that the infrastructure handles “below the surface.”

Organizing a business process platform as a set of distinct but integrated product lines, each with a well-defined scope, can help simplify building reusable service and process components. Figure 2 illustrates such a platform.

**Enhancing software development with scalable business process platforms**

Business process platforms can help simplify software development by making it possible to build new services, processes, and applications from existing reusable components. Scaling such platforms presents numerous challenges. Addressing these challenges now can enable business process platforms to provide business value throughout their continuing evolution.

**David S. Frankel** has been in the software industry for more than 25 years as a software developer, architect, and technical strategist. He is the author of many published articles and the book *Model-Driven Architecture: Applying MDA to Enterprise Computing*, and was the lead editor of the book *The MDA Journal: Model Driven Architecture Straight from the Masters*. David is currently the lead standards architect for model-driven systems at SAP Labs.

<sup>7</sup> “Software Product Lines,” by the Carnegie Mellon Software Engineering Institute, [www.sei.cmu.edu/productlines/index.html](http://www.sei.cmu.edu/productlines/index.html).