



# Service Engineering: Linking Business and IT

*Tiziana Margaria*, Universität Potsdam

*Bernhard Steffen*, Universität Dortmund

**Service-oriented design has long driven the development of the telecommunications infrastructure and applications, especially intelligent network services. Applying the same principles of domain specificity, virtualization, loose coupling, and seamless vertical integration to business processes has the potential to lead to a new generation of personalized, secure, and highly available Web services.**

**B**usiness processes are increasingly becoming the distinguishing intellectual property of leading enterprises. In addition, recent regulations like the US Sarbanes-Oxley Act of 2002 and the international Basel II Accord, updated in July 2006, require more transparent process modeling, including just-in-time audits and retraceability of business decisions and operations. These trends put enormous pressure on organizations to automate, standardize, and often radically reorganize their business processes.

With its strong emphasis on modularization, service-oriented computing radically alters the way business processes are modeled, realized, and maintained. Domain-specific services virtualize complex functions of the underlying business applications so that they can be loosely coupled to form transorganizational processes. This level of abstraction fosters agility and lessens traditional provider dependence.

In classical environments, for example, the choice of an enterprise resource planning system also implies the choice of a process management system, as interoperability typically can only be guaranteed within one homogeneous system. By separating the processes from the providers of individual functions, however, virtualization and loose coupling enable seamless cross-platform, -provider, and -domain process management.

This horizontally seamless process management requires new tools to guarantee a vertically seamless realization of the modeled processes. Based on global protocol and interface standards, these tools must specify and enforce the involved functions' interoperability, thereby enabling a truly model-driven design for processes.<sup>1</sup>

## **SERVICE ORIENTATION: THE EARLY YEARS**

Since the 1990s, service-oriented design has driven development of the telecommunications infrastructure and applications, in particular intelligent network (IN) services.<sup>2,3</sup> Examples of these widely utilized families of customized telephone services include

- freephone services, which bill the receiver of calls that meet some specified conditions;
- virtual private networks, which let groups of customers define their own private network within the Internet; and
- credit card calling, which enables billing of numerous services on one credit card account.

The manual realization of new IN services was complex, error prone, and extremely costly until the emergence of a service-oriented, feature-based architecture; a corresponding standardization of basic services and applications in real standards; and adequate program-

## jABC Service Design Environment

The jABC framework, like its predecessor the METAFrame SDE,<sup>1</sup> exhibits three key features: behavior-oriented development, incremental formalization, and library-based consistency checking.

### Behavior-oriented development

In jABC, application development combines behavior-oriented service-independent building blocks (SIBs) at a coarse granular level into workflows and business processes represented as service logic graphs. SIBs are software components with a particularly simple interface that enables them to be viewed semantically as I/O transformations. SIBs are here identified on a functional basis, understandable to application experts, and they usually encompass various classical programming units like procedures, classes, modules, or functions.

### Incremental formalization

The successive enrichment of the application-specific development environment is two-dimensional. Along with a library of application-specific SIBs, which grows dynamically when new functionalities become available, jABC supports the dynamic growth of a hierarchically organized library of constraints that control and govern the adequate use of these SIBs within application programs. This library grows with the experience gained while using the environment: Detected errors, strengthened policies, and new SIBs can directly impose new constraints. The potential looseness of these constraints makes them highly reusable and intuitively understandable. Here we consciously privilege understandability and practicality of the specification mechanisms over their completeness.<sup>2</sup>

### Library-based consistency checking

Throughout the behavior-oriented development process, jABC offers access to mechanisms that verify constraint libraries via model checking, as Figure A shows. The model checker individually checks hundreds of typically very small and application- and purpose-specific constraints over the flow-graph structure. This provides concise and comprehensible diagnostic information in the case of a constraint violation at the application rather than at the programming level. Using model checking to enforce technical frame conditions and other rules has drastically reduced the need for IT during change management and boosted the realization of customer-specific IN services.

### Supported roles

These features are the key to a well-functioning distribution of labor, according to the various professional profiles. Three groups typically crystallize:

- *Programming experts* are responsible for the software infrastructure, the runtime environment for the compiled services, and SIB programming.
- *Constraint modeling experts* know the underlying infrastructure's protocols and frame conditions, formulate the correctness conditions for services to properly run and interact, and are responsible for the constraint libraries.
- *Application experts* develop concrete applications by graphically combining SIBs into coarse granular flow graphs, which they can immediately execute by means

ming environments. These developments enabled more flexible services and dramatically reduced time to market.

Independently from the telecommunication world, around 2000, service orientation reached the software engineering community as a powerful paradigm for developing Internet-based services, now called Web services, and standardized interfaces and protocols gradually enabled the use of third-party functionality over the Internet.

The telecom and software engineering communities indeed pursue the same goal: a coarse granular approach to programming in which whole programs serve as elementary building blocks. However, they view what a service is and how it should be organized quite differently. For software engineers, a service is any nugget of functionality that is directly executable and can be published for use in complex applications. In the telecom world, the notion of service is typically reserved for the overall orchestrated application, and such elementary components are called *service-independent building blocks* (SIBs).

The networks underlying IN services consist of several subsystems that together implement the intended func-

tionality. They form complex distributed systems that require the cooperation of central computers, databases, the telephone network, and numerous peripherals under real-time availability and performance constraints. In particular, the design of new services must consider requirements that the underlying intelligent network imposes—for example, system-dependent frame conditions must be obeyed to guarantee reliable service execution.

Initially, the complexity of IN services as well as the distributed environment in which they function made service definition intricate and error prone; introducing new services required several years for development and testing. In the 1990s, a *model-driven approach* using service design environments supplanted the direct-programming-based approach. By supporting reliable service design and tailoring development to the intelligent network, SDEs have reduced time to market from months to days and enabled the low-cost, high-quality services available today.

## METAFRAME SERVICE DESIGN ENVIRONMENT

In the mid-1990s, in cooperation with Siemens

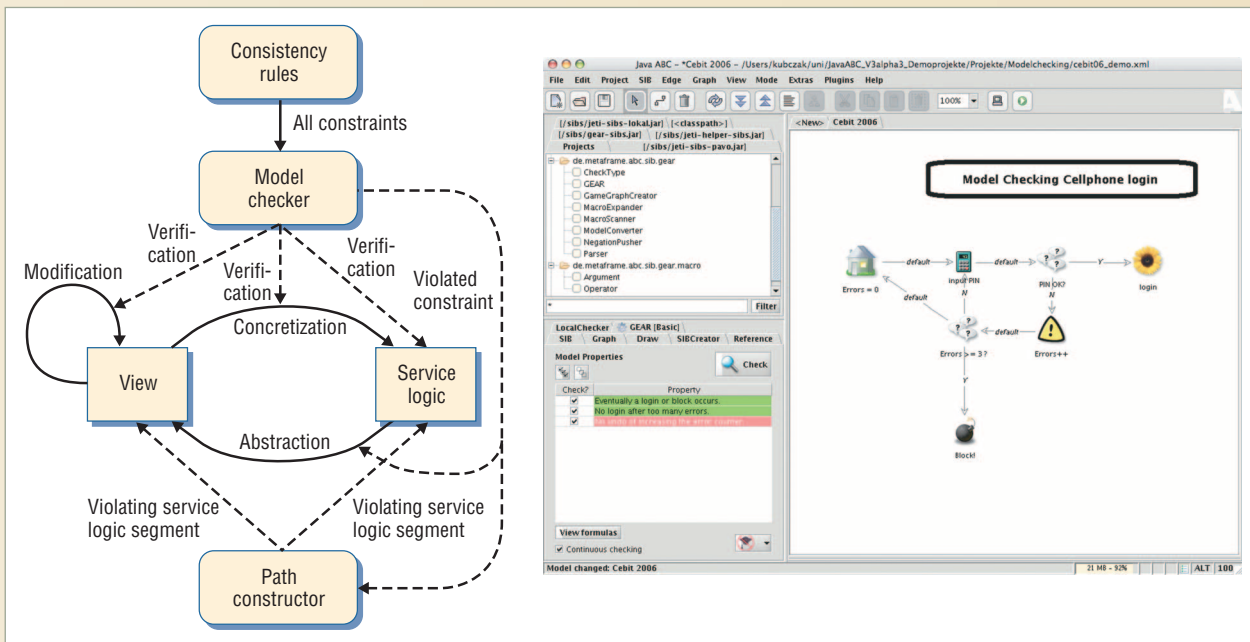


Figure A. Library-based consistency checking in jABC. The model checker individually checks hundreds of typically very small and application- and purpose-specific constraints over the flow-graph structure.

of an interpreter to validate the intended behavior (rapid prototyping). Model checking guarantees the consistency of the constructed graphs with respect to the constraint library.

Application experts thus profit most from the radical virtualization in terms of SIBs and from the fact that the loose coupling of components can be controlled via model checking.

### References

1. B. Steffen and T. Margaria, "METAFrame in Practice: Intelligent Network Service Design," E-R. Olderog and B. Steffen, eds., *Correct System Design: Recent Insights and Advances*, LNCS 1710, Springer, 1999, pp. 390-415.
2. B. Steffen et al., "Incremental Formalization: A Key to Industrial Success," *Software: Concepts and Tools*, vol. 17, no. 2, 1996, pp. 78-91.

Nixdorf, we created its SDE for advanced intelligent networks, commercialized under the name INXpress. At the time, this solution represented the state of the art in IN service definition, and its adoption by Deutsche Telekom, South Africa's Vodacom, Finland's Radiolinja, and other early adopters led to its widespread use in the late 1990s.

INXpress was based on our METAFrame SDE, a precursor of the modern jABC environment. As the "jABC Service Design Environment" sidebar describes, both METAFrame and its successor exhibit three key features: behavior-oriented development, incremental formalization, and library-based consistency checking.

### Thematic views

As Figure 1 shows, designers can use the METAFrame SDE to create flexible, reliable telephone services in a "divide and conquer" fashion—by successively modifying prototypes until they satisfy current requirements. Supporting the

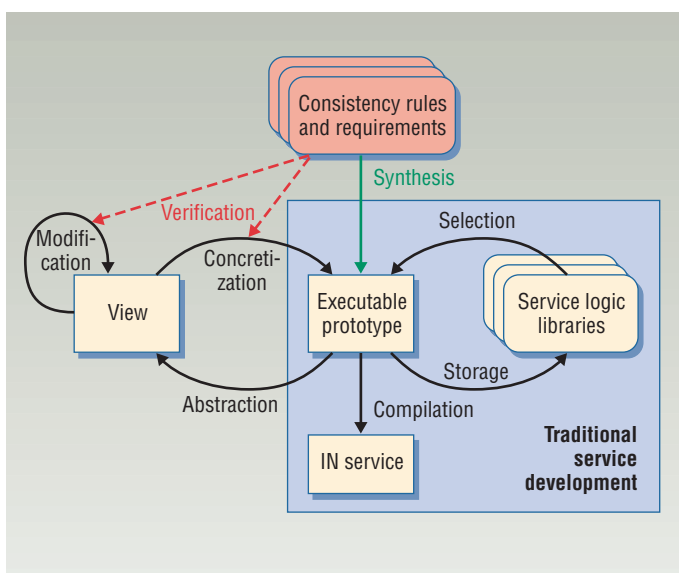


Figure 1. Intelligent network service development in the METAFrame service design environment. The use of abstract views and continual formal verification greatly simplifies service creation.

entire service creation process are thematic *views*<sup>4</sup> that abstract unnecessary details of the global context and thus let designers focus on particular aspects of the service under consideration.

Designers typically classify SIBs and reusable services according to

- technical criteria—their version, or specific hardware or software requirements;
- their origin—where they were developed; or
- their intent for a given application area.

**The ability to detect sources for typical failures already in the model rather than much later in the test laboratory greatly simplifies service design.**

The resulting *taxonomy*<sup>5</sup> is the basis for defining constraints in terms of modal formulas. The design of taxonomies goes hand in hand with the definition of aspect-specific views, as both are mutually supportive means to an application-specific structuring of the design process.

### Formal verification

The METAFrame SDE also features formal verification: Designers can immediately check the validity of an intermediate prototype's required features and executability conditions to make sure they do not conflict with the intended service's constraints and consistency conditions.<sup>5</sup> Formal verification can check the global consistency of each design step with implementation-related or service-dependent frame conditions. It relies here on fully automatic model-checking techniques and thus does not require any particular technical expertise on the user's part.<sup>4,6</sup> The ability to detect sources for typical failures already in the model rather than much later in the test laboratory greatly simplifies service design.

### Hierarchical service construction

Both formal verification and abstract views are fully compatible with hierarchical design via the METAFrame SDE's *macro* facility. This lets developers define whole subservices, or *features*, as primitive entities that can function just like SIBs. Macros can be defined online and expanded whenever their internal structure becomes relevant; in this way, the SDE supports truly hierarchical service construction.<sup>7</sup>

### Model checking

The complexity of IN services demands automated support for error detection, diagnosis, and correction. METAFrame encourages the use of new methods, as they can be introduced incrementally. If no formal constraints are defined, the SDE behaves like standard environments for model-based service creation. However, as more constraints are added, the created services become more reliable.

To enable real-time verification, the SDE uses finite-state model checkers optimized for dealing with large numbers of constraints<sup>6</sup> that verify whether a given model satisfies correctness or consistency constraints for the target IN service. Such *properties* are expressed in a natural-language-like macro language based on semantic linear-time logic (SLTL),<sup>8</sup> a variant of Dexter Kozen's modal  $\mu$ -calculus.<sup>9</sup> The models are the *service logic graphs* (SLGs), in which SIB names correspond to atomic propositions and branching conditions correspond to action names in the SLTL formulas.

Model checking a service can lead to the discovery of paths in the graph that violate some constraints. When the model checker detects such an inconsistency, a textual explanation appears in a window. To ease the location and correction of the error, the SDE automatically generates an abstract error view that shows only the relevant nodes. Errors can be corrected directly on the error view, and the subsequent view application transmits the modifications to the concrete model.

### The IN heritage: A mature service orientation

The METAFrame SDE embodies four key features of mature service orientation.

**Domain specificity.** METAFrame achieves domain specificity through use of a worldwide standardized component model, the SIB, as described in the International Telecommunication Union's Recommendation Q.1211 ([www.itu.int/rec/T-REC-Q.1211/en](http://www.itu.int/rec/T-REC-Q.1211/en)), as a basis of domain modeling.

**Virtualization.** The SDE supports virtualization of the complex, heterogeneous, distributed, multivendor IN infrastructure to guarantee seamless interoperation.

**Loose coupling.** The loose coupling between components allows their flexible combination (horizontal integration) in terms of directly executable SLGs, which can be regarded as the corresponding process description or as user processes for dealing with the IN infrastructure. As SLGs are directly executable, this supports truly model-driven service development.

**Seamless vertical integration.** Hierarchical modeling of these SLGs supports seamless vertical integration by allowing the refinement of service functionality directly to the level of elementary components.

### MAINSTREAM SERVICE ORIENTATION TODAY

Today, there is a surging service orientation movement that encompasses all aspects of software engineering and involves providers of the underlying infrastructure—hardware, middleware, databases, telecommunication, specific applications—as well as the enterprise-level decision makers responsible for shaping, reshaping, and inte-

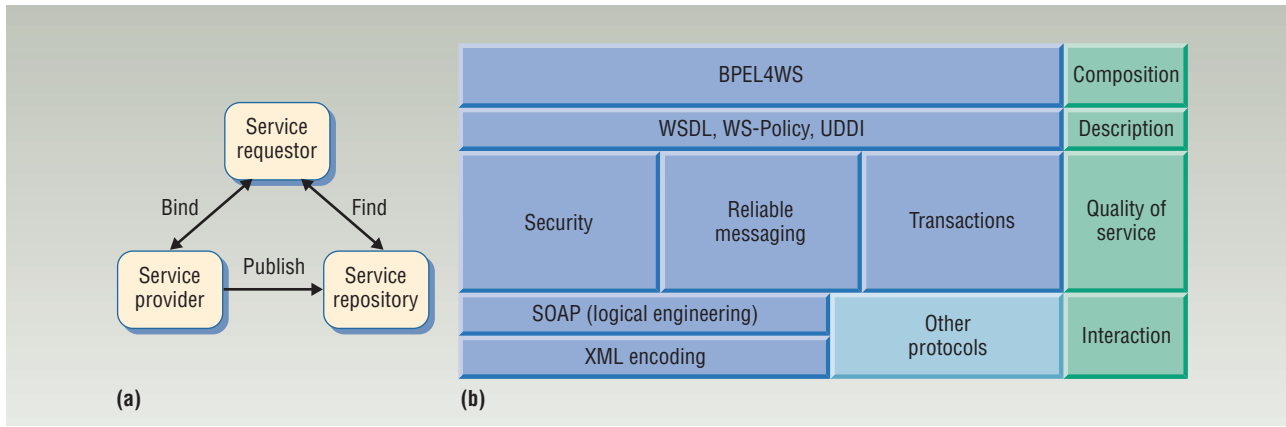


Figure 2. Service-oriented architecture. (a) SOA interaction structure. (b) Web services standards stack.

grating the cross-organizational processes that use the applications, software, and infrastructure.

Service orientation is both a top-down approach, reaching into the services from the strategic, managerial, and operative level of enterprises, covered by the business-process-modeling players, as well as a bottom-up approach: All the infrastructure players are now positioning themselves as service providers for the type of services they have always provided—storing data, shipping data, executing applications, wrapping functionalities, and so on—at various abstraction levels.

To be marketable as services, basic infrastructural entities are becoming coarser grained, more complex and articulated, and typically multimodal. For example, in the telecom domain, media convergence is pushing innovation toward “triple play” services that blend voice, video, and data on broadband wireline and wireless networks. This evolution builds on the success of INs while meeting new technological and operational challenges.

### Standardization initiatives

We can expect a similar evolution to reach the service-oriented software community as well. For example, considering the border between infrastructure and applications, the Service Availability Forum ([www.saforum.org](http://www.saforum.org)) is a consortium of industry-leading communications and computing companies—including Fujitsu Siemens Computers, IBM, Intel, Motorola, Oracle, and Veritas Software—and a large number of smaller companies offering solutions that must work with the global players’ platforms. The SA Forum is developing and publishing interface specifications for managing complex, highly available software platforms, especially services, as well as promoting their adoption by the industry at large.<sup>10</sup>

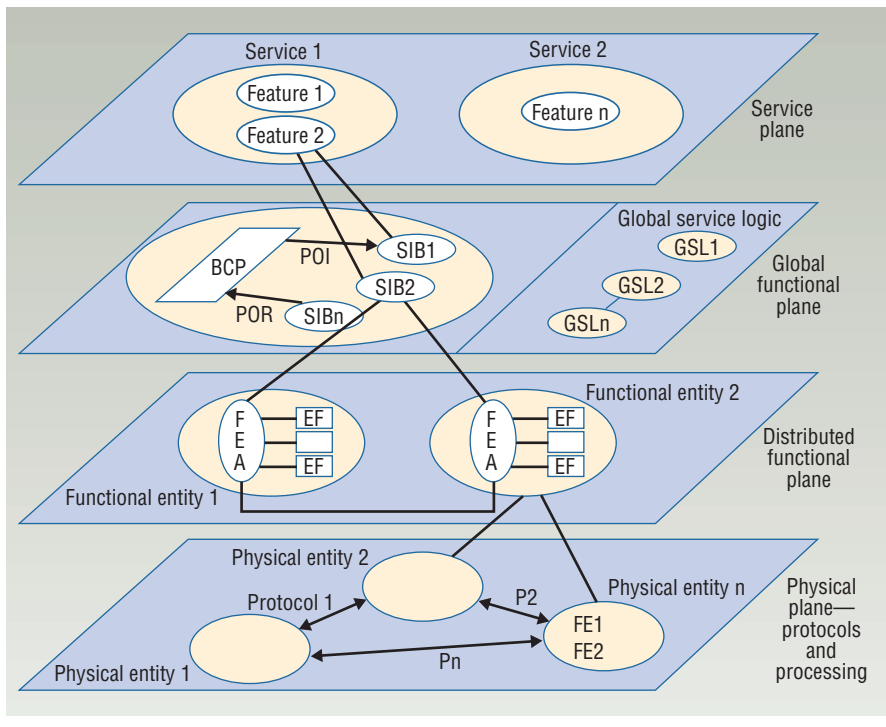
The infrastructure-software interface is also where cultural differences emerge: Participating companies widely perceive that the SA Forum’s goals far exceed the service-oriented architecture (SOA) community’s current strategy in several ways.

**Service-oriented architectures.** The SOA basic interaction structure establishes the simple, generic communication pattern shown in Figure 2a. Numerous layers for specific SOA architectures are now the object of standardization efforts, each with peculiar design decisions. XML establishes compatibility in a particularly simple *syntactic* way: via an ASCII-based framework using trees as semantic structures that can be customized to define concrete formalisms. As such, XML is not linked itself to any specific concept but is just a way of structuring and conveying domain-specific information. Because XML is ASCII, it is an open format—a major advantage over the closed proprietary interfaces still commonly used in the telecom and business domains. However, since XML defines no *semantics*, it is not sufficient to enforce any communication compatibility.

Figure 2b shows the core Web services standards stack. SOAP is an XML messaging standard that uses underlying protocols for the real communication and virtualizes them for the service layers. However, SOAP is unidirectional and stateless: It defines single messages, not conversations. Both telecom protocols and business processes describe conversations among stakeholders, so the layers both above and below SOAP must cope with being linked by a stateless layer and manage their state machines somehow.

The Web Services Description Language (WSDL) for service descriptions and UDDI for service repositories are de facto standards for today’s most popular service-oriented implementation: Web services.<sup>11</sup> A Web service implementation aims at enabling loose coupling between business partners and, because all needed interactions can be automated, just-in-time queries to locate available services (service discovery).

Nevertheless, Web services standardization concerns mostly formats, not content. Standardized formats include domain-specific languages for defining service interfaces (WSDL), data description languages (XML derivatives like RDF and other proposals such as OWL-S, WSMO, or lightweight solutions like WSDL-S), behavior com-



**Figure 3. Telecommunication services layers. The IN service conceptual architecture is a well-organized layering of services, features, and SIBs.**

position languages (BPEL4WS), and mechanisms that include execution (like the WSMO-based WSMX), but not the services themselves and their granularities.

This state of affairs is typical of the software design culture and different from engineering domains like telecommunications and hardware design. In both those domains, there are established catalogs of service behaviors that

- service providers in one domain must provide;
- must satisfy a given standard, but whose implementation might differ—for example, by resorting to different technologies or platforms; and
- must be capable of interoperation, in the sense that they are guaranteed to be interchangeable.

Mainstream service orientation today lacks the depth customary for mature sectors of the economy and is much weaker than that described for IN telecom services.

**Standardizing services and their ingredients.** Remarkably, the first attempt to specify precise granularity and modes of composition—parallel for conversations among services and hierarchical for building larger services that themselves use services—the Service Component Architecture (SCA; [www-128.ibm.com/developerworks/library/specification/ws-sca](http://www-128.ibm.com/developerworks/library/specification/ws-sca)) was proposed less than a year ago in November 2005. In contrast, the IN service architecture of the late 1980s defines a well-organized layering of services, features, and SIBs.

As Figure 3 shows, a *service plane* defines the collection of services provided by the platform to the users, and the structure of each service is described in increasing detail within well-defined layers.

The *global functional plane* corresponds to METAFrame SLGs for the user-level service: It defines the choreography of features and SIBs that collaboratively form the service and the communication with the backbone ISDN service. This communication concerns the ISDN basic control process (BCP) and IN interaction at the point of interrupt (POI) and point of return (POR), and it is greatly simplified in the case of Web services.

The *distributed functional plane* describes the orchestration of single features, which are reusable subservices also described as SLGs, in terms of functional entities and their distribution at different locations.

The *physical plane* deals with the single basic services, the functional entities, their actual deployment as physical entities (PEs), and the actual communication between them via protocols.

The concept of concrete sets of *features* and *services* as objects of standardization, which is natural and well accepted in the telecom world, is still extraneous to the SOA community. A widely usable and robust domain-specific standardization of concrete services in one branch of economy is still not yet in sight, although standardized abstraction levels and basic primitives are clearly critical for building reusable sets of services. Commercially successful and mature design environments based on libraries of functions have been available in the integrated circuit industry for decades—Xilinx ([www.xilinx.com](http://www.xilinx.com)), for example. Only with such domain-specific standardizations can Web services realize the full potential of service orientation.

In particular industrial sectors, agreed-upon names for categories and services are slowly emerging. In the business-to-business trading domain, for example, a collection of proposals for informal specifications of business processes is available from the RosettaNet consortium ([www.rosettanet.org](http://www.rosettanet.org)). Although the Web site speaks of standards, the specifications of single business processes found there are undergoing continuous revision, often at the lowest level of granularity: that of a single business action.

The repository is organized in thematic clusters of elementary business process components called partner interface processes that are described individually. Each PIP provides a high-level graphical description of the process and concrete specifications as document type definitions (DTDs) of the exchanged messages. Complementing the specifications of the business processes is the Implementation Framework—a specification of the runtime environment in terms of packaging, routing, and transport of all PIP messages and business signals. This development, however, paralleled and was independent of the Web services movement.<sup>12</sup>

Ontology-based approaches, one of the emerging technologies currently intensively investigated within the Semantic Web paradigm, handle this in a more abstract way. The RosettaNet dictionaries are examples of informal collections of concepts and could become a core for simple domain-specific ontologies. However, today's ontologies are still operationally insufficient, as they are closer to isolated solutions than to accepted standards.

### Still a long way to go

Due to its still enormous degrees of freedom, mainstream service orientation in software engineering, more specifically in the area of Web services, thus falls well short of the maturity of the IN model. Altogether, there seems to be an ongoing quest for the right abstraction level and repartition of responsibilities, which is reflected in the current imbalance in the levels and scopes of the Web services stack's different elements.

**Domain specificity.** A global and stable agreement on a domain model, even within a particular subdomain, remains elusive despite numerous ongoing initiatives. This may be due to a tendency to be too general. We also lack openness to the communities outside XML. For example, how to deal with telecom processes, which model in the Specification and Description Language (SDL), or with embedded systems processes, which model in Simulink, is not a current concern.

**Virtualization.** There are many initiatives for establishing layers of virtualization, like CORBA and SOAP at the transport layer, Java at the application layer, and BPEL or WSDL at the process layer. In fact, this dimension is arguably the most advanced, even if the attempts focus strongly on syntax (through variations of XML) and do not really reach a level where useful semantic modeling is possible.

**Loose coupling.** Currently, the emphasis is on generality rather than on simplicity, even though there is a strong demand for the flexible management of complex services and cross-organizational processes, which

would profit from a simple, tailored component model supporting high-level orchestration. Thus, model-driven service development is still far beyond reach.

**Seamless vertical integration.** Service refinement in terms of services is foreseen as part of the SCA architecture and thus still being defined. The fact that current standards have stateless protocols (units of communications) and component descriptions (units of execution) pushes the treatment of states into the concrete protocols used (that is, into the virtualized infrastructure), into the orchestration level (that is, upward into the business processes), or into the ontology (that is, into the domain modeling). Thus, some conceptual work remains.

In enterprises, service orientation will likely be designed from the top down and implemented from the bottom up.

### OUR VIEW AND VISION

The transition to a truly service-oriented paradigm still requires substantial effort in various dimensions, three of which are likely to change the way IT is generally perceived and practiced.

#### From business to IT

In enterprises, service orientation will likely be designed from the top down and implemented from the bottom up. "Design" of service orientation is moving up the corporate ladder and in time will inevitably originate at the strategic and management level, which defines and monitors business-process compliance with the company's strategy, the markets, legislation, and government regulations. These concerns are concretized in terms of strategic policy frameworks, which set the rules of the game: They are translated into constraints that all processes within the enterprise and between the enterprise and its environment—suppliers, customers, service providers, consultants, banks, and tax and legal offices—must respect.

Basel II in Europe and the Sarbanes-Oxley Act in the US are examples of such regulations, with both a global (horizontal) and pervasive (vertical) impact: They ripple from the banking sector via credit regulation to any enterprise, in terms of risk identification, quantification, and management; and from the USA to enterprises worldwide, either mandatorily, to suppliers of US companies, or voluntarily, in terms of generally raising the level of good practice.

Basel II dramatically changed how banks do business; there are more rules and dependencies to respect, and even though they do not prescribe or prohibit any concrete action, they pervade as a general framework the bank's activities and attitude. The resulting collections of conformance constraints (enacted top-down) must be accounted for at the IT level (enacted bottom-up) for each concrete action in the entire enterprise.

## Agility for business and technology

Increased agility means being able to adapt in a goal-driven fashion all the business processes and actions in a company at an accelerated pace. Being able to reschedule and reshape processes and quickly scale them up or down tightly in line with the market can be the difference between profit and loss. Following 9/11, for example, air carriers with more agile technologies and platforms in place were able to adjust to the large and rapid decrease in demand and avoid flying for months with nearly empty aircraft while other companies experienced immense losses.

Central here is the increased level of virtualization compared to traditional IT: With high IT virtualization, business experts could directly enact changes rather than have to cycle back through the IT department. Gone are the days when running the analysis component, while planning or following a marketing campaign, depended on the availability of database experts to formulate the queries.

Time to market is too precious for this staged use of IT; those who benefit from the IT infrastructure must also become its direct users. Consequently, IT must become easier to use: It must become itself a tool for business developers and business analysts, at least at the application or process levels. In contrast, today even marketing personnel are still hooked to IT experts to enforce any change.

## Tools and environments

The strongest change is likely to happen on the tool side: Which tools and environments are going to adequately serve users in the era of service orientation? The partition of tools, environments, and techniques currently in use reflects the managerial versus IT cultural dichotomy. While business strategists and managers rely on spreadsheets, text editors, and drawing tools like Visio to formulate requirements and capture processes in the early phases, IT experts use Unified Modeling Language diagrams to model the implementation and various programming and query languages to actually implement it.

In fact, there is no common ground between the two groups. The lack of a common master model that is directly executable and analyzable from the very beginning, adequate to mathematically prove rule compliance, and linked by refinement to the detailed model and implementation hampers communication and traceability of features and dependencies.

Even workflow definition systems such as QPR ([www.qpr.com/Products/QPRColSuite/index.html](http://www.qpr.com/Products/QPRColSuite/index.html)) or ARIS ([www.ids-scheer.com/international/english/products/49622](http://www.ids-scheer.com/international/english/products/49622)) are inadequate to serve as this link because

the models are often too verbose, seldom executable, insufficiently formal to be analyzable, and typically not linked to the implementation. More formal tools, like  $\pi$ -calculus or those based on variants and extensions of Petri nets such as YAWL ([www.yawl.fit.qut.edu.au](http://www.yawl.fit.qut.edu.au)),<sup>13</sup> also fail to reach business experts as they typically tend to express a rich semantics that is counterintuitive to management, or are too low-level to directly express business needs.<sup>14</sup>

## A shining path ahead

The essence of service orientation will be reflected in mature environments encompassing the following criteria, which are central to obtaining service-oriented platforms that are technologically flexible and agile, yet easy to use by the domain experts.

**Domain specificity.** Basic services for standardized access to applications and infrastructural layers are a commodity, and they form the basis for domain-specific modeling and orchestrated intra- or cross-organizational services.

**Virtualization.** The highly complex, heterogeneous, distributed, multivendor composition of applications and infrastructure is virtualized in terms of reliable, replicated services to guarantee seamless interoperability and availability.

**Loose coupling.** The looseness of coupling is based on contracts in an assume/guarantee paradigm expressed in a declarative way. This enables non-IT experts to use process descriptions similar to extended SLGs to design new flexible combinations (horizontal integration of services). The SLGs are directly executable and subject to reasoning and analysis, thereby supporting truly model-driven service development.

**Seamless vertical integration.** Recursive refinement combined with declarative, checkable specifications and a compatible semantic model across the abstraction levels allows the vertical integration of service functions directly from business-process capture to the code and hardware level.

## EXPERIENCE WITH SERVICE-CENTERED CONTINUOUS ENGINEERING

Early experience in the telecom era, and with more recent industrial projects using jABC ([www.jabc.de](http://www.jabc.de)), shows that service-centered continuous engineering ([www.scce.info](http://www.scce.info)) is indeed possible. As Figure 4 shows, with jABC we can provide continuous support from the strategic-management level down to the level of elementary, directly executable components, with a consistent and simple semantic model based on finite-state automata with fork-join parallelism. This is sufficient to cover the entire spectrum of agile-process-oriented design—extensible, if necessary,

The partition of tools, environments, and techniques currently in use reflects the managerial versus IT cultural dichotomy.

down to the coding level of software and even to the hardware.

Based on *lightweight process coordination*,<sup>15</sup> jABC makes it easy for nonprogrammers and technical experts to collaborate on complex software projects. Product developers and system/software designers can easily develop services and applications by composing reusable building blocks into hierarchical flow-graph structures that are executable models of the application. Supporting this process is an extensible set of plug-ins with additional functionalities that make it possible to animate, analyze, simulate, verify, execute, and compile jABC models.

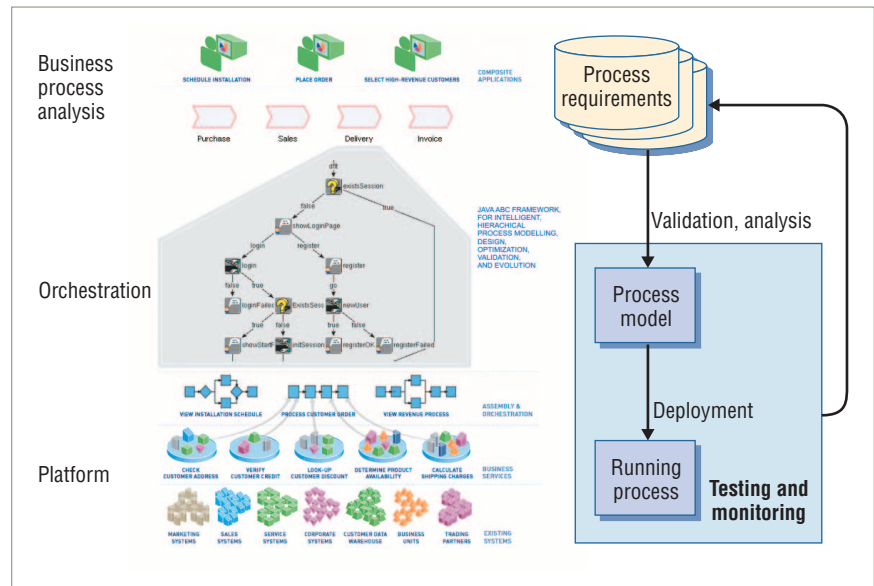
In fact, our approach to service definition, composition, and verification has been meanwhile successfully applied in other application domains: With jABC, we have built Internet-based distributed decision support systems,<sup>16</sup> an integrated test environment for regression test of complex computer telephony integration (CTI) systems,<sup>17</sup> and a management infrastructure for remote intelligent configuration of systems (MaTRICS),<sup>18</sup> as well as many other industrial applications in e-business, supply chain management, and production control systems. Among these are complex supply-chain management models with Ikea and a factory-like application development platform for future Galileo services. Although these projects were very different in nature, we observed a surprisingly high potential of synergy due to the jABC approach.

In the area of Internet-based service orchestration and coordination, we have developed the Electronic Tool Integration (ETI) platform<sup>8</sup> and its Web-services-based successor, jETI.<sup>19</sup> The jETI framework, which is based on jABC, uniquely provides

- lightweight remote component integration by registration,
- distributed component libraries,
- a graphical coordination environment, and
- a distributed execution environment

independent of the underlying technology for the components and the basic services.

Currently, jETI's application focus is on tools for program analysis, verification, and validation—for example, in the jETI FMICS platform of automatic verification tools for industrial critical systems (<http://jeti.cs.uni-dortmund.de/fmics>); dataflow analysis of Java programs;<sup>20</sup> modeling and execution of bioinformatics



**Figure 4. Agile process-oriented design.** The jABC service design environment provides continuous support for service design and application development from the strategic and management level down to elementary, directly executable components.

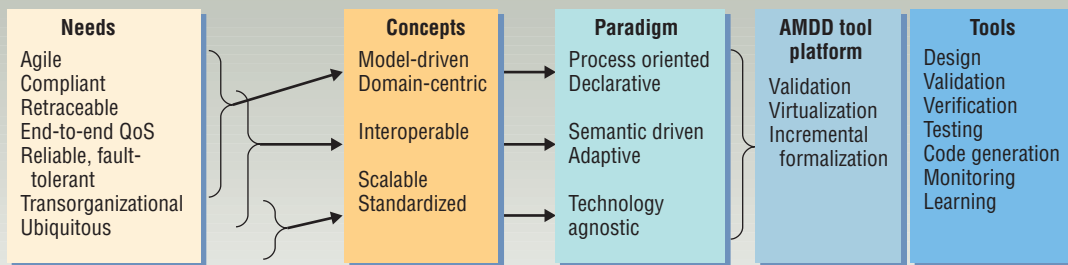
workflows;<sup>21</sup> and solving the Semantic Web Services Challenge (<http://sws-challenge.org>) for service mediation and service discovery.<sup>22</sup>

Altogether, this approach supports evolution, not revolution, within an agile, model-driven style along the entire life cycle, as illustrated by Figure 4 for the application to business services and by Figure A in the “jABC Service Design Environment” sidebar for the general development cycle in METAFrame and in jABC.

**M**ainstream service orientation remains focused on the infrastructural level: Standardization in Web services addresses how to program and plug, rather than providing a declarative, semantic-driven layer, and catalogs of basic and compound services comparable to those in the telecom industry are not available. However, we are convinced that the concept of concrete sets of features and services as objects of standardization will emerge here as well, driven by need.

In fact, we believe that service orientation will dramatically impact business organization and system and software architectures: In the long run, the dominant commercial software's architectures will become de facto standards and constitute standardization seeds for the remaining applications, thereby offering a standardized means for ensuring adequate virtualization and decoupling of components in a service-oriented framework.

This paradigm shift will fully *decouple* the rate of software systems updates (slow, and slowing further down) from the update rate of business processes (fast, and still speeding up). Rapid change at the process level will no longer be an implementation issue, and virtualization



**Figure 5. Virtualization and decoupling of components will lead to a new generation of personalized, secure, and highly available services with aggressive model-driven design (AMDD) tool support.**

will make software updates transparent. In addition, software updates will be far more *incremental*, as the exchange of loosely coupled components in a framework based on virtualization can occur stepwise with little global impact on their environments.

Figure 5 shows a conceptual design path for domain-specific design of service definition environments and service oriented platforms: The *needs* for services and applications (agility, retraceability, global quality of service, and so on) are first linked to the necessary *concepts* (model-driven approach, domain-centric style, interoperability, and so on), determining the central traits of a suitable *paradigm* (technology agnostic for virtualization, semantic driven for verification, adaptive for agility, and so on) and the resulting characteristics of an *agile, model-driven tool platform* for aggressive model-driven design (AMDD) and the contained *tools* for designing and developing those services and applications.

We are convinced that combining the flexibility of the service-oriented scenario with the rigor and semantic standardization culture of the telecommunication community can be the key to a new generation of personalized, secure, and highly available services. In particular, we believe that using declarative constraints to control the evolution of processes and applications will complement the virtualization and decoupling effects to such an extent that application experts can safely control and effect most changes.

This is akin to what happened in the IN application domain in the 1990s, when the use of model checking to enforce technical frame conditions and other rules and regulations drastically reduced the need for IT during change management and the realization of customer-specific IN services. Acceptance of this type of incremental formalization was high, and the adoption hurdle proved to be low: Adding constraints directly impacts the developed service's quality, and the system gains power as the constraint library grows.

Of course, the impact of service orientation depends on adequate domain modeling, including the SIBs and constraint libraries, and is therefore not a genuine programming paradigm, particularly for highly perfor-

mance-sensitive custom applications. However, we believe that its influence will grow in the future to cover most standard application scenarios and thus more than 90 percent of the market. ■

## References

1. T. Margaria and B. Steffen, "Aggressive Model-Driven Development for the Management of Service Evolution," *Ann. Rev. Comm.*, vol. 57, Int'l Eng. Consortium, 2004.
2. T. Margaria, B. Steffen, and M. Reitenspiess, "Service-Oriented Design: The Roots," B. Benatallah, F. Casati, and P. Traverso, eds., *Proc. 3rd Int'l Conf. Service-Oriented Computing*, LNCS 3826, Springer, 2005, pp. 450-464.
3. B. Steffen et al., "An Environment for the Creation of Intelligent Network Services," *Intelligent Networks: IN/AIN Technologies, Operations, Services, and Applications*, Int'l Eng. Consortium, 1996, pp. 287-300.
4. V. Braun et al., "Automatic Error Location for IN Service Definition," T. Margaria et al., eds., *Services and Visualization: Towards User-Friendly Design*, LNCS 1385, Springer, 1998, pp. 222-237.
5. B. Steffen et al., "A Constraint-Oriented Service Creation Environment," *Proc. 2nd Int'l Conf. Practical Applications of Constraint Technology*, Practical Application Co., 1996, pp. 283-298.
6. B. Steffen et al., "The Fixpoint-Analysis Machine," I. Lee and S.A. Smolka, eds., *Proc. 6th Int'l Conf. Concurrency Theory*, LNCS 962, Springer, 1995, pp. 72-87.
7. B. Steffen et al., "Hierarchical Service Definition," *Ann. Rev. Comm.*, vol. 51, Int'l Eng. Consortium, 1997, pp. 847-856.
8. B. Steffen, T. Margaria, and V. Braun, "The Electronic Tool Integration Platform: Concepts and Design," *Int'l J. Software Tools for Technology Transfer*, vol. 1, nos. 1-2, 1997, pp. 9-30.
9. D. Kozen, "Results on the Propositional  $\mu$ -Calculus," *Theoretical Computer Science*, vol. 27, no. 3, 1983, pp. 333-354.
10. M. Reitenspiess, "High Availability and Standard Interfaces—The Way to Go!" *Boards & Solutions*, Apr. 2004, pp. 34-36; [www.saforum.org/press/articles/SAFBoardsandSolutions\\_Apr04.pdf](http://www.saforum.org/press/articles/SAFBoardsandSolutions_Apr04.pdf).
11. G. Alonso et al., *Web Services: Concepts, Architectures and Applications*, Springer, 2004.

12. S. Damodaran, "B2B Integration over the Internet with XML—RosettaNet Successes and Challenges," *Proc. 13th Int'l Conf. World Wide Web*, ACM Press, 2004, pp. 188-195.
13. R. Milner, *Communicating and Mobile Systems: The  $\pi$ -Calculus*, Cambridge Univ. Press, 1999.
14. W.M.P. van der Aalst, "Pi Calculus versus Petri Nets: Let Us Eat 'Humble Pie' Rather Than Further Inflate the 'Pi Hype,'" *BPTrends*, May 2005, pp. 1-11; <http://is.tm.tue.nl/research/patterns/download/bptrendsPiHype.pdf>.
15. T. Margaria and B. Steffen, "Lightweight Coarse-Grained Coordination: A Scalable System-Level Approach," *Int'l J. Software Tools for Technology Transfer*, vol. 5, nos. 2-3, 2004, pp. 107-123.
16. T. Margaria, "Components, Features, and Agents in the ABC," M.D. Ryan, J-J. Ch. Meyer, and H-D. Ehrich, eds., *Proc. Int'l Seminar on Objects, Agents, and Features*, LNCS 2975, Springer, 2003, pp. 154-174.
17. B. Steffen et al., "Service Creation: Formal Verification and Abstract Views," *Proc. 4th Int'l Conf. Intelligent Networks*, 1996, pp. 96-101.
18. M. Bajohr and T. Margaria, "MaTRICS: A Service-Based Management Tool for Remote Intelligent Configuration of Systems," *Innovations in System and Software Eng.*, vol. 2, no. 2, 2006, pp. 99-111.
19. T. Margaria, R. Nagel, and B. Steffen, "Remote Integration and Coordination of Verification Tools in JETI," *Proc. 12th IEEE Int'l Conf. and Workshops on the Eng. of Computer-Based Systems*, IEEE CS Press, 2005, pp. 431-436.
20. A-L. Lamprecht, T. Margaria, and B. Steffen, "Data-Flow Analysis as Model Checking within the jABC," A. Mycroft and A. Zeller, eds., *Proc. 15th Int'l Conf. Compiler Construction*, LNCS 3923, Springer, 2006, pp. 101-104.
21. T. Margaria et al., "Model-Based Design of Distributed Collaborative Bioinformatics Processes in the jABC," to appear in *Proc. 11th IEEE Conf. Eng. Complex Computer Systems*, IEEE CS Press, 2006.
22. C. Kubczak et al., "The jABC Approach to Mediation and Choreography," *Semantic Web Services Challenge 2006*; [http://leo.cs.uni-dortmund.de:8000/papers/sws\\_challenge\\_jabc.pdf](http://leo.cs.uni-dortmund.de:8000/papers/sws_challenge_jabc.pdf).

*Tiziana Margaria is chair of service and software engineering at the Institute of Informatics, Universität Potsdam, Germany. Her research focuses on model-based system and service engineering as well as biologically inspired computing. Margaria received a PhD in computer and systems engineering from the Politecnico di Torino, Italy. She is a member of the IEEE, the ACM, the International Federation for Information Processing (IFIP), and the German Association for Computer Science (GI) as well as president of the European Association of Software Science and Technology (EASST). Contact her at [margaria@cs.uni-potsdam.de](mailto:margaria@cs.uni-potsdam.de).*

*Bernhard Steffen is chair of programming systems in the Department of Computer Science, Universität Dortmund, Germany. His research interests include program analysis and optimization, model-based analysis of distributed systems, and service-oriented computing. Steffen received a PhD in computer science from the Christian-Albrechts-Universität zu Kiel, Germany. He is a member of the IEEE, the ACM, IFIP, the GI, and EASST. Contact him at [steffen@cs.uni-dortmund.de](mailto:steffen@cs.uni-dortmund.de).*

## Stay on Track

IEEE Internet Computing reports emerging tools, technologies, and applications implemented through the Internet to support a worldwide computing environment.

In future issues, we'll look at

- Malware/spyware
- Autonomic computing
- Roaming
- Distance learning
- Dynamic information dissemination

... and more!

**IEEE**  
**Internet Computing**

[www.computer.org/internet](http://www.computer.org/internet)

