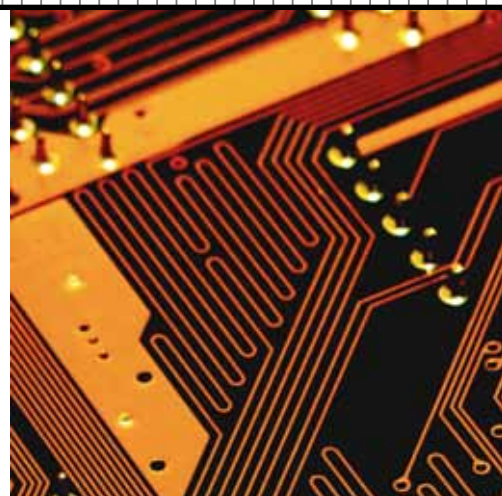


Commitment-Based Service-Oriented Architecture



- ➔ **Munindar P. Singh**, *North Carolina State University*
- ➔ **Amit K. Chopra**, *Università degli Studi di Trento, Italy*
- ➔ **Nirmit Desai**, *IBM India Research Labs*

Existing service-oriented architectures are formulated in terms of low-level abstractions far removed from business services. In a new SOA, the components are business services and the connectors are patterns, modeled as commitments, that support key elements of service engagements.

The vision of service-oriented computing (SOC) promises the creation of a dynamic Web of value. According to this vision, anyone desiring to offer something of value can create and deploy a corresponding service; anyone wishing to benefit from that value can simply select one or more services and compose them into a desired application—or another service.

Current service-oriented architectures (SOAs) purport to support the SOC vision, but what they realize is fundamentally more limited than the vision. The SOC vision implies that services are business services. However, current SOAs interpret services narrowly—as surrogates for computational objects. Whereas business services are *engaged* (often involving subtle business considerations), objects are *invoked* (with business considerations hidden within computational artifacts). More importantly, business services are usually autonomous entities that come together in a service engagement.

Consider the familiar purchase scenario as modeled in leading SOA approaches. Purchasing, say, books is a business service that combines individual services such as placing an order, paying, and shipping. Different organizations could provide these services.

Business Process Modeling Notation (BPMN; <http://bpmn.org>) and the Business Process Execution Language (BPEL; <http://docs.oasis-open.org/wsbpel/2.0>) represent composed services as processes specified via control and data flows over tasks (the differences between BPMN and BPEL are syntactic; <http://bpmn.org/Documents/Mapping%20BPMN%20to%20BPEL%20Example.pdf>). For example, BPMN would model a purchase as three tasks—ordering, paying, shipping—where control and data (book identifier and price) flow from ordering to both paying and shipping.

The Choreography Description Language (WS-CDL; www.w3.org/TR/ws-cdl-10), another leading SOA approach, specifies how services exchange messages. Unlike procedure calls, messaging decouples the parties involved and is thus better suited for distributed systems. WS-CDL would specify how the ordering service sends messages to the paying and shipping services, which perform their work upon receipt of such messages. Declarative approaches for constraining task or message order and occurrence improve modularity and inspectability^{1,2} but continue to emphasize control and data flow.

COMMITMENTS AND CSOA BENEFITS

In contrast to existing approaches, commitment-based SOA (CSOA) gives primacy to service engagements' *business meanings*, which it captures through participants' *commitments* to one another. CSOA constrains tasks or messages only when doing so affects the business meaning. Computationally, it represents each participant as an agent; interacting agents carry out a service engagement by creating and manipulating commitments to one another.

Commitments

A commitment relates three parties: a *debtor* who is committed to a *creditor*, typically within the scope of an organizational *context*. The context may be an institution—for example, a marketplace such as eBay or a legal jurisdiction such as California—in which the interaction occurs. Institution members who fail to discharge their commitments risk sanction. The Uniform Commercial Code (UCC; www.law.cornell.edu/ucc), which applies in many US jurisdictions, dictates conditions such as when a customer need not pay for purchased goods—for instance, if the goods arrive damaged and the customer returns them immediately. In general, the context is crucial in handling exceptions, which are rife in business settings. For modeling purposes, CSOA treats the context as an agent in its own right.

Importantly, commitments can be manipulated, which supports flexibility. A debtor may *create* a commitment, thus activating it, or *discharge* it, thus satisfying it. Given a commitment, its creditor may *assign* it to a new creditor and its debtor may *delegate* it to a new debtor. A debtor may cancel a commitment, whereas a creditor may *release* the debtor from the commitment.

CSOA benefits

CSOA thus offers the following specific benefits.

Enactment and compliance. Service enactments can be judged correct as long as the parties don't violate their commitments. This notion of correctness enhances flexibility by expanding the operational choices for each party.³ For example, if the customer substitutes a new way to make a payment or elects to pay first, no harm is done because the behavior is correct at the business level. The seller can employ a new shipper; the buyer can return damaged goods for credit; and so on. Conversely, a customer would be in violation if he keeps the goods but fails to pay. Thus, commitments support business-level compliance without dictating specific operationalizations:⁴ Without business meaning, exercising such flexibility could cause noncompliance.

Specification and composition. Commitment-based specifications explicitly reflect business requirements, which are natural for stakeholders. For example, upon

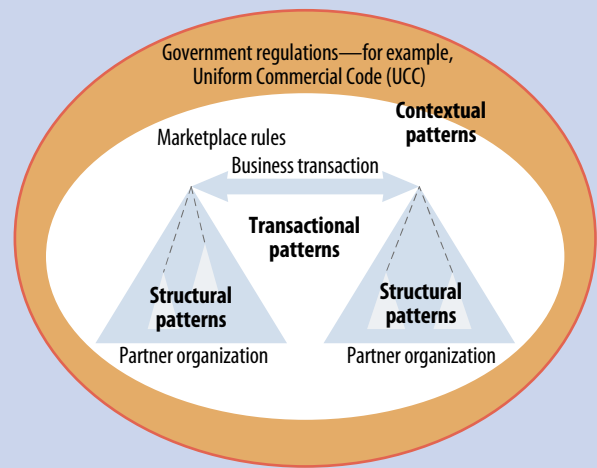


Figure 1. Commitment-based SOA patterns. Transactional patterns refer to the dealings among two or more participants, structural patterns refer to how a participant is organized, and contextual patterns refer to the organizational context in which the service engagement takes place.

placing an order, the customer becomes conditionally committed to the merchant to pay for the goods if they are delivered. The delivery of the goods unconditionally commits the customer to paying for them. When the customer pays, this commitment to pay is discharged. Commitments provide clear conceptual boundaries at which to compose service engagements. For example, we can specify an alternative service engagement that employs independent delivery and payment services. Without business meaning, there would be no basis for establishing that this alternative engagement was valid.

CSOA PATTERNS

As Figure 1 shows, CSOA is characterized by a family of reusable *patterns* that form the elements of a service engagement: *Transactional* patterns refer to the dealings among two or more participants; *structural* patterns refer to how a participant, including subcontractors, is organized; and *contextual* patterns refer to the organizational context in which the engagement takes place.

Key CSOA patterns are induced from existing approaches, including UCC, RosettaNet (www.rosettanet.org), the Transaction Workflow Innovation Standards Team (TWIST; www.twiststandards.org), the MIT Process Handbook (MITPH; <http://ccs.mit.edu/ph>), and extended transaction models.⁵ These approaches are not commitment based, but we analyze them via commitments and include the induced patterns within CSOA.

Commitment life cycle

CSOA pattern implementations are expressed as statecharts⁶ as shown in Figure 2. Labeled rectangles denote *states*. A state that refines another state is contained within

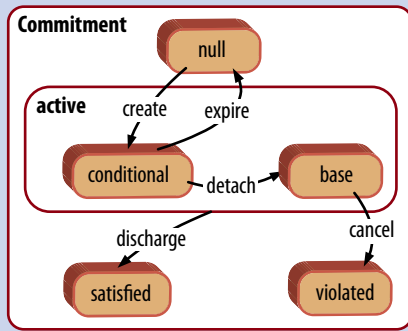


Figure 2. CSOA patterns are expressed in statecharts like this one, which captures a commitment's life cycle.

it. For example, **null** and **active** (containing **conditional** and **base**) are states. An arrow denotes a *transition* wherein the labeled event, if any, occurs. A transition takes the system from one state to the next. When the source state has substates, the transition occurs from each of them—for example, *discharge*.

Figure 2 captures a commitment's life cycle: **null** means it does not exist, **active** means it is fully in force, **satisfied** means it has been discharged, and **violated** means it cannot be discharged. A commitment in **base** may become **violated**; a commitment in **conditional** cannot directly become **violated** but transitions to **null** upon expiration. For example, a customer may offer to buy some goods by creating the commitment "If you ship I will pay." The commitment may expire or the customer may pay. If the merchant delivers, that would detach the commitment, unconditionally committing the customer to pay.

Each commitment has an *antecedent* and a *consequent*. The expression C (debtor, creditor, context, antecedent, consequent) means that the debtor commits to the creditor in the context that if the antecedent becomes true, the debtor would bring about the consequent. When the antecedent holds, the commitment undergoes a detach, meaning that the debtor becomes unconditionally committed to bringing about the consequent. When the consequent holds, the commitment undergoes a discharge. Figure 2 shows detach and discharge as transitions. An **active** commitment must be in either **conditional** or **base**, and this depends solely on whether its antecedent holds (**base**) or not (**conditional**).

Importantly, an agent explicitly performs *create* whereas *detach* and *discharge* occur automatically when antecedent and consequent, respectively, hold; *expire* occurs implicitly upon timeout, but an agent may perform *cancel* explicitly or it may occur via timeout.

Pattern language

Of the 13 attributes in the classical template for design patterns,⁷ the following are relevant for CSOA: *classifica-*

tion (according to Figure 1), *intent*, *motivation*, *applicability*, *consequences*, *implementation*, and *known uses*. A common consequence for CSOA patterns is that the parties involved be proactive and able to communicate flexibly—this is why they are modeled as agents.

The implementation, specified via a statechart, incorporates the *participants* and *structure*. To make the patterns modular, each statechart includes only the relevant states and transitions. (In this sense, our statecharts are not individually complete, and rely upon other patterns to have brought about the states from which they begin.) The commitment operations corresponding to a transition would be realized via business actions such as sending purchase orders, delivering goods, and so on, thereby enacting the corresponding business scenarios.

TRANSACTIONAL PATTERNS

The core of a service engagement is the business transaction that it seeks to accomplish. Transactional patterns describe the corresponding interactions in terms of how the associated commitments are created and manipulated. These patterns deal with common transactional primitives such as initiating a business transaction, formally creating suitable commitments, satisfying the commitments, and possibly updating, retrying, or compensating actions in light of the stated gating conditions. Each transactional pattern involves the same two participants.

We define the commitment life cycle in Figure 2 as the transactional pattern *Commit*, with the following attributes:

- *Intent*: Expressing an offer.
- *Applicability*: When an offer is made as part of setting up a service engagement.
- *Consequences*: For progress, the creditor should be ready to bring about the antecedent.
- *Known uses*: Purchase, MITPH's Purchase, Rosetta-Net's Purchase Order (PIP3A4).

Another important transactional pattern is *Compensate*, which has the following attributes:

- *Intent*: Some business action needs to be undone.
- *Motivation*: A customer sends payment, which commits the merchant to sending the goods; later, if the merchant fails to deliver the goods on time, thus violating its commitment, it must make amends by, for example, refunding the payment.
- *Applicability*: Supporting an extended form of transactional rollback to maintain an all-or-none effect despite exceptions.⁵
- *Consequences*: Typical usage is when the debtor is unable to discharge the original commitment.

- *Implementation:* Upon violation of the original commitment, the transaction requires creating a compensating commitment.
- *Known uses:* RosettaNet's Return Product (PIP3C1).

In the same vein, we can define transactional patterns for real-life cases such as Relieve based on RosettaNet's Purchase Order Cancel (PIP3A9) and the MITPH's Notify, Update based on MITPH's Update and RosettaNet's Purchase Order Change (PIP3A8), and Retry based on MITPH Rework to retry a failed task.

STRUCTURAL PATTERNS

Service engagements involve subtle relationships among the parties involved in a transaction. Structural patterns capture constraints on which party can play which role, or whether a party can delegate or assign certain commitments to another party. Each of these patterns involves two or more participants.

The simplest illustration of a structural pattern is a service engagement involving an organization with an internal structure. A participating organization may delegate its commitments under the engagement to appropriate members that could themselves be organizations. For example, auto insurance companies often delegate their customer service commitments to a regional branch, which might further delegate the commitments to a specific agency.

Composite states help describe patterns involving more than one commitment. For example, in Figure 3, a dotted vertical line separates **Original** and **Delegated**. Thus, if **Original** is in **pending** and **Delegated** is in **active**, the composite state is given by **Original** being in **pending** and **Delegated** being in **active**.

The structural pattern Delegate, Retaining Responsibility, shown in Figure 3a, has the following attributes:

- *Intent:* A debtor delegates its commitment but remains responsible for its satisfaction.
- *Motivation:* The merchant delegates its commitment to ship goods to a shipping service but remains committed to deliver the goods to the customer; discharging the delegated commitment discharges the original pending commitment.
- *Applicability:* When the delegatee and creditor don't have a strong business relationship.
- *Consequences:* The creditor is safe because the delegator remains responsible; this pattern enables and coheres with Escalate and Withdraw.
- *Implementation:* **Original** becomes **pending** and **Delegated** becomes **active**.
- *Known uses:* When an insurance company delegates a claimant's auto repair work to a mechanic, it remains responsible if the mechanic fails to make adequate repairs.

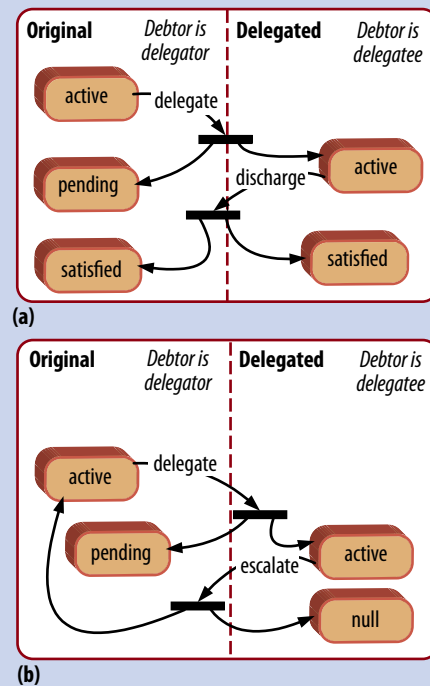


Figure 3. Structural patterns: (a) Delegate, Retaining Responsibility; (b) Escalate (Delegated Commitment). A solid bar with incoming and outgoing arrows is a synchronization primitive: When all events corresponding to the incoming arrows occur, the transitions corresponding to each outgoing arrow also execute.

A related structural pattern is Escalate (Delegated Commitment), shown in Figure 3b, which has the following attributes:

- *Intent:* The failure of a delegatee reactivates the original commitment.
- *Motivation:* If a shipper fails to deliver the goods, the merchant is held responsible.
- *Applicability:* When the delegatee does not provide guaranteed service.
- *Consequences:* The creditor would be the instigator.
- *Implementation:* **Delegated** goes to **null** and **Original** goes to **active**, thus reactivating the original commitment.
- *Known uses:* A customer who pays with a check delegates to the bank his commitment to pay the merchant; if the bank fails to pay—say, because of insufficient funds—the escalation reactivates the customer's original commitment to pay.

Sometimes the delegation transfers responsibility. This corresponds to a variation of the delegation pattern wherein the original commitment simply ends instead of becoming **pending**. Its becoming **null** forecloses the possibility of escalation.

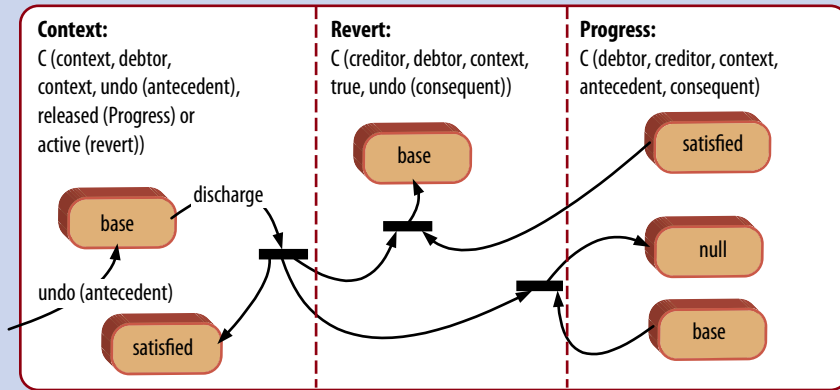


Figure 4. Contextual pattern Revert Offer. To understand this pattern, imagine a commitment Progress whose debtor is a customer who has received some goods (the antecedent) from a merchant and is therefore committed to paying for them (the consequent). The context is an agency that regulates this service engagement; it commits to the debtor that if the debtor undoes the antecedent (returns the goods) and hasn't already discharged Progress, it is released from Progress (need not pay). Conversely, if the debtor has discharged Progress, then the context activates a commitment Revert that reverses the debtor and creditor roles of Progress: Its debtor is the original creditor who must now undo the original consequent (return the payment).

The structural pattern Transfer Responsibility has the following attributes:

- *Intent:* A debtor nullifies its original commitment by delegating it to another party and is no longer concerned with the delegated commitment's satisfaction or violation.
- *Motivation:* If the customer delegates to his credit card company the payment to the merchant, the subsequent interactions for the payment occur between the company and the merchant; the customer need no longer be involved.
- *Applicability:* When the delegatee and creditor have a strong business relationship.
- *Consequences:* The creditor must accept the delegation and perhaps seek proof that the delegatee accepts it; the delegation may be risky for the creditor.
- *Implementation:* Original becomes null and Delegated becomes active.
- *Known uses:* When an airline "endorses" a ticket over to another airline based on a passenger's request, the second airline becomes responsible for transporting the ticketed passenger.

In addition, the structural pattern Withdraw Delegation applies when a delegated commitment is not yet satisfied. It nullifies the delegated commitment and restores the original commitment to active. An example is when an airline with an overbooked flight delegates its commitment to transport a passenger to another airline. If the second airline's flight is excessively delayed due to weather, the first

airline may reactivate its commitment to transport the passenger.

Yet another structural pattern is Division of Labor, where a service subcontracts a task to two or more other services. This pattern has numerous uses, including RosettaNet's Distribute Work (PIP7B1).

CONTEXTUAL PATTERNS

A service engagement's business context dictates the rules of encounter to which it is subject. For example, eBay users are subject to the online marketplace's terms and conditions, such as that they may not attempt to place false bids. More pertinently, the rules for dispute resolution are also contextual in nature. Each of these patterns involves the three participants—debtor, creditor, and context—with the context explicitly acting as a

debtor of a *metacommitment* whose antecedent and consequent involve commitments. The context has the power to create and manipulate commitments among the agents in its scope. Metacommitments provide guarantees to the participants.

In contextual patterns, the context agent itself features as a debtor or creditor. Often in such patterns the context commits to another party such that if some conditions prevail it will cause a specified commitment to transition to a suitable state.

Figure 4 shows the contextual pattern Revert Offer, which has the following attributes:

- *Intent:* To enable a party to back out of a transaction.
- *Motivation:* A customer commits to paying for some goods, which the merchant delivers. If the customer returns the goods before paying, the merchant releases him from paying; if the customer has paid, the merchant refunds the payment.
- *Applicability:* When an agency regulates the service engagement.
- *Consequences:* The context has the means to determine that the requisite conditions hold; it has power over the debtor such as removing it from a marketplace or voiding its license to operate.
- *Implementation:* An *undo(antecedent)* undoes the offer's antecedent. If Progress is in base, the system releases the debtor—Progress becomes null—and no further action is needed; if Progress is satisfied, *undo(antecedent)* cause the creation of Revert.
- *Known uses:* UCC.

An alternative contextual pattern is Penalize, which seeks to punish a party that violates a commitment. For example, if the debtor fails to pay \$10 by Monday, the new commitment could be to pay \$11 by Tuesday. If the original means commitment delivering the goods, the penalty could mean refunding the deposit and an additional 10 percent—this can be implemented by making a penalty commitment active.

APPLYING THE PATTERNS

Designing a service engagement using the CSOA patterns requires three steps:

- identify the commitments regarding the services involved,
- apply selected patterns to appropriate commitments, and
- map the operations occurring in the patterns to the engagement’s business actions.

Let’s revisit our purchase example. We begin with the main partner roles, buyer and seller, and their commitments: The buyer offers to pay if the seller ships him the goods; the seller offers to ship the goods if the buyer pays. Next, we introduce a bank and a shipper: The buyer delegates the payment commitment to the bank, and the seller delegates the shipping commitment to the shipper; the two apply different structural patterns. Last, we apply a contextual pattern enabling refunds upon return.

Figure 5 shows the resulting model, which captures the essential business meaning of the service agreement. Note that additional business requirements are accommodated simply by applying additional patterns, while the existing patterns remain as they are. In some cases, a service engagement may require additional operational constraints, such as that payments should precede shipping.

In contrast, traditional approaches such as BPMN are based solely on operational constraints. The control and data flows to capture the meaning of Figure 5 could be quite complex. Not only do the flows hide the business meaning, they also complicate accommodating additional business requirements: Even a simple change can lead to many additional intricate changes in the existing flows. Further, traditional models lack a formal representation of business meaning, instead relegating meaning to documentation. Modelers need the operationalizations, of

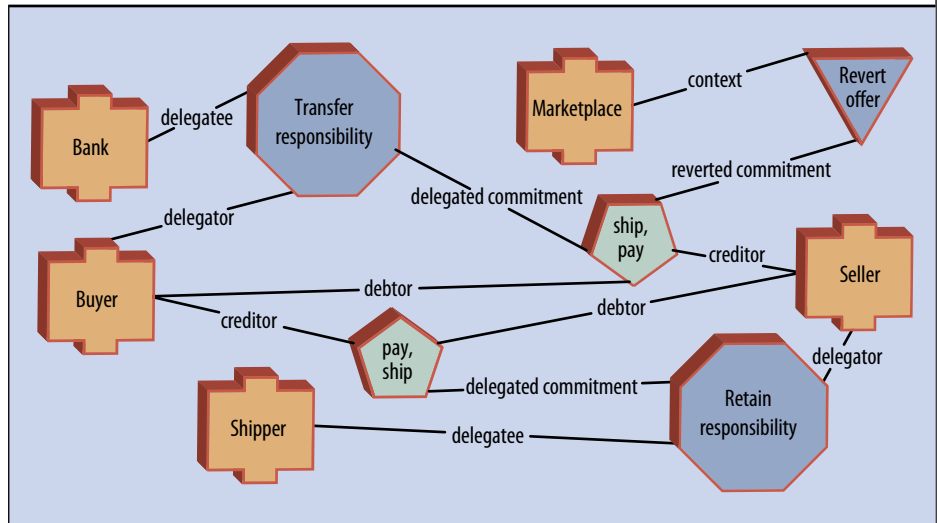


Figure 5. CSOA model of a purchase service engagement. Additional business requirements are accommodated simply by applying additional patterns, while the existing patterns remain as they are.

course, but should be concerned with business meanings, not low-level operationalizations.

To instantiate an engagement, business partners would adopt the specified roles and perform the services and other business actions specified. The patterns refer to several explicit actions, including *create*, *delegate*, *assign*, *release*, and *update*. Each such action is governed by the corresponding partner’s policy; at enactment, such policies determine what computations occur. Our prototype tools map commitment patterns to computations³ and produce role skeletons, which can be used to implement agents that can participate in an engagement.⁴

CSOA patterns describe abstract possibilities. However, applying the patterns involves matching them to the concrete business realities of a service engagement. For example, a transactional pattern allowing cancellation would make sense only if a commitment can be reasonably canceled. Further, it may not be possible to delegate a commitment if the intended delegatee would not accept the delegation. Finally, the context may not be able to ensure that an agent will discharge any commitments created by the context. In general, CSOA patterns work best when there is a suitable prior business or legal relationship among the parties involved. The patterns can guide the specification of the appropriate relationships or constraints to realize desired service engagements.

ARCHITECTURAL STYLES

An architectural style specifies a family of configurations of *components* and *connectors* subject to stated *constraints*.⁸

In these terms, existing SOAs are an architectural style in which the major components are service provider and consumer, and an invocation protocol serves as connec-

Table 1. Architectural styles of commitment-based SOA versus existing SOAs.

| Elements | Existing SOAs | Commitment-based SOA |
|------------|---|--|
| Components | Service provider and consumer | Business service provider and consumer agents |
| Connectors | Operations and message patterns (in, out, in-out, out-in) | Commitment patterns |
| Invariants | Match operation and message signatures | Debtor \neq creditor; delegator \neq delegatee |
| Model | Control and data flow | Operations on commitments |

tor. (For simplicity, we ignore registries as well as service publication and discovery.) A practical SOA includes specialized components and connectors, such as for resource management and other enterprise functions (identity, billing, and such), and imposes additional constraints so that appropriate components interoperate with each other.

Boualem Benatallah and colleagues proposed patterns called *business-level interfaces and protocols*.⁹ However, like WS-CDL and BPEL, their patterns ignore business meanings and thereby lead to rigid interoperation. For example, if a message interface specifies that a customer should make a payment subsequent to the receipt of goods, then a service realizing such an interface must behave accordingly. It ought not to take any liberties such as reversing the messages' order, interposing other messages, or introducing another party such as a payment agency. However, real-life service engagements typically presume such flexibility—thus traditional approaches subvert the SOC vision by creating avoidable friction in the web of value.

The motivation for considering business meaning is to improve the naturalness, maintainability, and reusability of service specifications and the flexibility of enactments. As Table 1, which contrasts commitment-based SOA with existing SOAs,⁸ shows, CSOA is not a unique style but has many flavors depending on the patterns selected. Such flexibility is necessary to support the nuances of service engagements. The primary constraint on a sound implementation of CSOA is that at runtime all commitments eventually become **null** or **satisfied**.

The reader may reasonably wonder why, given these differences, CSOA is still a SOA. The answer is twofold. First, CSOA is centered on services and is, arguably, more true to the SOC vision than existing SOAs. Second, CSOA doesn't seek to replace existing SOAs and their implementations. Specifically, the service engagements modeled in CSOA could translate into business processes expressed in BPMN.

A model-driven architecture (MDA; www.omg.org/mda) provides a useful way to think of the relationship between CSOA and existing SOAs. In MDA terms, CSOA is a *computation-independent* model whereas existing SOAs are *platform-independent* models. In other words, the move to CSOA would represent the step—often repeated in computer science—of moving from lower to higher abstractions. Because commitments are computation independent, yet lend themselves to rigorous operationalization, CSOA can help bridge the well-recognized gap between business and IT.¹⁰ Others have begun to recognize the importance of high-level abstractions, but their work still employs operational abstractions (www.ip-super.org).

Santhosh Kumaran¹¹ presented four abstraction layers for enterprise modeling: strategy (business considerations), operation (business functions conceptualized via tasks and artifacts), execution (analogous to existing SOAs), and implementation. CSOA would help extend Kumaran's operation layer to multienterprise service engagements, and commitment patterns would provide richer representations that facilitate modeling enterprise operations perspicuously and reusably.

Because of the subtleties of real-life service engagements, no small set of patterns would be provably complete. This is analogous to object-oriented design patterns, which are numerous and varied even though the underlying programming languages need only a few primitives.

However, despite their subtlety, service engagements for the most part exhibit regularities in how their transactions, structures, and contexts are applied. Consequently, a reasonably small set of patterns can help describe a large number of practical engagements. Thus, our main contributions are introducing a SOA that gives primacy to business interactions and showing how to formalize the concomitant patterns that provide an expressive vocabulary for modeling service engagements.

Typical service engagement models would include several CSOA patterns applied in routine ways. Thus, *aggregate service patterns*, which capture best practices in designing service engagement, can potentially be abstracted and applied in designing new engagements using CSOA. **□**

References

1. M.P. Singh, "Distributed Enactment of Multiagent Workflows: Temporal Logic for Service Composition," *Proc. 2nd Int'l Joint Conf. Autonomous Agents and Multiagent Systems (AAMAS 03)*, ACM Press, 2003, pp. 907-914.
2. W.M.P. van der Aalst and M. Pesic, "DecSerFlow: Towards a Truly Declarative Service Flow Language," *Proc. 3rd Int'l Workshop Web Services and Formal Methods (WS-FM 06)*, LNCS 4184, Springer, 2006, pp. 1-23.

3. A.K. Chopra and M.P. Singh, "Contextualizing Commitment Protocols," *Proc. 5th Int'l Joint Conf. Autonomous Agents and Multiagent Systems (AAMAS 06)*, ACM Press, 2006, pp. 1345-1352.
4. N. Desai et al., "Interaction Protocols as Design Abstractions for Business Processes," *IEEE Trans. Software Eng.*, Dec. 2005, pp. 1015-1027.
5. A.K. Elmagarmid, ed., *Database Transaction Models for Advanced Applications*, Morgan Kaufmann, 1992.
6. D. Harel, "Statecharts: A Visual Formalism for Complex Systems," *Science of Computer Programming*, June 1987, pp. 231-274.
7. E. Gamma et al., *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.
8. M. Shaw and D. Garlan, *Software Architecture: Perspectives on an Emerging Discipline*, Prentice Hall, 1996.
9. B. Benatallah et al., "Service Mosaic: A Model-Driven Framework for Web Services Life-Cycle Management," *IEEE Internet Computing*, July 2006, pp. 55-63.
10. H. Smith and P. Fingar, *Business Process Management: The Third Wave*, Meghan-Kiffer Press, 2002.
11. S. Kumaran, "Model-Driven Enterprise," *Proc. Global Enterprise Application Integration Summit (GIS 04)*, Integration Consortium, 2004, pp. 166-180.

Munindar P. Singh is a professor in the Department of Computer Science at North Carolina State University. His research interests include multiagent systems and service-oriented computing. Singh received a PhD in computer science from the University of Texas at Austin. He is a Fellow of the IEEE. Contact him at singh@ncsu.edu.

Amit K. Chopra is a postdoctoral fellow at the Università degli Studi di Trento, Italy. His research interests include service-oriented architectures and multiagent systems. Chopra received a PhD in computer science from North Carolina State University. Contact him at akchopra.mail@gmail.com.

Nirmit Desai is a research staff member at IBM India Research Labs, Bangalore. His research interests include cross-organizational business processes. Desai received a PhD in computer science from North Carolina State University. Contact him at nirmitv@gmail.com.