Commitment-Based SOA*

Munindar P. Singh, Amit K. Chopra, and Nirmit Desai

Department of Computer Science
North Carolina State University
Raleigh, NC 27695-8206
{singh, akchopra, nvdesai}@ncsu.edu

Abstract

The vision of service-oriented computing is centered on business services. By contrast, existing service-oriented architectures are formulated in terms of low-level abstractions that are far removed from business services. This paper describes a new architecture whose components are business services and whose interconnections are modeled in terms of the commitments that support key aspects of service engagements. This paper also shows how this architecture relates to existing SOAs.

Introduction

The services vision promises the creation of a dynamic web of value. According to this vision, anyone with something of value to offer can easily create and deploy a corresponding service; anyone needing to benefit from that value need simply select such a service and incorporate it into a desired solution. In other words, services are software components created, deployed, and flexibly composed to yield desired applications—or more services.

Current SOAs purport to support the services vision. But there is a fundamental discrepancy between how services are conceptualized and how they are realized. When the services vision is promulgated, the implication is that the services correspond to business services. However, current SOAs interpret services far more 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). Whereas business services are usually understood as autonomous entities, requested objects usually are not.

Our commitment-based SOA gives primacy to business interactions as modeled via the interacting parties’ commitments to one another. Commitments not only reflect service engagements naturally; they also lend themselves to operationalization in a computational sense. Thus, they bridge the gap between business and IT. Notice that meaning rather than complexity is the main point here: complex interactions can be constructed in current SOAs, but the essential “business” meanings of the interactions remain hidden.

*This research was partially supported by the US National Science Foundation under grant DST-0139037.
The main contribution of a commitment-based SOA is that it provides abstractions that better capture the true nature of business services by directly modeling how the various parties relate to one another. A simple purchase scenario where a customer and a merchant exchange specified goods at a specified price helps illustrate its benefits below.

**Specification** Commitment-based specifications can explicitly reflect user requirements. Upon placing an order, the customer becomes conditionally committed to the merchant to paying for the goods if they are delivered. The delivery of the goods commits the customer to paying the price. When the customer pays, this commitment is discharged.

**Composition** Commitments simplify composing service engagements. For example, we can reuse the above in specifying a more complex service engagement in which independent delivery and payment services are employed.

**Enactment** Specific interactions that occur at runtime may be flexibly determined based on the commitments. For example, if the customer elects to paying first, no harm is done.

**Context** Relevant external considerations can be readily captured in terms of how they affect commitments. For example, the Uniform Commercial Code dictates the conditions under which a customer need not pay for goods he bought (for instance, if the goods arrive damaged and the customer returns them immediately) [9]. In general, context is crucial in handling exceptions, which are rife in business settings.

**Compliance** The correctness of behaviors can be judged at the business level. The above cases illustrate how flexible interactions are allowed and can be judged sound. But not everything goes. Limits to flexibility are necessary: for example, a customer would be in violation if he fails to pay but keeps the goods.

Notice that the above points refer to *business-level* specification, composition, enactment, context, and compliance. These benefits of a commitment-based SOA become more prominent as more complex business scenarios are considered.

### Understanding Service Engagements via Commitments

Our interest in commitments is motivated by their central role in understanding service agreements or contracts. Computationally, each participant in a service engagement is modeled as an *agent*—it is autonomous and can enter into commitments. The agents carry out a service engagement by suitably creating and manipulating commitments to one another. A commitment relates three parties: a *debtor* who is committed to a *creditor* typically within the scope of a *context* (explained below). When a commitment is in force, it has the import of an obligation directed from the debtor to the creditor. However, commitments have two key distinctions from obligations.

**Manipulation** Certain operations can be performed on a commitment. A commitment may be *created* or *discharged*. 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.
Context A commitment occurs within a scope. In essence, the scope provides the legal context for the commitment. The scope may correspond to a transaction, a virtual organization, or a legal jurisdiction, and is also modeled as an agent. The operations on commitments must respect the constraints and policies of the context within which they hold. The above UCC example is a case in point.

The life cycle of a commitment ranges over four key states: null when it does not exist, active when it is fully in force (directly being acted upon), pending when it is in force but not directly being acted upon, and satisfied when it has been discharged.

Representing commitments enables us to express reusable patterns corresponding to a variety of business scenarios. These patterns are expressed in a statechart-like notation involving one or more commitments represented as composite concurrent states. The commitment operations corresponding to a transition would be realized via business actions such as sending purchase orders, delivering goods, and so on, thus achieving different business scenarios. What follows is a high-level categorization of such patterns, along with some illustrative patterns of each category.

Transactional Patterns

The core of a service engagement concerns 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 undoing, redoing, or compensating various actions in light of the various commitments.

![Figure 1: Basic transactional patterns.](image)

Notationally, $C(x, y, G, p)$ represents that debtor $x$ is committed to creditor $y$ in context $G$ to bring about condition $p$. Further, commitments may be conditional. A conditional commitment $CC(x, y, G, q, p)$ means that if the precondition $q$ is brought about, then $C(x, y, G, p)$ is created. This is a common path by which commitments can be created. When $p$ is brought about, the commitments and conditional commitments having $p$ as their condition are discharged. Note that conditional commitments may be discharged directly if the condition is brought about before the precondition. Whereas create, delegate, assign, cancel, and release are caused via actions of
agents, discharge is caused automatically when the commitment conditions are brought about. Additional operations such as update, compensate, and escalate are modeled in terms of the above operations.

- Figure 1(a) shows a commitment being created and discharged. No other operations are allowed. If the commitment is $C(c, m, G_1, \text{pay})$ ($c$ denoting the customer and $m$ the merchant), the merchant sends the goods to the customer, thereby creating the customer’s commitment to pay. The customer sends the payment thereby discharging its commitment.

- Figure 1(b) introduces the cancellation of a commitment. After the merchant sends the goods, the customer may return the goods and cancel its commitment to pay. Thus, cancel is associated with returning the goods.

- Figure 1(c) allows reactivating a satisfied commitment in case the action that caused the satisfaction of the commitment is undone. If a customer invalidates its payment by voiding the check (or in some other manner), it (again) becomes committed to paying. Thus, the second create is associated with the invalidation of payment.

- Figure 1(d) shows a pattern that allows the creditor to release the debtor from the commitment. If the merchant discovers that the shipped goods did not reach the customer, then it may invalidate the order altogether and release the customer from the commitment to pay.

Figure 2: Advanced transactional patterns. In (a), in effect, an existing commitment is canceled and replaced by a commitment involving the same parties but with a different condition. In (b), after the commitment $C(x, y, G, p)$ is discharged (on the left), the debtor $x$ may create another commitment $C(y, x, G, q)$ (on the right) for compensating the effect of $C(x, y, G, p)$.

Figure 2 shows two advanced patterns. A commitment may sometimes need to be updated, as when the customer asks for a hardcover book instead of a paperback, or the merchant refuses a check—perhaps due to a reduction in the customer’s credit rating.

Further, business transactions typically involve multiple interrelated commitments. Let the left commitment in Figure 2(b) be $C(c, m, T_1, \text{pay})$ and the right commitment be $C(m, c, T_1, \text{goods})$.
(the current transaction $T_1$ is the context). When the customer sends the payment the left commitment is discharged. Later, if the customer discovers the goods to be damaged, it may return the goods within the allowed period. Doing so activates the merchant’s commitment (on the right) to send the goods again. A create transition connecting states of the two commitments captures this relationship. In general, each of the individual commitments involved may follow any of the patterns of Figure 1; the patterns of Figure 2 address how the commitments are related.

**Structural Patterns**

In addition to transactions, service engagements involve subtle relationships among the parties involved. The structural patterns capture such relationships in terms that are not directly involved in a transaction. Specifically, the structural patterns may capture constraints on which party may play which role, or whether a party may delegate or assign certain commitments to another party.

Structural patterns can be most simply illustrated when a service engagement involves an organization with internal structure. A participating organization may delegate its commitments under the engagement to appropriate members (who could themselves be organizations). For example, auto insurance companies often delegate their customer service commitments to a local branch, which might further delegate the commitments to a specific staff member. As shown in Figure 3, the structural patterns for delegate vary depending on how the responsibility for a delegated commitment is treated.

- Delegation may occur with or without an associated change in responsibility. In its simplest form (Figure 3(a)), the original commitment is considered satisfied, and a commitment with the delegatee as the debtor is activated. For example, 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 a more complex form (Figure 3(b)), the original commitment becomes pending when the delegated commitment is activated. For example, 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.

- The more complex form supports additional subtleties, as in Figure 3(c). Canceling or releasing a pending commitment nullifies not only it but also the commitments delegated from it. Discharging delegated commitments discharges upper pending commitments as well. Failure to discharge a lower commitment can be framed as an escalate, which not only cancels the lower commitment but also reactivates the upper commitment. An example is when an airline with an over-booked 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.

In organizations, a service engagement often involves division of labor among members, as in Figure 4. Division of labor in situations with responsibility transfer is similar. Just as the debtor can delegate, the creditor can assign commitments. Patterns of assignment are similar to those of delegation.
Figure 3: The debtor $x$ delegates to $z$ its commitment to $y$. As a result, $z$ becomes committed for $p$ (the condition, creditor, and context are unchanged). When $z$ eventually brings about $p$, commitments of both $x$ and $z$ are discharged. In (a), $x$ transfers both the commitment and the associated responsibility to $z$. In (b), $x$ retains the responsibility for satisfying the commitment. In (c), which builds on (b), the subtle effects of cancellation are shown.

**Contextual Patterns**

The business context of a service engagement dictates the rules of encounter to which it is subject. For example, eBay users are subject to the terms and conditions of the eBay marketplace, such as that users may not attempt to shill the price of an item (e.g., by placing false bids). More pertinently, the rules for dispute resolution are also contextual in nature.

Contexts may nest or overlap in subtle ways. For example, eBay interactions in the US fall under the purview of the Federal Trade Commission, but those in France do not. Thus, eBay users in France may not trade in Nazi memorabilia, which is illegal in that country. International service engagements are often complex because they cross jurisdictions.

Whereas all commitments involve a context, in the contextual patterns, the context agent itself features as a debtor or creditor. For example, in Figure 5, the context commits to the creditor
The classical principle of division of labor is realized when the delegator delegates parts of a complex condition to different delegates. The individual commitments are left blank to indicate that they may follow any suitable pattern.

of $C(x, y, G, p)$ that if $C(x, y, G, p)$ is canceled, $C(x, y, G, q)$ will be created. For example, if $p$ means pay $10 by Monday, $q$ could mean pay $11 by Tuesday. Or, if $p$ means deliver the good, $q$ could mean refund the amount deposited.

The context has the power to create and manipulate commitments among the agents in its scope. The above shows how a contextual pattern dominates a transactional pattern. Contextual patterns can dominate a structural pattern as well. For example, it may sometimes be illegal to delegate a commitment without prior consent from the creditor. A typical interorganizational engagement would operate under a context instantiating multiple patterns.

**Applying the Patterns**

The above patterns merely enumerate abstract possibilities. To apply them successfully means matching them to the concrete realities of a service engagement. For example, a transactional
pattern allowing cancellation would make sense only if a commitment can be reasonably canceled. And, it may not be possible to delegate a commitment if the intended delegatee would not accept becoming a debtor of the specified commitment. Lastly, the context may not be able to ensure that an agent will discharge any commitments created by the context. In general, the above patterns would work best when there is a prior business or legal relationship among the parties involved. The desired patterns can guide the creation of the appropriate relationships of constraints so that desired service engagements are realized.

Commitment patterns would generally be applied in combination, potentially involving several from each of the above three categories. Thus a service engagement pattern would combine multiple commitment patterns. As an illustration, a value network consists of several interacting service providers and consumers. For example, an auto insurance company commits to insure its policy holders. The insurance company may task a mechanic for performing necessary repairs. The customer would deal with the mechanic directly to carry out the main repair transaction. The mechanic’s initial commitment is by delegation from the insurance company, based on the mechanic’s contract with the insurance company. The mechanic’s quality guarantee (e.g., limited 30-day warranty on new parts) would be a commitment based on the repair transaction, and inherited from local laws à la UCC.

Case Study: TWIST Foreign Exchange Interactions

The benefits of a commitment-based SOA have been demonstrated through improved modeling and enactment of foreign exchange (FX) business processes [2]. FX markets involve thousands of autonomous players. Plus they are huge—with an average daily volume of $2.3 trillion, and increasing [10]. Thus FX processes are of immense importance.

We studied processes for wholesale trade as specified by TWIST, the Transaction Workflow Innovation Standards Team, a nonprofit industry group [8]. The TWIST specification (like its predecessor standards) describes FX processes informally via natural language descriptions accompanied by sequence diagrams of typical scenarios. Thus the business meaning, which is at the heart of these
processes, is expressed only informally.

The resulting ambiguity complicates interoperation, limits flexibility, and reduces productivity. Moreover, like many SOA applications, FX processes exhibit several dimensions of variation, such as trading with or without credit checks and with or without a trading service. Current, informal approaches attempt to capture several variations explicitly, thus complicating the specifications and resulting implementations. But when modeled formally via commitments, the variations can be captured succinctly and independently, and composed as needed to produce the desired processes.

The above observations are based on a recent exercise in reengineering TWIST wholesale trade processes [2]. Specifically, we (1) discovered and corrected ambiguities in the original specification; (2) modeled the basic FX variations modularly as patterns; and (3) showed that 12 patterns not only capture the 28 TWIST wholesale trade scenarios, but also allow legitimate variations beyond the 28 originally modeled.

SOAs as Architectural Styles

An architectural style specifies a family of configurations of components and connectors subject to stated constraints [7]. An architectural style thus serves as a formal template for the family of systems being described.

In these terms, an SOA is an architectural style whose major components are (service) provider and consumer, and whose connector is a protocol by which a consumer invokes a service at a provider. (For simplicity, this paper ignores registries, and service publication and discovery.) A practical SOA includes specialized components and connectors, such as for resource management and other enterprise functions (identity, billing, and so on), and imposes additional constraints so appropriate components interoperate with the management and enterprise components.

Current SOAs address process management in two main ways [6]. Orchestration—epitomized by BPEL4WS (Business Process Execution Language for Web Services)—specifies a central program that manages the control and data flow among a set of services, by invoking operations or sending messages. Choreography—epitomized by CDL (Choreography Description Language)—specifies how messages are exchanged in a distributed manner among a set of services.

Benatallah et al. propose patterns called business-level interfaces and protocols [1]. However, their approach is purely syntactic. Like in CDL and BPEL, their patterns ignore business meanings, thereby leading 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. That is, it ought not to take any liberties such as reversing the order of the messages, interposing other messages, or introducing another party such as a payment agency. But, real-life service engagements typically presume such flexibility. Limiting flexibility subverts the services vision because it creates avoidable friction in the web of value.

By contrast, our commitment-based SOA (CSOA) is an architectural style whose components are business service provider and consumer agents and whose connectors are commitment patterns. Thus, CSOA is not a unique style but has many flavors depending on the patterns selected. Such flexibility is necessary to support various nuances of service engagements. The primary CSOA constraint is that all commitments must eventually be null or satisfied.

The fundamental CSOA style gives rise to several important styles characterized by service engagement patterns that result from combinations of transactional, structural, and contextual patterns.
Table 1: A comparison of commitment-based and existing SOAs with respect to five main aspects of an architectural style, as proposed by Garlan and Shaw. Here a (computational) model and (characteristic) patterns are added to the first three aspects.

<table>
<thead>
<tr>
<th>Aspects</th>
<th>Existing SOAs</th>
<th>Commitment-based SOA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Components</td>
<td>Service provider and consumer</td>
<td>Business service provider and consumer agents</td>
</tr>
<tr>
<td>Connectors</td>
<td>Operations and message patterns (in, out, in-out, out-in)</td>
<td>Commitment patterns</td>
</tr>
<tr>
<td>Constraints</td>
<td>Match operation and message signatures</td>
<td>Eventually satisfy or nullify commitments</td>
</tr>
<tr>
<td>Model</td>
<td>Control and data flow</td>
<td>Operations on commitments</td>
</tr>
<tr>
<td>Patterns</td>
<td>Orchestration and choreography patterns</td>
<td>Service engagement patterns</td>
</tr>
</tbody>
</table>

For example, subcontracting and value network styles may reflect the same transactional, but different structural patterns. Although the possible patterns depend upon the desired service engagements, the vocabulary of commitments bounds the possibilities. Table 1 contrasts commitment-based and existing SOAs.

Kumaran [4] presents four abstraction layers for enterprise modeling: strategy (business executive-level considerations), operation (business functions conceptualized in terms of 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 operations more perspicuously and reusably.

The foregoing makes the case for a fresh look at service orientation from the perspective of business services. It develops the elements of a new architectural style that differs crucially from existing SOAs. The reader may reasonably wonder: if it is so different why is it still an SOA? The answer is twofold. One, the CSOA is centered on services and is, arguably, more true to the services vision than existing SOAs. Two, CSOA doesn’t seek to replace existing SOAs and their implementations. Although CSOA is conceptualized in terms of business service engagements, it is meant to be realized through conventional means. Model-Driven Architecture (MDA) [5] provides a useful way to think of the relationship between commitment-based 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.

Tools and methodologies enable the transformation from CIMs to PIMs. Prototype tools and methodologies exist that map commitment patterns and computations [3]. Important directions are to specify and accommodate service engagement patterns as introduced above.

References


Author Biographies

**Munindar P. Singh** is a Professor of Computer Science at NCSU. He is a former EIC of *IEEE Internet Computing*. Singh coauthored the textbook *Service-Oriented Computing: Semantics, Processes, Agents*.

**Amit K. Chopra** is a doctoral candidate in Computer Science at NCSU. Chopra’s interests include service-oriented architectures and multiagent systems.

**Nirmit Desai** is a doctoral candidate in Computer Science at NCSU. Desai’s interests include cross-organizational business processes.