Business Process Adaptations via Protocols [*]

Nirmit Desai Amit K. Chopra Munindar P. Singh
{nvdesai, akchopra, singh}@ncsu.edu
Department of Computer Science
North Carolina State University

Abstract

Business process management in service-oriented computing (SOC) environments poses special challenges. In particular, SOC environments are dynamic, thereby requiring frequent changes in business processes. Current business process modeling approaches handle such changes in an ad hoc manner, and lack a principled means of determining what needs to be changed and where.

This paper addresses process adaptability through a novel application of business protocols, especially of protocol composition, introduced in our previous work. Through a real business scenario of auto-insurance claim processing, this paper demonstrates how a wide range of adaptations can be handled naturally and systematically via protocol composition. The illustrated adaptations have been evaluated via a prototype.

1. Introduction

Successful business process management requires supporting three key properties of service-oriented computing (SOC) environments, namely, *autonomy*, *heterogeneity*, and *dynamism* [17, pp. 7–10]. Supporting autonomy means modeling and enacting business processes in a manner that offers maximal flexibility to the participants by only minimally constraining their behavior. Supporting heterogeneity means making as few assumptions as possible about the construction of the components for the parties, concentrating instead on characterizing the interactions among them. For SOC environments, interconnections among components, not the components themselves, are critical.

Dynamism has two flavors. *Membership dynamism* means that the set of components may change. Components may be added and removed at either design-time (*design-time membership dynamism*) or run-time (*run-time membership dynamism*). *Structural dynamism* means that the set of interconnections is not fixed. Dynamism reflects changes in the business requirements; modern businesses face intense pressure and must repeatedly reconfigure themselves in order to thrive, if not to survive. As changes in requirements are routine, an elegant way of handling such changes is vital. Thus, dynamism poses a difficult challenge, one that has not been adequately addressed in the literature, and is the theme of this paper. Supporting dynamism implies supporting process adaptability.

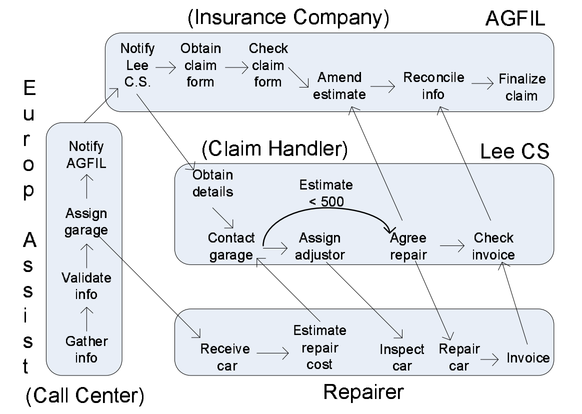

Figure 1. CrossFlow Insurance Claim Processing

Previously, Desai et al. [6] reported a methodology (named OWL-P) for developing business processes that supports autonomy and heterogeneity. The essence of OWL-P is an abstraction of business protocols. A *business protocol* is an abstract, modular, and publishable specification of rules that govern a business interaction among two or more *roles*. A protocol is abstract because it does not model the proprietary business logic of the agents enacting the roles. It is modular because it achieves a specific goal which may be a part of another bigger goal. Due to their abstract and modular na-

[*] This research was partially supported by the NSF under grant DST-0139037 and by DARPA under contract F30603-00-C-0178.

ture, business protocols, each achieving a goal, can be combined to form a *composite protocol* that achieves a bigger goal. Each role's perspective of a protocol is captured in its *role skeleton*. A role skeleton, when integrated with the private business logic of an agent enacting the role, yields the agent's *local process*. A *business process* is an aggregation of the local processes of all the agents involved.

In this way, protocols partition a process in a crosscutting manner whereas conventionally, flows are taken as partitions. For instance, peeking ahead on an example studied in great detail below, each of the boxes in Figure 1 represents a flow. Such boxes would be the modules of abstraction in conventional approaches. By contrast, OWL-P focuses on the interconnections among the boxes; such interconnections are a natural fit for the interorganizational nature of SOC environments [6]. As a striking illustration of the importance of focusing on the interconnections, notice that the process flow (taken from [4]) in Figure 1 misses the insurance holder and its interactions altogether. Although the internal flow of the insurance holder's process (its box) may not be important to AGFIL, its external interactions with other parties (i.e., the insurance holder's interconnections) are crucial for modeling the business of AGFIL. Figure 2 shows how the above example is modeled via protocols.

Previous work on protocols provides the basic mechanisms to specify, compose, and enact them to produce software engineering benefits such as reusability and flexibility. This paper enhances the treatment of protocols so as to support dynamism in a real business scenario. Adaptability in general involves a broad variety of research challenges. Hence, this paper concentrates on three kinds of adaptations: (1) exceptions, (2) changes in business policies, and (3) changes in business models.

To demonstrate and evaluate the protocol-based approach, this paper considers the above insurance claim processing case studied under the CrossFlow project [4]. Being a real-life business scenario previously studied by others, this example provides an independent and significant test-case for this approach. Figure 1 shows the parties involved in the process and the process flow. AGFIL is an insurance company in Ireland. The present scenario deals with automobile insurance. AGFIL underwrites the insurance policy and covers any losses incurred. Europ Assist provides a 24-hour help-line service for receiving claims. Lee CS is a consulting firm that coordinates with AGFIL and deals with repairers to handle the claims. A network of approved repairers provide repair services. AGFIL holds ultimate control in deciding if a given claim is valid and if payment will be made to the repairer.

Organization. Section 2 introduces protocols and their composition. Section 3 is the essence of this paper. It shows how protocol composition handles a variety of changes at different levels of the business process. Section 4 describes prototype tools, enactment software, and a usage scenario. Finally, Section 5 discusses related work and outlines some ideas for future research.

2. Protocols

This section presents the important concepts of OWL-P and illustrates them with the protocols extracted from the CrossFlow example.

2.1. Commitments

Commitments are used to give semantics to agent interaction. As agents interact, they create and manipulate commitments. A commitment $\mathsf{C}(x, y, p)$ denotes that agent x is obligated to agent y for bringing about condition p. Commitments can be *conditional*, denoted by $\mathsf{CC}(x, y, p, q)$, meaning that x is committed to y to bring about q if p holds where, p is called the precondition of the commitment. Commitments are created, satisfied, and transformed in certain ways. The following are the operations defined on commitments:

Op1. $\text{CREATE}(x, c)$ establishes commitment c.

Op2. $\text{CANCEL}(x, c)$ cancels the commitment c.

Op3. $\text{RELEASE}(y, c)$ releases c's debtor from the commitment c.

Op4. $\text{ASSIGN}(y, z, c)$ replaces y with z as c's creditor.

Op5. $\text{DELEGATE}(x, z, c)$ replaces x with z as c's debtor.

Op6. $\text{DISCHARGE}(x, c)$ (c's debtor x) fulfills the commitment.

The rules regarding discharge of a commitment are given below.

Dis1. $\frac{C(x,y,p) \wedge p}{discharge(x, C(x,y,p))}$

Dis2. $\frac{CC(x,y,p,q) \wedge p}{create(x, C(x,y,q)) \wedge discharge(x, CC(x,y,p,q))}$

Dis3. $\frac{CC(x,y,p,q) \wedge q}{discharge(x, CC(x,y,p,q))}$

2.2. Specifying a Protocol

OWL-P, which is based on OWL (Web Ontology Language), is an ontology for specifying protocols. A plugin for the Protégé ontology editor [16] facilitates specifying and publishing protocols to a repository.

A *Protocol* specifies a set of rules that govern the interaction among *roles* from a global view-point. OWL-P employs the Semantic Web Rule Language (SWRL) [10] to specify rules. Each rule is a SWRL implication where the

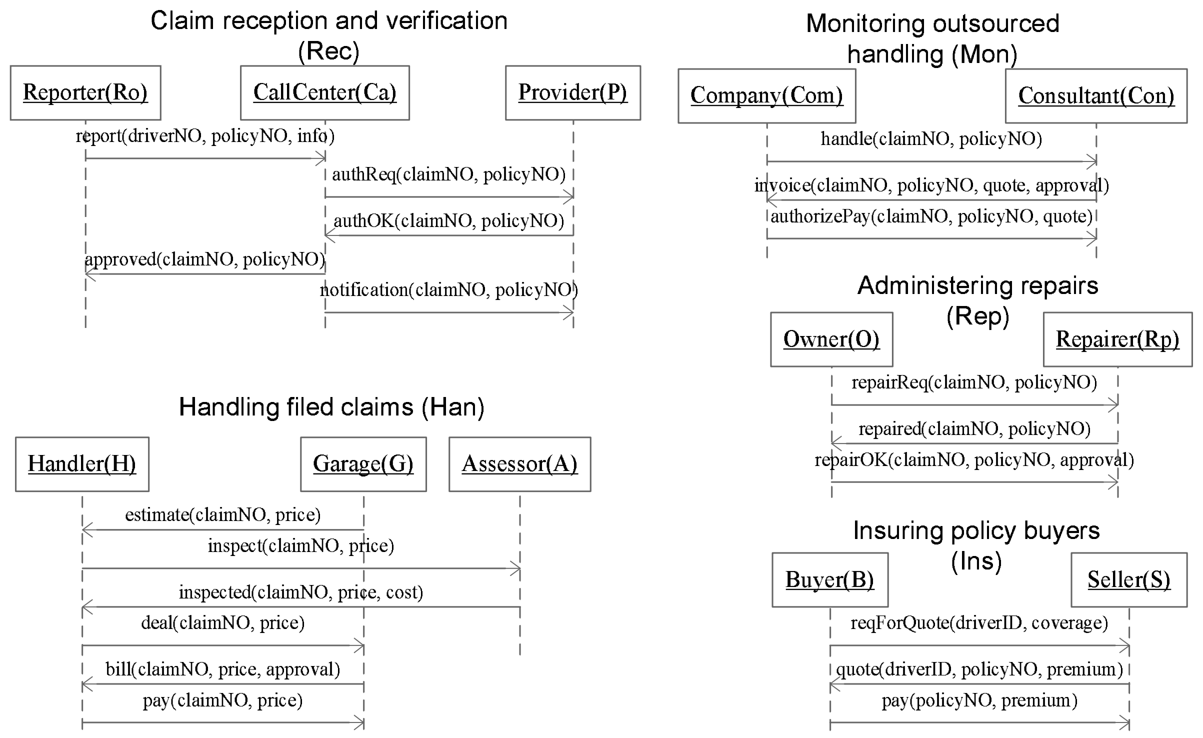
Claim reception and verification (Rec)
Reporter(Ro)
CallCenter(Ca)
Provider(P)
report(driverNO, policyNO, info)
authReq(claimNO, policyNO)
authOK(claimNO, policyNO)
approved(claimNO, policyNO)
notification(claimNO, policyNO)
Monitoring outsourced handling (Mon)
Company(Com)
Consultant(Con)
handle(claimNO, policyNO)
invoice(claimNO, policyNO, quote, approval)
authorizePay(claimNO, policyNO, quote)
Administering repairs (Rep)
Owner(O)
Repairer(Rp)
repairReq(claimNO, policyNO)
repaired(claimNO, policyNO)
repairOK(claimNO, policyNO, approval)
Handling filed claims (Han)
Handler(H)
Garage(G)
Assessor(A)
estimate(claimNO, price)
inspect(claimNO, price)
inspected(claimNO, price, cost)
deal(claimNO, price)
bill(claimNO, price, approval)
pay(claimNO, price)
Insuring policy buyers (Ins)
Buyer(B)
Seller(S)
reqForQuote(driverID, coverage)
quote(driverID, policyNO, premium)
pay(policyNO, premium)

Figure 2. Scenarios from the protocols corresponding to the insurance claim process of Figure 1

body (antecedent) specifies a set of *propositions* that must hold for the rule to fire, and the head (consequent) specifies operations such as sending messages, commitment operations, asserting propositions, and invoking business logic. A proposition may represent a *message* that has been sent or received, active commitments, or domain specific facts. Messages and propositions have *slots* which are analogous to parameters. The operational semantics of messages characterizes their effects on the commitments. For example, a message may create, discharge, delegate, assign, or cancel a commitment.

Figure 2 shows scenarios from the protocols involved in the above insurance example. The following discussion provides specifications of the protocols as needed. Appendices provide the specifications of *Rep* and *Han*. A \cdot (·) marks a parameter that is not relevat. Usually, the protocols are designed and maintained independently. Therefore, the ontologies of the protocols about similar concepts would differ. For example, driverID in *Ins* and driverNO in *Rec* refer to the same piece of information.

The *buyer* and the *seller* roles engage in the insurance buying protocol (*Ins*) for insuring the *buyer*. The rules for *Ins* are:

Ins1. start ⇒ reqForQuote(driverID, coverage)

Ins2. reqForQuote(driverID, coverage) ⇒ quote(driverID, policyNO, premium) ∧ CC(S, B, subscribe, insurance)

Ins3. pay(policyNO, premium) ∧ quote(·, policyNO, premium) ⇒ subscribe

Ins4. C(S, B, insurance) ⇒ CC(S, B, serviceReq ∧ validClaim, claimService) ∧ CC(S, B, reqForClaim, claimResponse)

Ins5. quote(·, policyNO, premium) ⇒ pay(policyNO, premium)

Rule Ins1 means that at the start, a request for an insurance quote is allowed. Rule Ins2 means that if a request for quote has happened, a quote for the requested coverage is allowed and the meaning of quote is given via the stated conditional commitment, that is, *seller* is committed to insuring the *buyer* if the *buyer* subscribes to the insurance policy. The meanings of subscribing and insuring are given by Rules Ins3 and Ins4, respectively. The *buyer* subscribes to the insurance by paying the quoted premium. To provide the insurance service, the *seller* must serve the *buyer*'s requests for handling the claim if the claim is valid and the *seller* must respond to the claims filed by the *buyer*. As long as the *buyer* is insured, multiple claims can be filed and have to be served, until the policy expires and insurance is asserted discharging the insurance commitment. Rule Ins5 states that the *buyer* can pay after quote has happened. Here, reqForQuote and quote in the body are propositions representing messages reqForQuote and quote in the head, respectively; driverID, coverage, policyNO, and so on are slots; subscribe, claimService, and so on are domain specific propositions.

2.3. Composing Protocols

Any realistic business process would involve multiple interaction protocols. For example, AGFIL would need protocols for selling insurance policies and for receiving and handling claims. Consider how protocols can be combined to form a composite protocol. (Section 3.3 shows how this example also illustrates a change in business model.) Let's assume AGFIL outsources the help-line service for receiving claim reports to a call center, but handles the claims itself. The requisite process can be obtained by combining the *Ins* and *Rec* protocols in a certain way yielding a new *Bas* (Basic insurance claim processing) composite protocol. Here is the specification of the *Rec* protocol:

Rec1. start $\wedge$ CC(Ca, Ro, reqForClaim, claimResponse) $\wedge$ CC(P, Ca, reqAuth, authResponse) $\Rightarrow$ report(driverNO, policyNO, info)
Rec2. report($\cdot$, policyNO, $\cdot$) $\Rightarrow$ authReq(claimNO, policyNO)
Rec3. authReq(claimNO, $\cdot$) $\Rightarrow$ authOK(claimNO, $\cdot$)
Rec4. authOK(claimNO, $\cdot$) $\Rightarrow$ approved(claimNO, $\cdot$)
Rec5. authReq(claimNO, $\cdot$) $\Rightarrow$ authNOK(claimNO, $\cdot$)
Rec6. authNOK(claimNO, $\cdot$) $\Rightarrow$ denied(claimNO, $\cdot$)
Rec7. approved(claimNO, $\cdot$) $\Rightarrow$ notification(claimNO, $\cdot$)
Rec8. authReq(claimNO, $\cdot$) $\Rightarrow$ reqAuth
Rec9. authReq(claimNO, $\cdot$) $\wedge$ authOK(claimNO, $\cdot$) $\Rightarrow$ authResponse
Rec10. authReq(claimNO, $\cdot$) $\wedge$ authNOK(claimNO, $\cdot$) $\Rightarrow$ authResponse
Rec11. approved(claimNO, policyNO) $\wedge$ report($\cdot$, policyNO, $\cdot$) $\Rightarrow$ claimResponse
Rec12. denied(claimNO, policyNO) $\wedge$ report($\cdot$, policyNO, $\cdot$) $\Rightarrow$ claimResponse
Rec13. report($\cdot$, policyNO, $\cdot$) $\Rightarrow$ reqForClaim

Typically, the protocols are enacted with some of the roles having business relationships established via participating in some other protocol, e.g., *Ins* establishes the contract between the *buyer* and the *seller*. This business relationship is a contract represented by a set of commitments. For brevity, this presentation skips such relationship formation protocols. Instead, it models the start of these protocols as conditioned on the corresponding commitments. For example, in *Rec*, the *provider* is committed to the *callCenter* to respond to requests for authorizing claims (the second commitment in Rec1). Also, the *provider* would have delegated the commitment CC(S, B, reqForClaim, claimResponse) to the *callCenter* (the first commitment in Rec1). As a comprehension aid, Figure 3 shows all such contractual commitments among the roles after all the relationships are formed and commitments delegated.

As the ontologies of the protocols may not match, and they may have interdependencies, simply unioning the rules of the component protocols is not enough. The constraints on combining protocols are specified as a set of *composition axioms* in a composition profile. Below are the set of composition axioms for the insurance example:

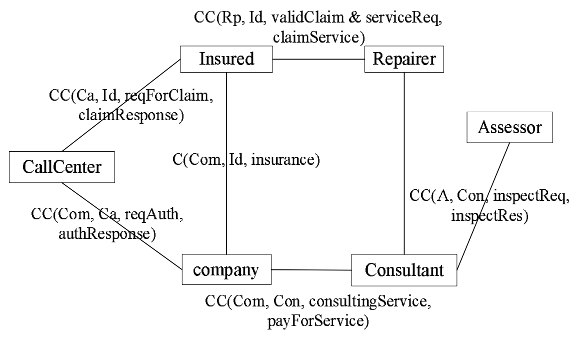
CC(Rp, Id, validClaim & serviceReq, claimService)
Insured
Repairer
CC(Ca, Id, reqForClaim, claimResponse)
Assessor
CallCenter
C(Com, Id, insurance)
CC(A, Con, inspectReq, inspectRes)
CC(Com, Ca, reqAuth, authResponse)
company
Consultant
CC(Com, Con, consultingService, payForService)

Figure 3. Commitments among parties

BasAx1. Bas.Insured $\doteq$ Ins.Buyer, Rec.Reporter
BasAx2. Bas.Insurer $\doteq$ Ins.Seller, Rec.Provider
BasAx3. Bas.CallCenter $\doteq$ Rec.CallCenter
BasAx4. Ins3.driverID $\rightsquigarrow$ Rec1.driverNO
BasAx5. Ins3.policyNO $\rightsquigarrow$ Rec1.policyNO
BasAx6. Rec.authOK $\rightarrow$ Ins.validClaim

The first three are *RoleDefinition* axioms that define a new role specified on the left by identifying it with the roles on the right. Intuitively, it says that the roles on the right become the role on the left in the composite protocol. In effect, a role definition axiom adds a new role in the composite protocol and replaces the roles on the right by this role (as the debtor or the creditor of commitments and the sender or the receiver of the messages).

Axioms BasAx4 and BasAx5 are *DataFlow* axioms that specify the data flow from the slot on the left to the slot on the right. This is effected by asserting an additional proposition in Ins3 having one slot (driverID), and conjuncting this proposition to the body of Rec1, thus binding driverNO to driverID.

Axiom BasAx6 is an *Implication* axiom that aligns the ontologies of the protocols. For example, authOK in *Rec* means validClaim in *Ins*. This axiom would be effected by adding a rule to the composite protocol to effect the implication from authOK to validClaim.

The rest of the protocol rules are unioned. The resulting protocol *Bas* is the desired behavior. Composite protocols can be treated and enacted like other protocols. This paper does not discuss the enactment of protocols (by integration with the business logic of the agents) and *EventOrder* axioms that allow the specification temporal ordering between the events in the protocols.

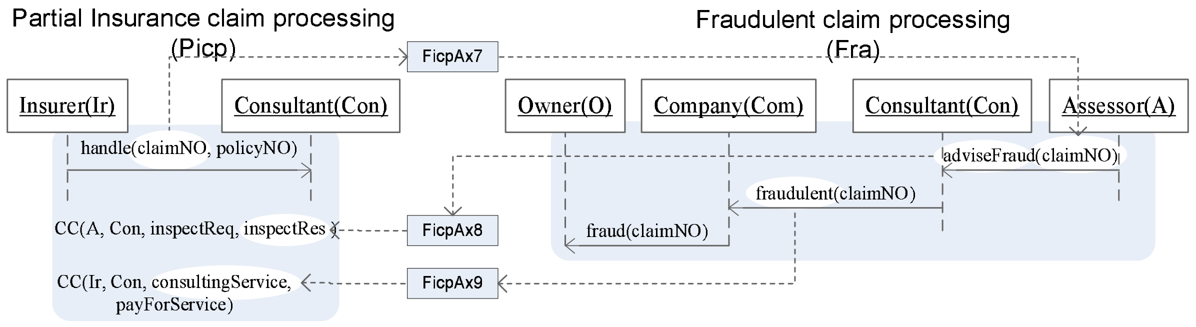
Partial Insurance claim processing
(Picp)
Fraudulent claim processing
(Fra)
FicpAx7
Insurer(Ir)
Consultant(Con)
Owner(O)
Company(Com)
Consultant(Con)
Assessor(A)
handle(claimNO, policyNO)
adviseFraud(claimNO)
fraudulent(claimNO)
CC(A, Con, inspectReq, inspectRes)
FicpAx8
fraud(claimNO)
CC(Ir, Con, consultingService, payForService)
FicpAx9

Figure 4. Handling fraudulent claims

3. Adapting Processes

Through a detailed study of the AGFIL insurance process, this section shows how OWL-P naturally allows process reengineering through adaptation. The core idea is to enhance and apply the basic mechanism of protocol composition in a variety of new ways. Each of the adaptations presented below feature varying degrees of autonomy, heterogeneity, structural dynamism, and design-time membership dynamism. The following assumes that a composite protocol *Picp* (Partial insurance claim processing) is constructed from the composition of *Bas*, *Mon*, *Han*, and *Rep* of Figure 2.

3.1. Exception Handling

Exceptions are (typically) uncommon conditions arising during a business interaction, e.g., fraudulent auto-insurance claims. Protocols facilitate incorporating the handling of an exception into an existing process. A protocol for handling frauds can be specified as follows:

Fra1. start $\wedge$ C(Com, O, insurance) $\Rightarrow$ adviseFraud(claimNO)
Fra2. adviseFraud(claimNO) $\Rightarrow$ fraudulent(claimNO) $\wedge$ release(Con, CC(Rp, Con, acceptEstimate, performRepair))
Fra3. fraudulent(claimNO) $\Rightarrow$ fraud(claimNO) $\wedge$ cancel(Com, C(Com, O, insurance))

A composite protocol *Ficp* (Fraudulent insurance claim processing) can be constructed by combining *Picp* with *Fra* (Fraudulent claims) as shown in Figure 4 (*insured* (Id), *repairer* (Rp), *callCenter* (Ca), and *assessor* (A) are not shown for *Picp*). When AGFIL and Lee CS form their relationship, a conditional commitment CC(Com, Con, consultingService, payForService) is created meaning that AGFIL will authorize payments for handling individual claims if Lee CS provides the consulting service. In case of fraud, the *company* cancels the policy coverage of the *insured* and the *consultant* releases the *repairer* from the commitment to perform repairs (Appendix A). The *assessor*'s advising of a fraud should discharge CC(A, Con, inspectReq, inspectRes) (Appendix B) and the *consultant*'s forwarding it to the *company* should mean the consulting service was provided. These conditions are modeled by the composition axioms here:

FicpAx1. Ficp.Insured $\doteq$ Picp.Insured, Fra.Owner
FicpAx2. Ficp.Insurer $\doteq$ Picp.Insurer, Fra.Company
FicpAx3. Ficp.Consultant $\doteq$ Picp.Consultant, Fra.Consultant
FicpAx4. Ficp.Repairer $\doteq$ Picp.Repairer
FicpAx5. Ficp.CallCenter $\doteq$ Picp.CallCenter
FicpAx6. Ficp.Assessor $\doteq$ Picp.Assessor, Fra.Assessor
FicpAx7. Picp_claimNO $\rightsquigarrow$ Fra1.claimNO
FicpAx8. Fra.adviseFraud $\rightarrow$ Picp.inspectRes
FicpAx9. Fra.fraudulent $\rightarrow$ Picp.consultingService

Notice how the resulting interaction allows AGFIL to end the process by canceling the commitments in case of a fraud. Simply adjusting the commitments can allow greater flexibility without affecting other parties adversely. Here, the support for structural dynamism comes from the added interconnections among the existing partners to handle the exceptions. Support for autonomy comes from the ability to control the effects of foul behavior of a participant.

3.2. Changes in Business Policy

In handling claims where the value of the car is less than the estimated cost of repairs, the *company* may want to scrap the car, i.e., declare it a total loss. To settle this, it pays the *insured* a sum equal to the value of the car and takes possession of the car instead of administering repairs. Also, if the damage is minor, the *insured* may accept a cash settlement instead of having the car repaired. Here is the specification of a protocol *Pcsc* (shown in Figure 5) that specifies interactions for such a policy:

Pcsc1. start $\Rightarrow$ adviseScrap(claimNO, value) $\wedge$ delegate(Con, Com, CC(Rp, O, serviceReq $\wedge$ claimValid, claimService))
Pcsc2. adviseScrap(claimNO, value) $\Rightarrow$ settle(claimNO, value)

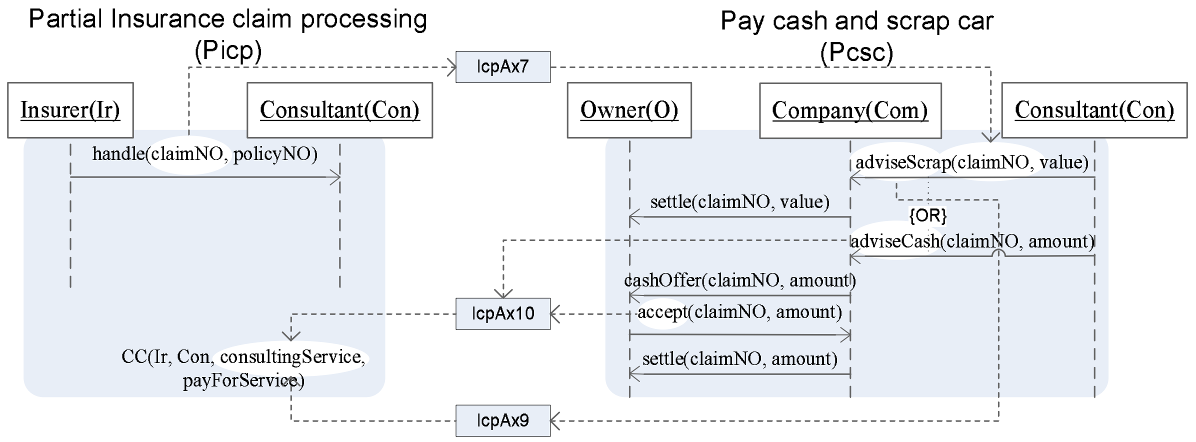
Partial Insurance claim processing (Picp)
Pay cash and scrap car (Pcsc)
IcpAx7
Insurer(Ir)
Consultant(Con)
Owner(O)
Company(Com)
Consultant(Con)
handle(claimNO, policyNO)
adviseScrap(claimNO, value)
settle(claimNO, value)
{OR}
adviseCash(claimNO, amount)
cashOffer(claimNO, amount)
IcpAx10
accept(claimNO, amount)
CC(Ir, Con, consultingService, payForService)
settle(claimNO, amount)
IcpAx9

Figure 5. Change in business policy

Pcsc3. start ⇒ adviseCash(claimNO, amount) ∧ delegate(Con, Com, CC(Rp, O, serviceReq ∧ claimValid, claimService))
Pcsc4. adviseCash(claimNO, amount) ⇒ cashOffer(claimNO, amount) ∧ CC(Com, O, acceptCash, settlement)
Pcsc5. cashOffer(claimNO, amount) ⇒ accept(claimNO, amount)
Pcsc6. accept(claimNO, amount) ∧ cashOffer(claimNO, amount) ⇒ acceptCash
Pcsc7. accept(claimNO, amount) ⇒ settle(claimNO, amount)
Pcsc8. settle(claimNO, amount) ∧ accept(claimNO, amount) ⇒ settlement
Pcsc9. cashOffer(claimNO, amount) ⇒ reject(claimNO, amount)

AGFIL would have delegated its conditional commitment CC(S, B, serviceReq ∧ claimValid, claimService) to Lee CS, thereby making Lee CS responsible for servicing claims, who would in turn delegate it to the *repairer*. When the *consultant* advises scrapping the car, it fulfills its commitment to provide the consulting service and the commitment to service the claim falls back to the *company* as the car is not to be repaired but to be paid for. Similarly, when the *consultant* advises a cash payment for a minor damage, and the *insured* agrees to it, it fulfills the commitment for the consulting service and the *company* becomes committed to paying the *insured*. Rules Pcsc1 and Pcsc3 model these by delegation. Assuming that *Picp* is the current interaction protocol among the partners, *Pcsc* can be composed with *Picp* yielding the composite protocol *Icp* (Insurance claim processing) that handles this policy change via the following axioms:

IcpAx1. Icp.Insured ≐ Picp.Insured, Pcsc.Owner
IcpAx2. Icp.Insurer ≐ Picp.Insurer, Pcsc.Company
IcpAx3. Icp.Consultant ≐ Picp.Consultant, Pcsc.Consultant
IcpAx4. Icp.Repairer ≐ Picp.Repairer
IcpAx5. Icp.CallCenter ≐ Picp.CallCenter
IcpAx6. Icp.Assessor ≐ Picp.Assessor
IcpAx7. Picp_claimNO ⇝ Pcsc1.claimNO
IcpAx8. Picp_claimNO ⇝ Pcsc3.claimNO
IcpAx9. Pcsc.adviseScrap → Picp.consultingService
IcpAx10. Pcsc.adviseCash ∧ Pcsc.accept → Picp.consultingService
IcpAx11. Pcsc.settlement → Picp.claimService
IcpAx12. Pcsc.settle(claimNO, value) ∧ Pcsc.adviseScrap(claimNO, value) → Picp.claimService

Figure 5 shows the composition graphically (*insured* (Id), *repairer* (Rp), *callCenter* (Ca), and *assessor* (A) are not shown for *Picp*). Notice how a business policy change internal to AGFIL is accommodated across the business process. Here, support for structural dynamism comes from the modified interconnections among the existing partners to handle the policy change. Support for heterogeneity comes from the fact that a change of business logic internal to AGFIL could be accommodated.

3.3. Changes in Business Model

Quite often, enterprises change their business models to respond to changes in their business environment. Business process outsourcing (BPO) [8] has recently become a popular approach for conducting business.

The composition of *Bas* discussed in Section 2.3 already illustrated this type of adaptation. Let's assume AGFIL could handle claims itself. AGFIL outsourced its call center to Europ Assist to focus on its core business of selling insurance policies. Thus, the business model of AGFIL changed to have a partnership with Europ Assist.

Obviously, a change in the business model is a big shift for an enterprise; new partnerships are formed and new interconnections among the parties emerge. Thus, such a change features not only structural dynamism, but also design-time membership dynamism. Both are supported via composition with protocols having new roles.

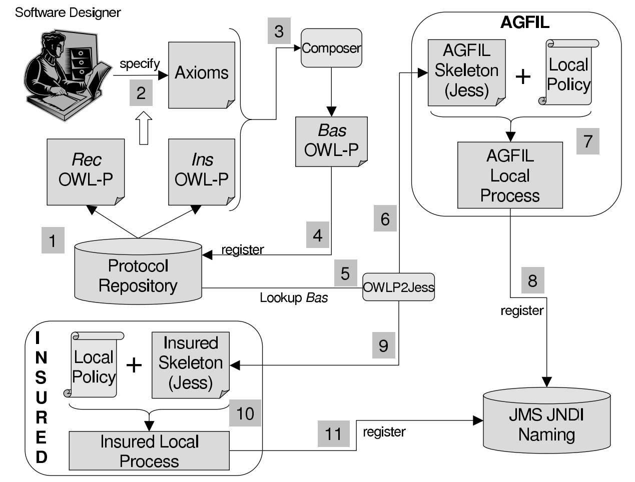

Figure 6. A typical usage scenario

4. Prototype

The above insurance protocols among several others have been modeled in OWL-P [14]. The prototype OWL-P framework, described here, further establishes the practicality of the proposed approach. This framework includes tools with which a designer may compose and adapt protocols.

The *Composer* tool automatically generates a composite protocol from a set of component protocols and a composition profile having composition axioms.

The *OWLP2Jess* tool translates a protocol specification in OWL-P to an equivalent set of Jess rules, which can then be executed by the Jess rule engine [7]. These rules capture the role skeletons that provide the business logic stubs to specify an agent.

A plugin for the Protégé ontology editor enables a designer to specify new protocols and composition profiles, generate role skeletons from protocols, and examine them.

The OWL-P approach uses conventional technologies where possible. In particular, it instantiates a J2EE-based agent that loads the Jess rules, and communicates using a JMS queue in an EJB container (WebSphere Application Server 6.0, in our prototype). OWL-P software, tools, and documentation are publicly available [15]. Figure 6 shows a typical usage scenario for enacting a business process from *Bas*; *CallCenter* agent is not shown.

5. Discussion and Future Work

This paper demonstrates the value of taking an interaction-oriented approach to business process adaptation.

Business processes have been traditionally modeled as workflows. However, workflows do not take interactions into account, and therefore, cannot adequately handle inter-organizational settings in which the autonomy of the participants is crucial [11, 3].

Agents (and therefore interactions) are an important abstraction for modeling autonomous participants. Agent-oriented software methodologies such as Tropos [2], Gaia [19], and others [9] support the modeling of inter-organizational applications such as business processes. In these approaches, however, protocols are designed from first principles usually depending on the organizational structure. As such, they do not exploit protocols as first-class abstractions. The proposed approach complements these methodologies by treating protocols as building blocks that can be composed and adapted.

Service composition has been extensively investigated. Local processes in the present framework correspond to services in SOC terminology. BPEL [1] is a language for specifying the static composition of Web services. However, it mixes the interaction activities with the business logic, making it unsuitable for reuse and systematic adaptations. OWL-S [5], which includes a process model for Web services, uses semantic annotations to facilitate dynamic composition. While dynamic service composition has some advantages, it assumes a perfect markup and an ontological matching of the services being composed. By contrast, the proposed approach does not emphasize dynamic service composition and instead seeks to provide a human designer with tools to facilitate service composition.

Vitteau *et al.* studied composing protocols from microprotocols in a bottom-up manner [18]. However, microprotocols cannot be arbitrarily interleaved and can be composed only in limited ways. Our composition axioms add a level of indirection which allows us to arbitrarily compose protocols. Composite protocols can also be built in a top-down manner by associating refinements with abstract transitions in Colored Petri Nets [12]. Again, the refinement cannot interleave with the protocol and thus arbitrary dependencies cannot be handled.

OMG's Model-Driven Architecture (MDA) [13] promotes multiple levels of abstractions (or models) of a system. Transformations between these models can be defined, allowing a much better way of handling a change at a particular level. In this regard, MDA philosophy is the most relevant to the concept of adaptation. MDA is highly generic and our approach can be seen as a special case in it.

Future Work.
The present paper introduces key computational abstractions and primitives with which to model processes and their adaptations. This opens up a fruitful line of research for service-oriented computing. The following are some interesting research questions. What are the boundaries of

adaptability? Can any protocol be adapted to any other protocol? If not, then what are the conditions that would allow such an adaptability? Also, successfully handling the changes takes more than a change of design. In particular, a change in the business model may trigger a variety of changes in the organizational structure. How would organizational structure and adaptation feature in a treatment of process adaptation?

References

[1] BPEL. Business process execution language for web services, version 1.1, Feb. 2005. www-106.ibm.com/developerworks/webservices/library/ws-bpel.

[2] P. Bresciani, A. Perini, P. Giorgini, F. Guinchiglia, and J. Mylopolous. Tropos: An agent-oriented software development methodology. *Autonomous Agents and Multi-Agent Systems*, 8(3):203–236, May 2004.

[3] C. Bussler. The role of B2B protocols in inter-enterprise process execution. In *Proceedings of the International Workshop on Technologies for E-Services*, pages 16–29, 2001.

[4] CrossFlow consortium / AGFIL. Insurance (motor damage claims) scenario. Technical report, CrossFlow consortium, 1999. Document identifier: D1.a, http://www.crossflow.org.

[5] DAML Services Coalition. DAML-S: Web service description for the semantic Web. In *Proceedings of the 1st International Semantic Web Conference (ISWC)*, July 2002.

[6] N. Desai, A. U. Mallya, A. K. Chopra, and M. P. Singh. Interaction protocols as design abstractions for business processes. *IEEE Transactions on Software Engineering*, 31(12):1015–1027, 2005.

[7] E. J. Friedman-Hill. *Jess in Action: Java Rule-based Systems*. Manning, 2003.

[8] P. Harmon. *Business Process Change: A Manager's Guide to Improving, Redesigning, and Automating Processes*. Morgan Kaufmann, 2002.

[9] B. Henderson-Sellers and P. Giorgini. *Agent-Oriented Methodologies*. IDEA Group Publishing, 2005.

[10] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean. SWRL: A semantic web rule language combining OWL and RuleML, May, 2004 (W3C Submission). http://www.w3.org/Submission/2004/SUBM-SWRL-20040521/.

[11] N. R. Jennings, P. Faratin, T. J. Norman, P. O'Brien, and B. Odgers. Autonomous agents for business process management. *Applied Artificial Intelligence*, 14(2):145–189, 2000.

[12] H. Mazouzi, A. E. F. Seghrouchni, and S. Haddad. Open protocol design for complex interactions in multi-agent systems. In *Proceedings of AAMAS*, pages 517–526, 2002.

[13] OMG. MDA: Model driven architecture. http://www.omg.org/mda.

[14] OWL-P Examples. Business protocols modeled with owl-p. http://research.csc.ncsu.edu/mas/OWL-P/.

[15] OWL-P Project. Software, tools, and documentation. http://projects.semwebcentral.org/projects/owlp/.

[16] Protégé. The Protégé ontology editor and knowledge acquisition system, 2004. http://protege.stanford.edu/.

[17] M. P. Singh and M. N. Huhns. *Service-Oriented Computing: Semantics, Processes, Agents*. Wiley, 2005.

[18] B. Vitteau and M.-P. Huget. Modularity in interaction protocols. In F. Dignum, editor, *Advances in Agent Communication*, volume 2922 of *LNCS*, pages 291–309, 2004.

[19] F. Zambonelli, N. R. Jennings, and M. Wooldridge. Developing multiagent systems: The Gaia methodology. *ACM Transactions on Software Engineering Methodology*, 12(3):317–370, 2003.

A. Protocol *Rep*

Rep1. start $\wedge$ CC(Rp, O, serviceReq $\wedge$ claimValid, claimService) $\Rightarrow$ repairReq(claimNO, policyNO) $\wedge$ CC(O, Rp, repairServed, affirm)
Rep2. repairReq(claimNO, $\cdot$) $\Rightarrow$ repaired(claimNO, $\cdot$)
Rep3. repaired(claimNO, $\cdot$) $\Rightarrow$ repairOK(claimNO, approval)
Rep4. repaired(claimNO, policyNO) $\Rightarrow$ repairNOK(claimNO, policyNO)
Rep5. repairReq(claimNO, policyNO) $\Rightarrow$ serviceReq
Rep6. repaired(claimNO, policyNO) $\wedge$ repairReq(claimNO, policyNO) $\Rightarrow$ repairServed
Rep7. repairOK(claimNO, approval) $\wedge$ repaired(claimNO, policyNO) $\Rightarrow$ affirm
Rep8. repairNOK(claimNO, policyNO) $\wedge$ repaired(claimNO, policyNO) $\Rightarrow$ affirm
Rep9. repaired(claimNO, policyNO) $\wedge$ repairOK(claimNO, $\cdot$) $\Rightarrow$ claimService

B. Protocol *Han*

Han1. start $\wedge$ CC(A, H, inspectReq, inspectRes) $\Rightarrow$ estimate(claimNO, price) $\wedge$ CC(G, H, acceptEstimate, performRepair)
Han2. estimate(claimNO, price) $\Rightarrow$ deal(claimNO, price) $\wedge$ CC(H, G, performRepair, payment)
Han3. estimate(claimNO, price) $\Rightarrow$ noDeal(claimNO, price)
Han4. estimate(claimNO, price) $\Rightarrow$ inspect(claimNO)
Han5. inspect(claimNO) $\Rightarrow$ inspected(claimNO, cost)
Han6. deal(claimNO, price) $\Rightarrow$ bill(claimNO, price, approval)
Han7. bill(claimNO, price, approval) $\Rightarrow$ pay(claimNO, price)
Han8. pay(claimNO, price) $\wedge$ bill(claimNO, price, $\cdot$) $\Rightarrow$ payment
Han9. inspect(claimNO) $\Rightarrow$ inspectReq
Han10. inspected(claimNO, $\cdot$) $\wedge$ inspect(claimNO) $\Rightarrow$ inspectRes
Han11. deal(claimNO, price) $\wedge$ estimate(claimNO, price) $\Rightarrow$ acceptEstimate
Han12. bill(claimNO, price, approval) $\wedge$ deal(claimNO, price) $\Rightarrow$ performRepair