# Amoeba: A Methodology for Modeling and Evolving Cross-Organizational Business Processes

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

Business service engagements involve processes that extend across two or more autonomous organizations. Because of regulatory and competitive reasons, requirements for cross-organizational business processes often evolve in subtle ways. The changes may concern the business transactions supported by a process, the organizational structure of the parties participating in the process, or the contextual policies that apply to the process. Current business process modeling approaches handle such changes in an ad hoc manner, and lack a principled means for determining what needs to be changed and where. Cross-organizational settings exacerbate the shortcomings of traditional approaches because changes in one organization can potentially affect the workings of another.

This paper describes Amoeba, a methodology for business processes that is based on business *protocols*. Protocols capture the business meaning of interactions among autonomous parties via commitments. Amoeba includes guidelines for (1) specifying cross-organizational processes using business protocols, and (2) handling the evolution of requirements via a novel application of protocol composition. This paper evaluates Amoeba using enhancements of a real-life business scenario of auto-insurance claim processing, and an aerospace case study.

Categories and Subject Descriptors: H.1.0 [**Information Systems**]: Models and Principles—*General*; D.2.1 [**Software Engineering**]: Requirements/Specifications—*Methodologies*; K.4.3 [**Computing Milieux**]: Organizational Impacts—*Reengineering*; I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Multiagent systems*

General Terms: Design

Additional Key Words and Phrases: Business process modeling, requirements evolution, business protocols

## 1. INTRODUCTION

Successful cross-organizational business process management requires supporting the participants' *autonomy*, *heterogeneity*, and *dynamism* [Singh and Huhns 2005, pp. 7–10]. Supporting autonomy means modeling and enacting business processes in a manner that minimally constrains the participants, thus maximizing their flexibility. Supporting heterogeneity means specifying the interactions among the participants, not their internal business logics. Supporting dynamism means dealing with changing business requirements in a

Nirmit Desai: IBM India Research Lab, Embassy Golf Links Business Park, Bangalore 560071, India nirmdesa@in.ibm.com

Amit K. Chopra and Munindar P. Singh: Department of Computer Science, North Carolina State University, Raleigh, NC 27695-8206, USA. {akchopra,singh}@ncsu.edu

Some parts of this paper previously appeared as [Desai et al. 2006], which won the *Best Student Paper Award* at the *IEEE International Services Computing Conference (SCC)*, Chicago, 2006. This paper incorporates substantial revisions and extensions, especially in terms of contributing a methodology.

natural manner. Dynamism is crucial because modern businesses must often reconfigure in the face of regulatory and competitive pressures. This paper concentrates on requirements that pertain to interactions among the participants of a cross-organizational process. It proposes guidelines not only for creating a process model but also for modifying a process model to accommodate evolving requirements.

Information Technology practice increasingly recognizes the challenges of requirements evolution in business processes [Smith and Fingar 2002]. The term *business-IT divide* alludes partly to the difficulty of accommodating changing business needs in current IT systems. Smith and Fingar observe the importance of interaction and note that, as a process evolves, its set of participants and their capabilities may grow or shrink. Interestingly, we realized after naming our project that Smith and Fingar refer to process evolution as being "amoeba-like" (ch. 2). Existing approaches either ignore interaction or address it purely in low-level terms that correspond more to implementation (such as by specifying the legal sequences of message exchanges between services) than to the business-level specification of interaction. Consequently, existing approaches not only limit flexibility in implementation, but also lack a notion of compliance suitable for business interaction.

A key feature of our approach is its treatment of interaction at the level of business meaning, not merely at the level of messaging, as is common today. We use business protocols as the basic building blocks of business processes. A business *protocol* specifies a conceptually cohesive set of interactions among two or more roles. Examples include *Order* placement, *Payment*, and *Shipping*. A protocol is

—meaningful, being based on a business purpose associated with each interaction in terms of commitments and other propositions [Yolum and Singh 2002; Winikoff et al. 2005];

—abstract because, like a component interface, it does not model the proprietary reasoning databases or business logic (e.g., what item to ship and what price to quote) of the agents enacting its roles; and

—modular because it groups interactions relating to a specific business goal while supporting composition with other protocols.

We employ UML sequence charts (with extensions to notate commitments) to graphically depict selected scenarios of protocols. However, the textual descriptions and the corresponding formal specifications (provided below) are definitive. (As a convention, in this paper, we write protocol names *Slanted* and role names in SMALL CAPS.) For example, Fig. 1 shows a scenario of *Order* protocol for specifying interactions between a BUYER and a SELLER. Here, the BUYER sends a request for quote for an item to which the SELLER responds with a quote. The business meaning of a message is captured in terms of commitments (shown below the given message). For example, sending a quote creates a commitment from the SELLER to the BUYER that if the BUYER pays, the SELLER will deliver the goods. The BUYER may accept or reject the quote (Fig. 1 shows only the acceptance scenario).

*Commitments* are central to representing the business meaning of a protocol. In simple terms, commitments are reified directed obligations: they can be flexibly manipulated, such as via delegation and assignment [Singh 1999]. (Section 2.1 discusses commitments in greater detail.) As the agents participating in a business protocol interact, they enter into and manipulate their commitments to each other. Commitments yield a notion of compliance expressly suited to business processes: an agent is compliant as long as it

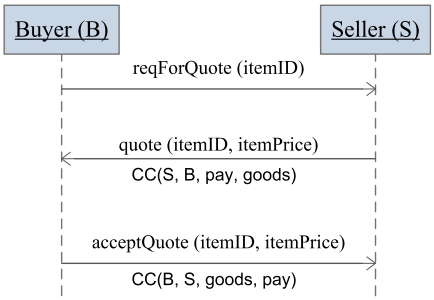


Fig. 1. Example: A scenario of *Order* protocol

discharges its commitments. This, in turn, enables flexible implementation and enactment: all runs of the protocol wherein the participants discharge their commitments are allowed.

Protocols may be composed [Desai et al. 2005]. For example, we may define *Purchase* as a composition of *Order*, *Payment*, and *Shipping*. The same protocols may be composed in different ways, thus enabling their reuse across processes. A composed protocol is like any other protocol in every way: the only difference might be that some protocols exist before the process design and some are created during the design. We show how composition is central to the ability to adapt process models according to evolving requirements.

Following Singh *et al.*'s classification of architectural patterns for services [2009], we identify three classes of business requirements and changes to them. The identification of the classes derives from three architectural elements of business processes: the transactions a business process represents, the organizations that participate in a process, and the overarching context within which the process operates. Fig. 2 depicts these elements. The corresponding classes of requirements are as follows.

—*Transactional*. The business transaction that the process seeks to accomplish, e.g., a purchase. An example of change is if we decide to modify a purchase process to include refunds for damaged goods.

—*Structural*. The relationships within and among the organizations involved, such as which party plays which role, or whether a party may delegate or assign certain commitments to another party. An example of change is when a vendor outsources payment processing to another party.

—*Contextual*. The rules of encounter to which the business process is subject. For example, a contract is voided in case of fraud by any of the participants. An example of change is when marketplace rules or government regulations change.

The above classification is one way of partitioning the requirements space. Others have studied alternative—yet similar in spirit—classes of requirements and changes [Harker and Eason 1993; Lam and Loomes 1998]. To evaluate Amoeba, we identify requirements changes corresponding to each of the above classes in the case of a real-world process, and describe how the guidelines proposed in Amoeba handle these changes.

### Contributions

This paper seeks to justify the claim that commitment-based process modeling better accommodates requirements evolution in cross-organizational processes than traditional ap-

proaches do. Based on previous work on protocol specification and enactment, this paper proposes the Amoeba methodology for designing and maintaining cross-organizational processes. Two advantages of using protocols are that they not only enable flexible enactment [Winikoff 2007; Yolum and Singh 2002] but also facilitate accommodating requirements changes [Desai et al. 2006]. Amoeba gives center stage to managing commitments among the participants as a way of handling requirements evolution. It shows (1) how to derive protocols from interaction requirements as may be hidden in conventional designs and (2) how to guide designers in accommodating evolving interaction requirements. A real-world business process scenario helps evaluate Amoeba.

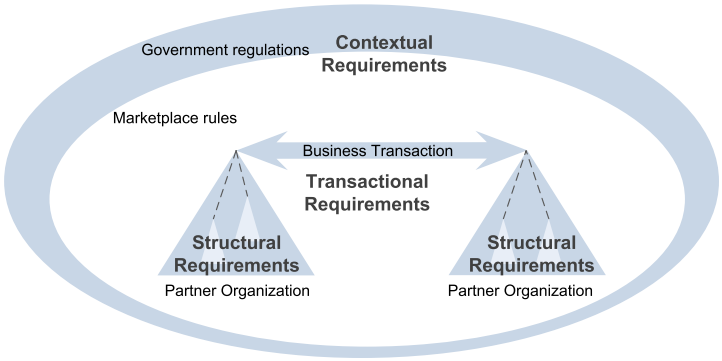


Fig. 2. Three elements of requirements of cross-organizational business processes

### Organization

Section 2 introduces the background on commitments and protocols necessary for Amoeba. Section 3 introduces and applies Amoeba to a real-life insurance industry scenario. Section 4 addresses handling requirements changes that pertain to the interactions among the participants in a process. It applies Amoeba to the insurance scenario as it goes through the three kinds of requirements changes outlined above. Section 5 evaluates Amoeba by applying it to a case study from a European Union project. Section 6 discusses related work and outlines some directions for future research.

## 2. BACKGROUND AND RUNNING EXAMPLE

This section briefly presents the key concepts of commitments, protocols, and protocol composition needed to understand Amoeba. Since the participants in cross-organizational processes are autonomous and heterogeneous, we represent them computationally as *agents* [Wooldridge 2002; Singh and Huhns 2005]. Whereas agents are instantiated executable entities, *roles* are abstract entities. Thus, protocols are specified in terms of roles, and *business processes* as instantiations of protocols where each agent plays one or more roles. The (generally private) business logic of an agent determines how it plays its roles. A *process model* consists of a protocol and a set of preexisting contractual relationships among its roles.

We use the term *participant* in the descriptions of Amoeba to emphasize that the processes are specified in terms of business entities. The participants are abstracted into roles, and the roles would be played by agents who realize the various participants.

### 2.1 Commitments

Commitments [Singh 1999] help capture the business meaning of the interactions of interest. At runtime, the commitments arise among agents. In the models, commitments are expressed abstractly among roles. The following discussion is about agents, but applies equally to roles. A *base* commitment $\mathsf{C}(x, y, p)$ denotes that agent $x$ is committed (roughly obligated) to agent $y$ for bringing about condition $p$. Here, $x$ is the debtor, $y$ the creditor, and $p$ the condition of the commitment. Commitments can be *conditional*, denoted by $\mathsf{CC}(x, y, p, q)$, meaning that $x$ is committed to $y$ to bringing about $q$ if $p$ holds. Here $p$ is called the antecedent of the commitment and $q$ its consequent. A base commitment is merely an abbreviation for a conditional commitment whose antecedent is true. A *commitment condition* is a subformula of the antecedent or consequent of a commitment. Commitments are created, satisfied, and transformed according to the following operations:

—CREATE$(x, y, p, q)$ is an operation performed by $x$ and it causes $\mathsf{C}(x, y, p, q)$ to hold.

—CANCEL$(x, y, p, q)$ is an operation performed by $x$ and it causes $\mathsf{C}(x, y, p, q)$ to not hold.

—RELEASE$(x, y, p, q)$ is an operation performed by $y$ and it causes $\mathsf{C}(x, y, p, q)$ to not hold.

—DELEGATE$(x, y, p, q, z)$ is an operation performed by $x$ and it causes $\mathsf{C}(z, y, p, q)$ to hold.

—ASSIGN$(x, y, p, q, z)$ is an operation performed by $y$ and it causes $\mathsf{C}(x, z, p, q)$ to hold.

The rules below describe the discharge of a commitment. Each rule is specified in terms of the conditions and the caused actions.

—A base commitment is discharged when its consequent is brought about.

—A conditional commitment is detached when its antecedent is brought about, and a corresponding base commitment is created.

—A conditional commitment is discharged when its consequent is brought about. No base commitment is created in this case because the consequent has already been brought about.

Consider, for example, a scenario where a buyer and a seller are exchanging goods for payment. A conditional commitment CC(BUYER, SELLER, goods, payment) denotes an obligation from the buyer to the seller that if the goods are delivered, the buyer will pay. In the event that the antecedent goods holds, the conditional commitment changes to a base commitment C(BUYER, SELLER, payment). In the event that payment holds, the buyer's commitment is discharged. Commitments do not imply temporal ordering. For example, payment may happen before goods, thus discharging the above conditional commitment.

We give messages a business meaning by specifying how they affect various commitments. In the example above, a shipment message would bring about the antecedent goods and a payment message would bring about the consequent payment. In the *Order* protocol

of Fig. 1, sending a quote message creates a commitment for the SELLER. As the interaction progresses, the messages exchanged manipulate the commitments. At any time, the active commitments reflect the pending obligations of the concerned parties.

Previous works describe the formal semantics of all commitment operations, especially in the face of concurrency [Desai et al. 2005; Desai et al. 2007]. Other considerations include the transfer (or not) of responsibility upon a delegate or assign [Singh et al. 2009]. For example, a payer may relinquish responsibility of paying by delegating its commitment to pay to a bank. Conversely, a seller may not relinquish its responsibility of delivering some goods by delegating the commitment to a shipper. Business scenarios can differ in this regard. The present examples involve retaining responsibility, which is the more complex situation.

### 2.2 Specifying a Protocol

The following discussion provides an overview of our protocol specification language; the semantics of the language is formalized elsewhere [Desai and Singh 2007]. Briefly, a protocol specification maps to a transition system consisting of states and transition between the states. Each path in the transition system corresponds to a *conversation* in the protocol. For example, the *Order* protocol introduced earlier yields the transition system of Fig. 3. Events occur along transitions and a state is defined by the fluents that hold in it.

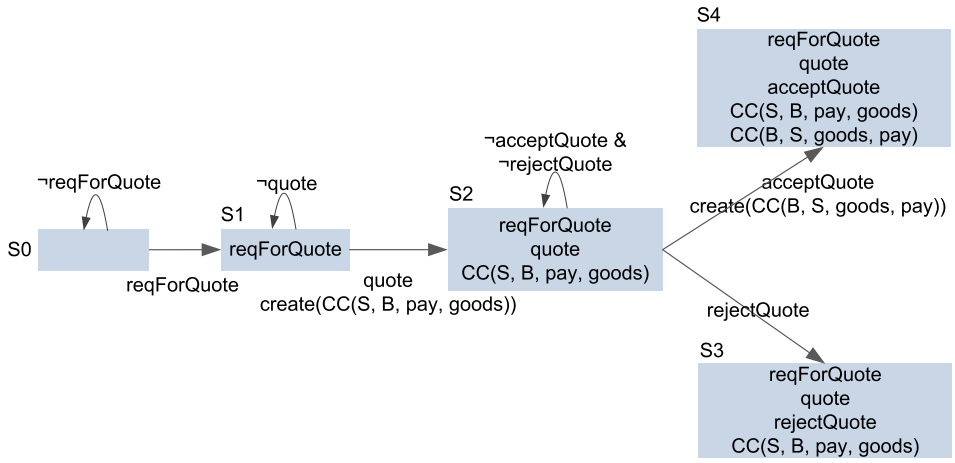


Fig. 3. The transition system model of the *Order* protocol

A protocol primarily specifies one or more messages in terms of the conditions they bring about, and the operations they perform on commitments. Further, a protocol constrains the occurrence and ordering of the messages. A protocol specification thus consists of role and message declarations, and logical axioms. Three axiom schemas are relevant.

—*Message* axioms specify the effects of messages—the effects may be conditional. Message axioms are written $\ulcorner$*event* $\rightarrow$ *effect* if *premise* $\urcorner$, where *event* is a message, *effect* is either a commitment operation or a commitment condition, and *premise* is a logical expression over fluents. An *event* corresponds to the exchange of a message; the event counts as bringing about the specified effects only if the specified premises hold. For

example, the payment of a specified amount would count as discharging the commitment to pay that amount. And, quoting a price for an item to a customer may create a commitment to deliver the specified item if the customer pays the price. The latter can be specified as ⌜ quote → CREATE(CC(S, B, pay, goods)) if true ⌝. For simplicity, we omit the name of the operation when it is CREATE and the premise when it is true.

—*Data Flow* axioms specify the data flow from the parameters of a source message to those of a sink message. Data flow axioms are written $\ulcorner msg_1.param_1 \leadsto msg_2.param_2 \urcorner$, where $msg_1$ is the source message and $msg_2$ is the sink message. Such axioms specify the constraint that in all conversations of the protocol, the sink message parameter is bound to the value of the source message parameter. For example, in *Order*, the item parameter of a quote must match that of the preceding reqForQuote. This can be specified as $\ulcorner reqForQuote.itemID \leadsto quote.itemID \urcorner$. To specify data flow across protocols, we qualify the source and sink messages by the respective protocol names.

—*Event Order* axioms specify temporal dependencies between the occurrences of various messages. Event order axioms are written either $\ulcorner msg_1 \prec msg_2 \urcorner$ or $\ulcorner msg_1$ XOR $msg_2 \urcorner$. The first schema specifies the constraint that in all conversations of the protocol, $msg_1$ must occur before $msg_2$. For example, in (prepaid) purchase, an item must be paid for before it is shipped. This can be specified as $\ulcorner pay \prec goods \urcorner$. The second schema specifies the constraint that in all conversations, either $msg_1$ or $msg_2$ but not both can occur. (This interpretation is different from the conventional meaning of XOR, since it does not require one of the alternatives.) For example, in *Order*, the buyer may either accept or reject a quote exclusively. This can be specified as $\ulcorner accept$ XOR $reject \urcorner$. Note that a data flow axiom implicitly specifies a temporal ordering with the source message preceding the sink message.

These axiom schemas are subject to important well-formedness properties, as discussed in our prior work [Desai and Singh 2007]. The following axioms specify *Order* of Fig. 1, further illustrating the above axiom schemas of our language. For readability, we elide the parameters of the commitment conditions in the figures although we include them in the formal specifications.

**ORD$_1$.** quote(itemID, itemPrice) → CC(S, B, pay(itemPrice), goods(itemID))

**ORD$_2$.** acceptQuote(itemID, itemPrice) → CC(B, S, goods(itemID), pay(itemPrice))

**ORD$_3$.** reqForQuote.itemID ⇝ quote.itemID

**ORD$_4$.** quote.itemID ⇝ acceptQuote.itemID

**ORD$_5$.** quote.itemPrice ⇝ acceptQuote.itemPrice

**ORD$_6$.** quote.itemID ⇝ rejectQuote.itemID

**ORD$_7$.** quote.itemID ⇝ rejectQuote.itemPrice

**ORD$_8$.** acceptQuote XOR rejectQuote

The parties enacting a protocol would play their respective roles in that protocol. Their behavior is constrained only up to their commitments. For example, in *Order*, when the SELLER quotes a price, it commits to providing the goods at that price. Whether goods are shipped first or the payment is made first does not matter for this commitment. Also, whether and when the receipts are provided is immaterial.

As mentioned earlier, protocol specifications map to transition systems, against which queries may be run to establish useful formal properties of protocols [Desai and Singh

2007]. Also, tools exists that help extract role skeletons—a role's perspective of the interaction—from the protocols. Role skeletons can be augmented with business policies to create executable agents [Desai et al. 2005]. A business process corresponding to a protocol is *enacted* when agents that have derived role skeletons from the protocol interact with each other. It is worth noting that protocol specifications are enactable if and only if they satisfy certain well-formedness properties [Desai and Singh 2008].

### 2.3 Composing Protocols: Concepts

The power of protocols in modeling arises from the fact that they can be readily composed. A classical example is *Purchase*, which can be modeled as a composition of simpler protocols that handle *Order*, *Payment*, and *Shipping*, respectively. A composite protocol is treated on par with any other protocol.

Specifying the composition of two or more protocols involves the axiom schemas of Section 2.2 augmented with the following, which help relate the roles of the protocols being composed.

—*Role Identification* axioms specify how a role in the composite protocol replaces selected roles in existing protocols (as debtor or creditor of any commitments and sender or receiver of any messages). Role identification axioms are written $\ulcorner protocol_{new}.role_{new} \doteq protocol_1.role_1, \ldots, protocol_k.role_k \urcorner$. Here $role_{new}$ is a role in the composite protocol $protocol_{new}$ and for $1 \leq j \leq k$, $protocol_j$ is one of the protocols being composed and $role_j$ is a role selected from it. Multiple roles from the same protocol may be identified. The constraint being specified is that the agent playing $role_{new}$ must play each $role_j$. In the model, this is achieved by simply renaming each $role_j$ to $role_{new}$. For example, the PURCHASER in *Purchase* would be the BUYER in *Order*, the PAYER in *Payment*, and the RECEIVER in *Shipping*. This can be specified as $\ulcorner Purchase.purchaser \doteq Order.buyer, Payment.payer, Shipping.receiver \urcorner$.

Section 3.5 illustrates the composition of two protocols.

### 2.4 Running Example: Automobile insurance processing

This paper demonstrates and evaluates Amoeba using a real-life insurance claim processing case [Browne and Kellett 1999]. This real-life business scenario provides an independent and significant test-case and helps contrast Amoeba with a traditional approach. This contrast is significant because even the prevalent process modeling techniques such as BPEL [2007] are based on the abstraction of workflows similar to those employed in the insurance case study. Section 5 describes how Amoeba applies in a case study from the aerospace industry.

Fig. 4 (due to [Browne and Kellett 1999]) shows the parties involved and the overall process flow. AGF Irish Life Holdings (AGFIL), a subsidiary of Allianz, is an insurance company in Ireland. AGFIL underwrites automobile insurance policies and covers losses incurred by policy holders. Europ Assist provides a 24-hour help-line service for receiving claims. An approved mechanic provides repair services. Lee CS is a consulting firm that coordinates with AGFIL and the mechanics to handle a claim. AGFIL holds ultimate control in deciding if a given claim is valid and if payment is made to a mechanic.

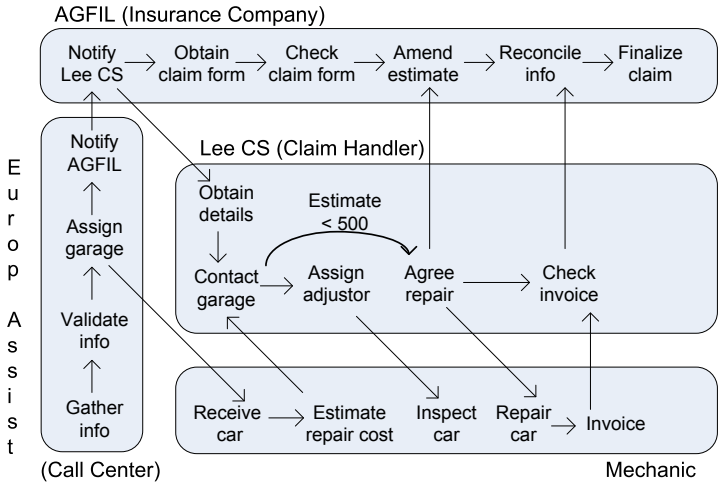


Fig. 4. Traditional model of a cross-organizational insurance claim process

## 3. MODELING A CROSS-ORGANIZATIONAL BUSINESS PROCESS VIA PROTOCOLS

The motivation behind business protocols is to address the major shortcomings of the traditional approaches for modeling cross-organizational business processes [Desai et al. 2005].

To contrast protocol and traditional flow-based representations, notice that each box in Fig. 4 represents a participant's flow that synchronizes at various touch points with other flows. However, the touch points lack a business-level meaning: they fix the control flow but ignore the business significance of the synchronization. Real-life cases are rife with subtle business significance. For example, one might ask: does AGFIL relinquish the responsibility for a claim to Lee CS by asking it to handle it? Is it significant at the business-level if Europ Assist assigns a claim to a garage before notifying AGFIL? How does it affect the various business relationships if inspectors are directly controlled by AGFIL and not by Lee CS? There is no basis for answering such questions in the absence of an appropriate business-level meaning. Organizations must thus follow a rigid sequence of steps and any deviation from this expected sequence must be treated as significant, regardless of the business-level significance—or lack thereof—of the deviation.

By contrast, based on its explicit meaning, a protocol can be readily substituted by another protocol involving different roles or messages while preserving the business meaning of the overall process. For example, a single message may be replaced by an extended negotiation or vice versa. In a purchase process, a merchant—instead of following a rigid sequence of steps—may advertise goods instead of waiting for requests for quotes, and yet remain compliant with the protocol if it discharges its commitments. The essence of the interaction—captured via appropriate commitments—is to exchange goods and money without constraining the agents to follow a rigid sequence of steps.

Another limitation of the flow-oriented approaches is that the flows are monolithic, and formed by *ad hoc* intertwining of the participants' internal business logics and external interactions. For example, Lee CS may have a unique policy to behave differently if the estimated cost of repairs is under a threshold amount. Since such business logic is typically proprietary, the flow of one agent would not be available for reuse by another. Moreover,

since such business logic is contextual, the flow of one agent would not be readily usable by another agent, even when available. For instance, if a new consultant were to participate in this process, its flow would need to be developed from scratch, even though it would interact with the other partners in the same manner as Lee CS does. Protocols capture the reusable interaction patterns and abstract out the business logic. Each agent would specify its business logic as it adopts a role in a protocol.

To illustrate a shortcoming of flow-oriented approaches, it is worth remarking that Fig. 4 does not include the insurance holder. From the standpoint of interactions, this is a major shortcoming. 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. Although omitting a participant may be an exception rather than the norm, it points to the unsuitability of flow-oriented modeling abstractions for cross-organizational processes.

Table I. The main steps of Amoeba

| Step | Description | Knowledge Required | Artifacts Produced |
|---|---|---|---|
| M1 | Identify roles played by the participants in the process, and the corresponding interactions | Boundaries of autonomy | Role identities and protocol names (with message declarations) |
| M2 | Identify and capture contractual relationships as commitments | Roles and expectations from roles | Commitments describing contractual relationships |
| M3 | Specify message meanings emphasizing the creation and manipulation of commitments | How the contracts play out | Message axioms |
| M4 | Specify constraints on message occurrence and ordering within each protocol | Bindings among parameters and applicable conventions regarding order | Data flow and event order axioms |
| M5 | Compose individual protocols to specify the process model | How the roles are identified in the process; how the messages affect commitments; and how the messages are constrained | Process model specifying the participants' interactions |

Commitments and protocols yield a natural way of modeling a cross-organizational process in terms of interactions among the roles, which would be played by autonomous business partners. What Amoeba primarily demonstrates is how (as Section 4 shows) protocol-based modeling yields a natural treatment of evolving requirements. Table I outlines Amoeba, showing the inputs and outputs for its main steps, which are described below in greater detail. The steps are to be performed in sequence and may be iterated over.

Previous work on protocol-based process modeling represents the actors, their goals, and their mutual dependencies to induce the protocols of interest [Mallya and Singh 2006; Bresciani et al. 2004]. Additionally, Amoeba accommodates the equally important situation when a process has already been modeled using traditional means. Accordingly, this section illustrates Amoeba in such a reverse engineering setting, where protocols are identified from a traditionally modeled cross-organizational process. However, the above steps

are equally applicable to a case where no traditional model exists and the protocols must be identified and designed from scratch.

When a conventional process model exists, it often includes a specification of the business logics of one or more of the participants (depending on whose perspective was taken in the original model). Such business logics can be readily identified as they do not involve interacting with another participant. However, whereas interactions are reusable, business logics generally are not. For this reason, Amoeba concentrates on the interactions among the participants and disregards their business logics. For example, how Lee CS determines whether an estimate is below a certain threshold depends upon its business logic. Any other claims handler would participate in the same interactions, but would apply its own potentially distinct business logic.

### 3.1 Step M1: Identify Roles and Protocols

3.1.1 *Identify Participants.* Since we begin from a concrete process, it helps to first identify the participants and then abstract them into the roles of interest. The participants are the units of autonomy at a chosen level of detail. In general, identifying the relevant participants requires human insight.

An existing process may have been specified as a *choreography*, i.e., a constrained set of messages exchanged among the participants from a global perspective, or as an *orchestration*, i.e., control and data flows of service invocations from a single participant's perspective. A choreography makes the participants explicit, but might sometimes artificially separate a participant into multiple roles. A monolithic orchestration would invoke services: the participants are the orchestrator and the providers of the invoked services. (When the participants are proactive, more than one orchestration may be involved. Fig. 4 illustrates synchronized orchestrations: each box therein represents a participant.)

Thus, the participants in the insurance process are AGFIL, Lee CS, Europ Assist, the mechanics, the policy holder, and the inspectors (the last two having been added by us.)

3.1.2 *Identify and Group Logically Related Interactions.* Some interactions together contribute to a related business purpose. These interactions include communications such as reporting claims, gathering information (policy holder and call center), validating policy details (call center and insurance company), and so on. Likewise, interactions pertaining to the buying and selling of insurance coverage go together. In some cases, existing (traditional) models may fail to identify an interaction. For example, AGFIL would obtain the claim form from Europ Assist even though Fig. 4 omits this interaction.

3.1.3 *Map Participants to Roles and Interactions to Protocols.* Each related group of interactions identified in Section 3.1.2 forms a protocol. For example, the interactions related to receiving claims would naturally go together as a claim reception protocol. For each protocol, define suitable roles, ideally with names that reflect their essential contribution to that protocol. For example, describe the claim reception protocol as arising between the REPORTER, CALL CENTER, and PROVIDER roles, not the specific participants. These roles would apply in any process involving insurance claims, not just the present process. Likewise, we can consider *Ins*, the insurance buying (and selling) protocol, which involves two roles (SUBSCRIBER (S) and VENDOR (V)).

Interactions map to sets of messages. For each message, identify its parameters, including identifiers of the information records exchanged such as policy numbers and claim numbers. Such parameters can be found as the content of the interactions. If the protocol

is being defined from scratch, then the parameters are defined as well. For example, *Ins* includes the messages reqForQuote(driverID, coverage), quote(driverID, policyNO, premium), and pay(policyNO, premium).

### 3.2 Step M2: Identify Contractual Relationships

Identify any assumed contractual relationships among the participants that exist prior to their engaging in the process. For example, AGFIL already has partnered with Lee CS and Europ Assist. The negotiations that yield such partnerships are out of the scope of insurance claim processing. Identify additional contractual relationships that are generated as the interaction progresses. Capture both kinds of relationships in terms of commitments. In most cases, both kinds of relationships are derived from the clauses described in the legal contracts among the participants. The obligatory clauses in such legal contracts naturally map to commitments. If the obligation under question is created before the interactions take place, then it is an assumed contractual relationship. An obligation that is created as a result of the interactions is a generated contractual relationship.

In the case of assumed relationships, leave out the specification of creation of the corresponding commitments. Specify how the assumed commitments are manipulated and satisfied by the protocols. For other relationships, specify the creation, manipulation, and discharge of the corresponding commitments.

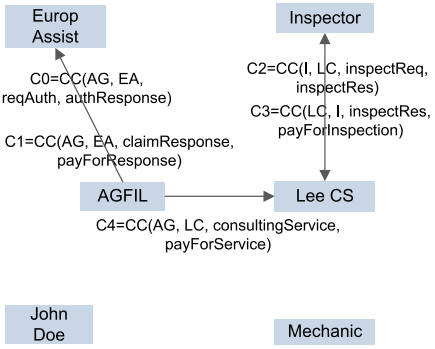


Fig. 5. Assumed relationships as commitments in the auto insurance scenario

We propose relationship diagrams to depict contractual relationships among partners in business ecosystems. A relationship diagram depicts each partner as a node. An edge between two nodes depicts a contractual relationship between the corresponding partners as a set of commitments. Edges are directed from the debtor to the creditor. Some of the edges depict assumed and some generated relationships.

Fig. 5 shows the assumed relationships in our running example. AGFIL has agreed to pay Europ Assist for responding to filed claims on its behalf. Also, AGFIL would authenticate the policy holders filing the reports. Similarly, AGFIL has agreed to pay for the consulting services provided by Lee CS. Lee CS has hired inspectors, who would assess vehicle damage and estimate repair expenses. The relationship between AGFIL and John Doe (insured) is not assumed: protocols would specify how the corresponding commitments are created.

### 3.3 Step M3: Specify Message Meanings

Specify the meaning of a message in terms of the conditions it brings about and how it affects the commitments among the participants. As an example, the axioms below capture the meanings of the messages in *Ins* (identified in Section 3.1.3). $\textsc{Ins}_1$ specifies the meaning of quote as creating a commitment from the VENDOR (abbreviated V) to the SUBSCRIBER (abbreviated S) that if the SUBSCRIBER subscribes to the policy then the VENDOR commits to providing insurance coverage. The parameters occurring in a commitment condition are bound to those of the message that creates or manipulates the commitment. $\textsc{Ins}_2$ specifies that if the payment occurs and the amount matches the quoted premium then it counts as a subscription for the specified policy with the premium paid. That is, the SUBSCRIBER is subscribed if the quoted premium is paid. $\textsc{Ins}_3$ decomposes the commitment to provide insurance coverage to the policy holder into two commitments: all valid claims must be served and all claim requests must be responded to. The parameters of these conditions are left unspecified in *Ins*: they would possibly be specified when *Ins* is composed with other protocols. Also, as long as the SUBSCRIBER is insured, multiple claims can be filed and served. This is because $\textsc{Ins}_3$ keeps creating the commitments to serve as long as the insurance commitment is active. Thus, filing and servicing a claim discharges the commitments on the right, which are recreated because the commitment on the left remains active. When the policy expires and insurance is caused, the commitment on the left is discharged.

$\textsc{Ins}_1$. quote(driverID, policyNO, premium) $\rightarrow$
CC(V, S, subscribe(policyNO, premium), insurance(policyNO))

$\textsc{Ins}_2$. pay(policyNO, premium) $\rightarrow$ subscribe(policyNO, premium) if
quote(driverID, policyNO, premium)

$\textsc{Ins}_3$. true $\rightarrow$ CC(V, S, serviceReq $\wedge$ validClaim, claimService)
$\wedge$ CC(V, S, reqForClaim, claimResponse) if C(V, S, insurance(policyNO))

### 3.4 Step M4: Specify Constraints Among Messages

Constrain message occurrences based on data flow requirements or temporal ordering requirements. The axioms below illustrate this step. $\textsc{Ins}_4$ specifies that the parameter driverID of quote must be bound to the parameter driverID of reqForQuote. Data flow axioms imply temporal ordering between the messages. Similarly, $\textsc{Ins}_5$ and $\textsc{Ins}_6$ specify that the parameters policyNO and premium of pay are bound to those of quote.

$\textsc{Ins}_4$. reqForQuote.driverID $\leadsto$ quote.driverID

$\textsc{Ins}_5$. quote.policyNO $\leadsto$ pay.policyNO

$\textsc{Ins}_6$. quote.premium $\leadsto$ pay.premium

The formal specifications of protocols (as given in the appendix) are definitive. To ease exposition, Fig. 6 (and Fig. 9 below) show representative scenarios from some protocols derived from Fig. 4. The messages in *Ins* are annotated with the commitments they create.

Fig. 7 illustrates the progression of contractual relationships in this example. As described earlier, part (a) (copied from Fig. 5) shows the relationships assumed at the outset—these relationships exist even when the process model under construction includes no protocols. As we proceed, the model progresses to include additional protocols, and the relationships affected by these protocols (as they are incorporated in the model via composition) are depicted. Part (b) depicts the contractual relationships for the model corre-

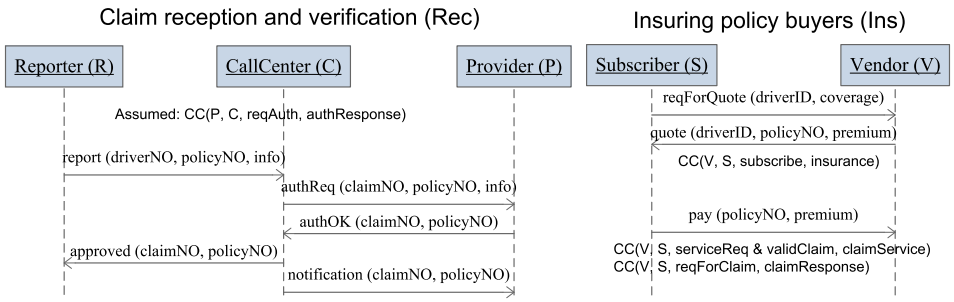


Fig. 6. Example scenarios (annotated with commitments) from *Rec* and *Ins* in the insurance claim process of Fig. 4

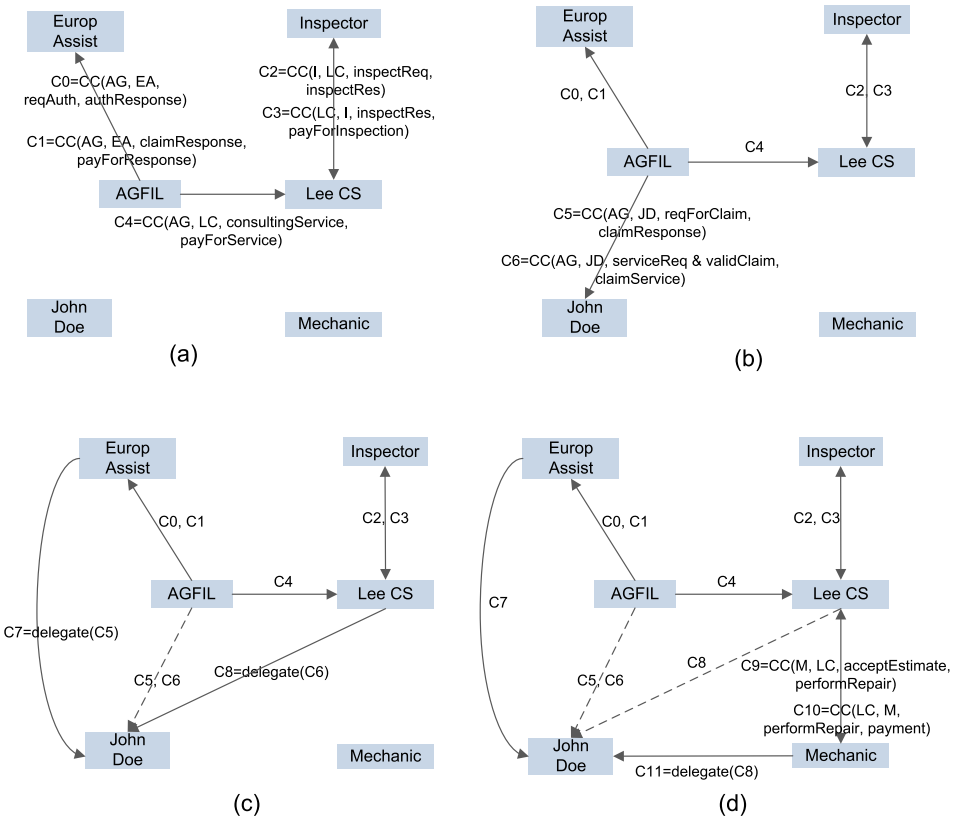


Fig. 7. The progression of contractual relationships via protocol composition

sponding to *Ins*. *Ins* creates the relationship between AGFIL and John Doe. Similarly, part (c) depicts the contractual relationships for the model corresponding to a composition of *Ins*, *Rec*, and *Mon*. In Fig. 7, dashed edges denote commitments that have been delegated to a new debtor. The original debtors remain responsible for ensuring the fulfill-

ment of the commitments: in case the new debtor fails to fulfill a commitment, the original debtor would be expected to arrange for its fulfillment. AGFIL delegates to Europ Assist its commitment to John Doe for responding to claims. AGFIL also delegates to Lee CS its commitment to John Doe for handling all valid claims. Finally, part (d) shows the contractual relationships for the model corresponding to a composition of *Ins*, *Rec*, *Mon*, and *Han*. Here, Lee CS enters into agreement with a mechanic (by virtue of *Han*), and delegates to the mechanic its commitment for providing claim services.

### 3.5 Step M5: Compose Protocols to Reconstruct a Business Process

Once we factor out individual protocols from a traditionally modeled process, we can compose these protocols to reconstruct the original process.

Any realistic business process would involve multiple protocols. For example, AGFIL participates in protocols for selling insurance policies (*Ins*) and for receiving and handling claims (*Rec*). Consider how protocols can be composed to form a more complete protocol. Let's assume that AGFIL outsources its help-line service for receiving claim reports to a call center, but handles the claims itself (without the benefit of any partnership with Lee CS, the mechanics, or the inspectors). The claim reception and validation part is supported by the protocol *Rec* (shown in Fig. 6). The requisite process can be obtained by composing *Ins* (specified above) and *Rec* (Appendix A.1) into a new protocol called *Bas* (Basic insurance claim processing).

3.5.1 *Specify role identification axioms.* Identify the roles in the desired composite protocol. These are typically mapped from the participants identified in the initial part of Step M1 (Section 3.1.1). For each of the identified roles, determine the roles of the components protocols that should be played by an agent that plays the role. Accordingly, specify a role identification axiom for each of the roles of the composite protocol.

For example, $\textbf{BAS}_1$ defines a role INSURED in protocol *Bas* and states that the SUBSCRIBER in *Ins* and the REPORTER in *Rec* are identified and replaced by INSURED in *Bas*. $\textbf{BAS}_2$ and $\textbf{BAS}_3$ similarly define additional roles in *Bas*.

**$\textbf{BAS}_1$.** Bas.Insured $\doteq$ Ins.Subscriber, Rec.Reporter

**$\textbf{BAS}_2$.** Bas.Insurer $\doteq$ Ins.Vendor, Rec.Provider

**$\textbf{BAS}_3$.** Bas.CallCenter $\doteq$ Rec.CallCenter

3.5.2 *Specify message axioms.* As in Step M3 (Section 3.3), specify the meaning of a message in terms of the conditions it brings about and how it affects the commitments among the participants. Here too the message effects are local to one protocol, although a message originally from one protocol may affect a commitment originally specified in another protocol. For each pair of messages and commitments in the protocols being composed, determine if the given message affects the commitment. If so, specify a message axiom to capture the desired effect.

For example, $\textbf{BAS}_4$ is a message axiom stating that the authentication of the REPORTER by the PROVIDER in *Rec* means that the filed claim should be counted as being valid in the context of *Ins*. Notice how the meaningful parameter claimNO for the validClaim condition of *Ins* is provided here. $\textbf{BAS}_5$ states that the reporting of a claim counts as a request for claim service in the context of *Ins*. Similarly, according to $\textbf{BAS}_6$ and $\textbf{BAS}_7$, both the approval and denial of a reported claim count as claim responses in the context of *Ins*. For brevity, let '$\cdot$' denote a parameter that is not relevant in the given axiom.

**BAS$_4$.** Rec.authOK(claimNO, policyNO) → Ins.validClaim(claimNO)

**BAS$_5$.** Rec.report(driverNO, policyNO, info) → Ins.reqForClaim(driverNO, policyNO)

**BAS$_6$.** Rec.approved(claimNO, policyNO) → Ins.claimResponse(claimNO, policyNO)
if Rec.report(·, policyNO, ·)

**BAS$_7$.** Rec.denied(claimNO, policyNO) → Ins.claimResponse(claimNO, policyNO)
if Rec.report(·, policyNO, ·)

3.5.3 *Specify data flow axioms.* As in Step M4 (Section 3.4), constrain message occurrences based on data flow requirements. The the messages being constrained may be originally specified in different protocols. For each pair of messages in the protocols being composed, determine if a parameter of a message must be bound to the value of a parameter of the other message. If so, specify a data flow axiom to capture the constraint.

For example, **BAS$_8$** binds the driverNO parameter of the quote messages in *Ins* to driverID parameter of the report message in *Rec* via a data flow axiom. Similarly, **BAS$_9$** binds the policyNO parameter.

**BAS$_8$.** Ins.quote.driverID ⇝ Rec.report.driverNO

**BAS$_9$.** Ins.quote.policyNO ⇝ Rec.report.policyNO

3.5.4 *Specify event order axioms.* As in Step M4 (Section 3.4) constrain message occurrences based on temporal ordering requirements. Again, the messages being constrained may originally be specified in different protocols. The messages constrained by a data flow axiom are implicitly temporally ordered: the source precedes the sink. Apart from these, for each pair of the messages from the protocols being composed, determine if temporal ordering or mutual exclusivity between them is desired. If so, capture it via an event order axiom. The present example does not need event order axioms. However, protocols *Rep*, *Han*, *Picp*, and *Pcsc* demonstrate such axioms.

3.5.5 *Result.* The axioms introduced above are simply unioned with the axioms of *Ins* and *Rec* to yield the protocol *Bas*, which captures the desired interactions. Fig. 12 shows a representative scenario of *Bas* in the context of another example.

Following [Desai et al. 2005], we use protocol-composition diagrams to provide a high-level view of compositions of protocols. Fig. 8 shows role identifications by binding the new roles to the original roles being identified. Message, data flow, and event order axioms are depicted as directed bridges between specified elements of the protocols being composed. The direction of the data flow axioms is the direction in which the data flow occurs. Similarly, the direction of the message axioms is the direction of causality. The direction of the event order axioms is earlier to later.

The diagrams become dense as the number of axioms grows. This is not a weakness of the notation *per se*. The notation is no less scalable than an ontology visualizer or a UML model visualizer. A tool for designing interactions would support examining a few related interactions at a time. To reduce clutter, composition diagrams in this paper show only selected axioms. A fundamental advantage of Amoeba is that in natural situations, we need to compose only a small number of protocols at a time. This helps reduce the complexity of the design compared to monolithic approaches. Imagine that Fig. 4 were to be fleshed out with details of parameters, and expanded to include additional paths to support flexible execution. It would be quite unwieldy then.

By virtue of composition, the protocols factored out of a traditional model can be composed in many ways. One of the compositions would yield the original (as traditionally

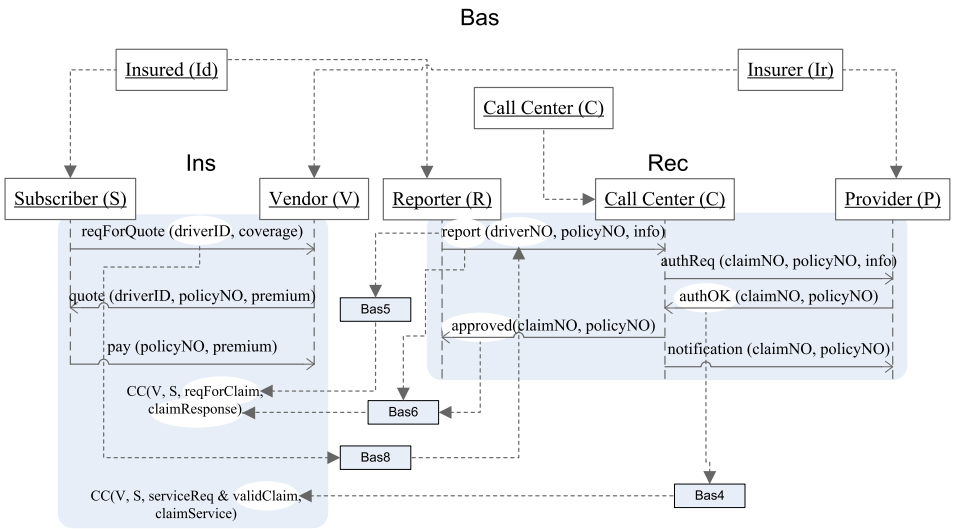


Fig. 8. Composing *Ins* with *Rec* to produce *Bas*

modeled) process. Thus, a process model based on protocols would generalize over a traditionally modeled process by including alternatives that represent various configurations of the original process.

Composing protocols is an iterative process. Initially, designers may overlook some of the desired composition axioms. Thus, tools to determine if a given composite protocol is missing important axioms are essential. Desai and Singh introduce properties of protocols that ensure they can be enacted correctly, and a method to check whether a protocol satisfies these properties [2008].

The following discussion assumes a composite protocol *Picp* (Partial insurance claim processing) constructed from the composition of *Bas*, *Mon* (Monitoring outsourced handling), *Han* (Handling filed claims), and *Rep* (Administering repairs). The roles in *Picp* are INSURER, INSURED, CONSULTANT, CALL CENTER, REPAIRER, and ASSESSOR. *Picp* carries out the core steps of the insurance claim process. Appendix A.5 specifies *Picp*.

## 4. ACCOMMODATING EVOLVING INTERACTION REQUIREMENTS

The foregoing shows how a cross-organizational process may be specified via its interaction requirements, and in a manner that deemphasizes internal business logics and behaviors. Let us now consider the evolution of the requirements of cross-organizational processes, specifically, evolution that calls for modifications in the structure of the interactions among the parties involved. Changes local to the internal functioning of any of the participants can be handled through conventional means.

Section 4.1 shows how Amoeba accommodates requirements changes that affect interactions. Sections 4.2, 4.3, and 4.4 exercise Amoeba on the three kinds of requirements changes introduced in Section 1: transactional, structural, and contextual.

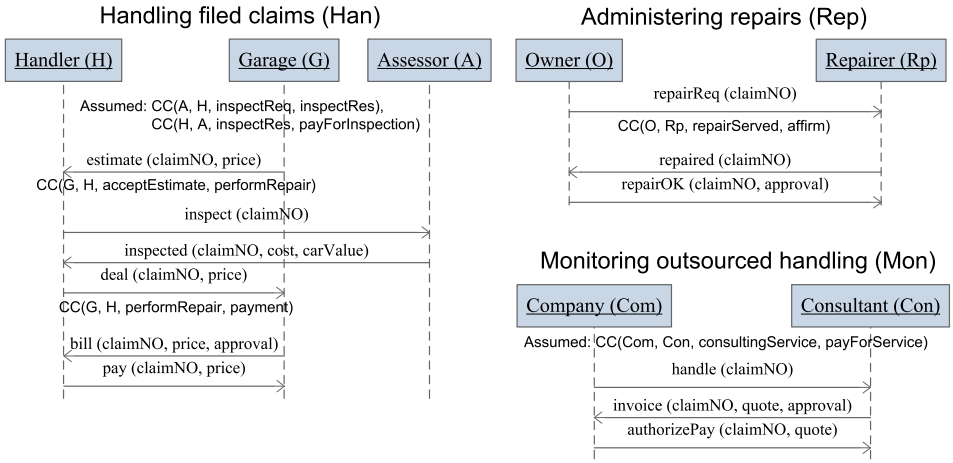


Fig. 9. Example scenarios (annotated with commitments) from *Mon*, *Rep*, and *Han* in the insurance claim process of Fig. 4

### 4.1 Amoeba: Process Adaptation via Protocol Composition

Given a changed requirement, how does a designer come up with the right set of composition axioms to adapt the process model? When a change calls for additional interactions, participants, and commitments, what guidance can we provide to a designer on what elements are needed? Let us assume we are given (1) the process model corresponding to the original process and (2) the new requirements. Then, the following steps guide designers in handling the changed requirements. In essence, these steps derive from those presented in Table I: instead of identification and specification of elements from a traditionally modeled process, these steps involve modifications and adjustments to the protocols.

4.1.1 *Step M1: Identify new roles and protocols.* Generally, the evolution of a business model results in the formation of new business relationships, possibly with new participants. Some existing participants may leave. Identify the new participants according to the original Step M1 (Section 3.1.1).

Also, the interactions among the participants may change: a new group of interactions may become necessary, or an existing protocol may become obsolete. Define a protocol for the new group of interactions according to the original Step M1 (Section 3.1.2). Drop any obsolete protocols simply by excluding them from the composition.

Finally, such evolution may introduce new roles or render existing roles unnecessary. This is a common situation. For example, buyers and sellers may switch from a direct transaction to an escrow model for routing payments, or back, thereby introducing or removing the role of an escrow agency. Following the original Step M1 (Section 3.1.3), update the mapping of participants to roles and interactions to protocols. Capture the new mappings via role identification axioms. Handle the removal of a role by fusing it with an existing role via a role identification axiom.

4.1.2 *Step M2: Identify changes to contractual relationships.* The new participants, may form new contractual relationships outside the scope of the process. Capture such new assumed contractual relationships among existing or newly identified participants as

commitments according to the original Step M2 (Section 3.2).

Determine what commitments are affected by the proposed changes. If necessary, identify newer ways to operate on existing commitments, e.g., via discharge, cancel, or delegate. Specifically, a message in a newly introduced protocol may discharge a commitment created in an existing protocol. For example, a buyer and a seller may commit to each other: the buyer to paying and the seller to delivering the goods. However, to fulfill these commitments presupposes two new protocols: *Shipping* and *Payment*, respectively.

4.1.3 *Step M3: Modify message meanings.* Either identify the existing messages, or introduce new messages to communicate the modified operations on commitments and effects on other conditions (as identified in the previous step). Express the meanings of the newly introduced messages via message axioms. Group new messages into newly defined protocols based on the commonality of their purpose. If the necessary roles are not yet defined, go back to the first step to introduce such roles.

In general, identify messages that affect the commitments in the newly formed contractual relationships. Also, identify commitments that are affected by messages in the newly defined protocols. Capture these via message axioms.

4.1.4 *Step M4: Modify message constraints.* Perform this step according to the guidance given in the original Step M4 (Section 3.4).

*Capture data flows among messages.* Capture data flows via parameter bindings, establishing the main relationships among the existing protocols and the newly defined protocols, and any refinements within the newly defined protocols.

*Capture event orders.* Typically, these constraints are not required by the data but are imposed as a matter of convention (possibly reflecting negotiations among the concerned parties). For example, we may wish to restrict *Purchase* to allow only prepayment: in that case, payment must precede shipment. Identify and capture such constraints among the existing protocols and the newly defined protocols, and any refinements within the newly defined protocols.

4.1.5 *Step M5: Compose new protocols.* Perform this step according to the guidance given in the original Step M5 (Section 3.5).

The above steps are the simplest when elements are introduced and composed with existing protocols. But what happens if an existing element needs to be changed or removed? Examples include changes in message parameters, message ordering and data flows, or message meaning. For such cases, simply replace an existing protocol with a new protocol (retrieved from a repository or defined afresh) via composition.

Let us now apply Amoeba on the three kinds of requirements changes introduced above.

### 4.2 Transactional Change: Alternative Enactment to Discharge Commitments

A transactional change is caused by a change in the way the business transaction supported by the process is carried out. A business transaction can be captured in terms of the life cycles of the commitments involved. Thus, a change in the business transaction would map to alternative life cycles of the underlying commitments. For example, damaged goods may be returned by a buyer, thereby canceling the buyer's commitment to pay if the payment was not made. If the payment was already made, a new commitment for the seller to refund

the payment is created. The transaction respects its previous function via the fulfillment of the corresponding commitments, but more flexibly than before.

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 such a case, the COMPANY pays the OWNER a sum equal to the value of the car and takes possession of the car instead of administering repairs on it. Alternatively, especially if the damage is minor, the OWNER may accept a cash settlement instead of having the car repaired. The net result is that the COMPANY becomes committed to paying the OWNER the value of the car or an amount it offers in lieu of repairs (if the OWNER accepts the offer).

4.2.1 *Step M1: Identify new roles and protocols.* There may be no change to the set of participants due to the evolution of transactional requirements. However, new roles and a new protocol are needed to capture the change.

According to the changed business policy, AGFIL, Lee CS, and the policy holders would interact in new ways to achieve a settlement. Each party would adopt a role, say, COMPANY, CONSULTANT, and OWNER, respectively. Let us name their containing protocol *Pcsc* (Pay cash and scrap car). Since no new participants are involved, the new roles are fused with the existing roles of *Picp* yielding the roles of the composite protocol *Icp* (Insurance claim processing) as follows.

**ICP$_1$.** Icp.Insured $\doteq$ Picp.Insured, Pcsc.Owner

**ICP$_2$.** Icp.Insurer $\doteq$ Picp.Insurer, Pcsc.Company

**ICP$_3$.** Icp.Consultant $\doteq$ Picp.Consultant, Pcsc.Consultant

**ICP$_4$.** Icp.Repairer $\doteq$ Picp.Repairer

**ICP$_5$.** Icp.CallCenter $\doteq$ Picp.CallCenter

**ICP$_6$.** Icp.Assessor $\doteq$ Picp.Assessor

4.2.2 *Step M2: Identify changes to contractual relationships.* AGFIL would have delegated its commitment CC(V, S, serviceReq $\wedge$ validClaim, claimService) to Lee CS, thereby making Lee CS responsible for servicing claims. Lee CS would in turn delegate the servicing of claims to the REPAIRER. Fig. 7(d) shows the REPAIRER (mechanic) being committed to the POLICY HOLDER (John Doe). When Lee CS advises scrapping the car or making a cash payment for minor damage, it fulfills its commitment to provide the consulting service. As a result, the commitment to service the claim falls back to AGFIL, which becomes committed to paying John Doe. Further, when AGFIL settles the payment, it provides the claim service, thereby discharging its commitment.

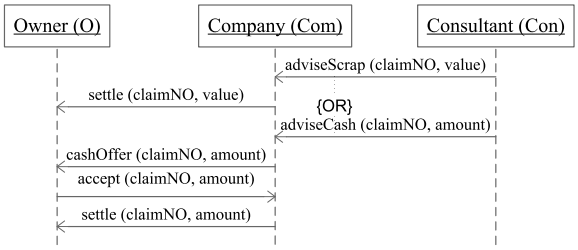


Fig. 10. A scenario of *Pcsc* (pay cash and scrap car)

4.2.3 *Step M3: Modify message meanings.* The new roles would need additional messages to effect the above described evolution of commitments. Fig. 10 shows a scenario of *Pcsc* having new messages and their respective parameters. The CONSULTANT advises the COMPANY to either scrap the car and pay the value of the car to the OWNER or to pay a cash amount in lieu of administering repairs. In the former case, the COMPANY may pay the value of the car to the OWNER via a settle message. In the latter case, the COMPANY offers a cash amount and, if the OWNER accepts the offer, pays that amount via the settle message. Appendix A.6 specifies *Pcsc.*

The following message axioms capture the desired evolution of commitments.

**ICP$_7$.** Pcsc.adviseScrap(claimNO, value) →
Picp.delegate(Con, Ir, CC(Rp, Id, serviceReq ∧ validClaim, claimService))

**ICP$_8$.** Pcsc.adviseCash(claimNO, amount) →
Picp.delegate(Con, Ir, CC(Rp, Id, serviceReq ∧ validClaim, claimService))

**ICP$_9$.** Pcsc.adviseScrap(claimNO, value) → Picp.consultingService(claimNO)

**ICP$_{10}$.** Pcsc.accept(claimNO, amount) → Picp.consultingService(claimNO)
if Pcsc.adviseCash(claimNO, amount)

**ICP$_{11}$.** Pcsc.settlement(claimNO, amount) → Picp.claimService(claimNO)
if Pcsc.accept(claimNO, amount)

**ICP$_{12}$.** Pcsc.settle(claimNO, value) → Picp.claimService(claimNO)
if Pcsc.adviseScrap(claimNO, value)

4.2.4 *Step M4: Modify message constraints.*

*Capture data flows among messages.* Since either of adviseScrap and adviseCash may occur, and since *Pcsc* does not open a new claim, the value of the claimNO parameter in these messages must flow in from other protocols. The new behavior would apply only to claims approved during claim reception. Thus, we need the following data flows.

**ICP$_{13}$.** Picp.approved.claimNO ⇝ Pcsc.adviseScrap.claimNO

**ICP$_{14}$.** Picp.approved.claimNO ⇝ Pcsc.adviseCash.claimNO

*Capture event orders.* No additional temporal constraints are needed between the messages of *Pcsc* and *Picp.*

4.2.5 *Step M5: Compose new protocols.* No other changes within existing protocols are needed and thus no existing protocols need to be replaced. Axioms **INS$_1$** through **ICP$_{14}$** yield the composite protocol *Icp.*

4.2.6 *Result.* Fig. 11 shows the corresponding composition diagram (omitting INSURED (Id), REPAIRER (Rp), CALL CENTER (Ca), and ASSESSOR (A) for *Picp* to reduce clutter). Notice how Amoeba helps accommodate a business logic change internal to AGFIL across the business process. In practical terms, the commitments involved are not affected. AGFIL must still handle an insured subscriber's claim: instead of repairing the car, AGFIL discharges its commitment by paying off the subscriber.

### 4.3 Structural Change: Outsourcing

A structural change is caused by changes in the participants and their relationships captured via commitments among them. Thus, changes in relationships amount to new commitments among the participants, and the delegation or assignment of the original commitments to new or existing participants. Outsourcing illustrates changes not only to the

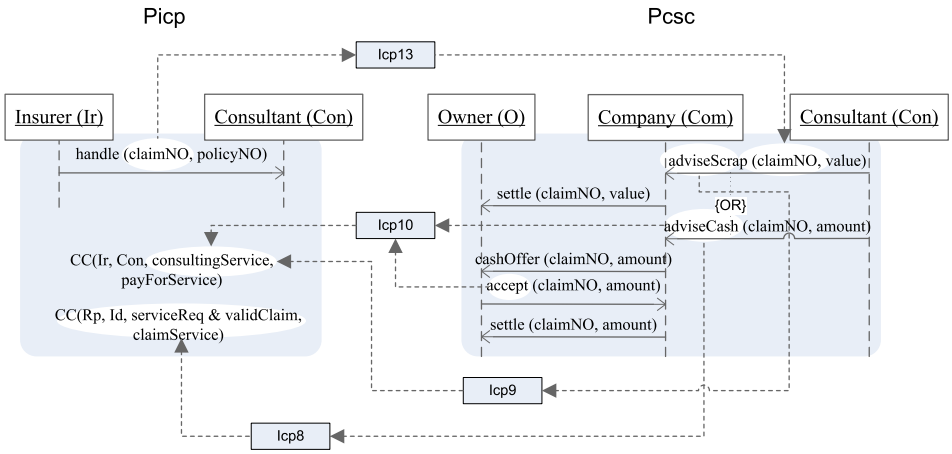


Fig. 11. Accommodating a transactional change by composing *Icp* from *Picp* and *Pcsc*

internal functioning of the outsourcer but also to the interactions involved because a new participant is introduced that interacts with the other existing participants. Insourcing to undo the effects of outsourcing would likewise alter the interactions.

Let's assume AGFIL is operating based on *Bas* (as in Section 3.5). Say AGFIL wishes to outsource the handling of claims to a consulting firm. To outsource the claim handling to a consultant presupposes that AGFIL interacts with the consultants, monitors progress, and makes the necessary decisions. AGFIL may do so by reusing *Mon* of Fig. 9, as specified in Appendix A.4. The COMPANY assigns claims to the CONSULTANT to handle. The CONSULTANT takes the necessary steps and returns with an invoice for repairs. The final decision on whether or not to authorize such repairs is up to the COMPANY. Thus *Mon* and *Bas* can be composed to yield *Out* (Outsourced handling).

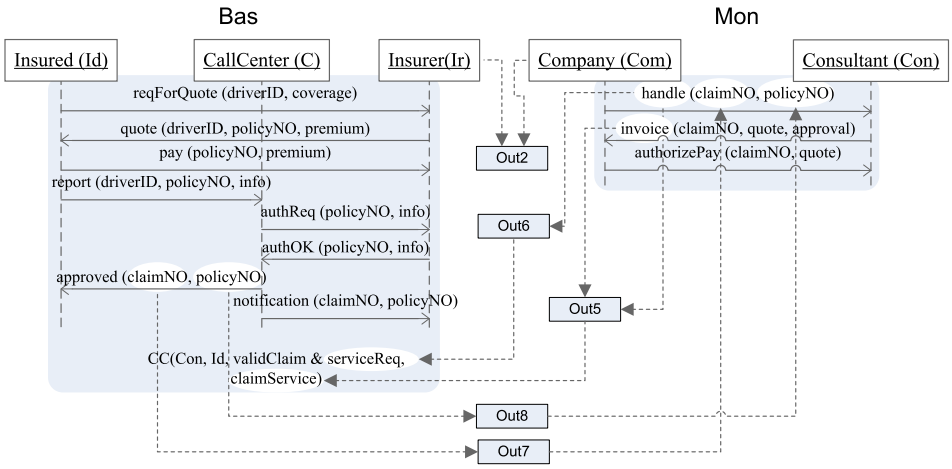


Fig. 12. Accommodating a structural change by composing *Out* from *Mon* and *Bas*

Appendix A.7 describes the composition of *Out* from *Mon* and *Bas*. Fig. 12 shows the corresponding composition diagram. Obviously, a change in the business model is a big shift for an enterprise; new partnerships are formed and new interactions emerge.

### 4.4 Contextual Change: Handling Business Exceptions

A contextual change reflects changes in the legal or other context (such as government regulations) under which the participants interact. The contextual rules can be captured as metacommitments. For example, a legal context requires that a merchant delivers the goods by the deadline the merchant specified in its offer. If the merchant fails to meet this commitment, then the context ensures that the customer's commitment to pay by a deadline is canceled. Such contextual rules enable handling business exceptions elegantly. Exceptions are abnormal conditions arising during a business interaction. In our example, a fraudulent auto-insurance claim can be understood as an exception. Policy holders may file fraudulent claims. AGFIL would need a way to detect such claims and respond appropriately. If the current process model does not accommodate the appropriate treatment of fraudulent claims, it needs to be updated with additional interactions. We classify handling fraud as a contextual change because, due to the surrounding legal framework, fraudulent activity by one party can release another party from its commitments, in essence relieving it from its contractual obligations to the fraudulent party.

4.4.1 *Step M1: Identify new roles and protocols.* Handling fraudulent claims in this setting does not involve additional contracts and does not introduce new participants. However, new roles and a new protocol are needed to capture the change.

Fraudulent claims are detected by the inspectors when they conduct an inspection. Because the inspectors have no direct contract with AGFIL, any interaction triggered by fraud detection must be propagated to AGFIL via Lee CS. AGFIL can then notify the policy holder. Thus, new roles need to be adopted by the policy holder (OWNER), AGFIL (COMPANY), Lee CS (CONSULTANT), and the inspectors (ASSESSOR). The new roles fall into a new protocol, *Fra* (Fraudulent claims detection). Since no new participants are needed, the new roles are fused with the roles of *Picp*, yielding the roles of the composite protocol *Ficp* (Fraud-resistant insurance claim processing) as follows.

**FICP$_1$.** Ficp.Insured $\doteq$ Picp.Insured, Fra.Owner
**FICP$_2$.** Ficp.Insurer $\doteq$ Picp.Insurer, Fra.Company
**FICP$_3$.** Ficp.Consultant $\doteq$ Picp.Consultant, Fra.Consultant
**FICP$_4$.** Ficp.Repairer $\doteq$ Picp.Repairer
**FICP$_5$.** Ficp.CallCenter $\doteq$ Picp.CallCenter
**FICP$_6$.** Ficp.Assessor $\doteq$ Picp.Assessor, Fra.Assessor

4.4.2 *Step M2: Identify changes to contractual relationships.* It is clear that handling this exception involves addressing the distinct goals of detecting it and responding to it. An auto inspector detects the exception, and AGFIL and Lee CS respond to it.

AGFIL responds by canceling the policy coverage of the policy holder and Lee CS responds by releasing the repairers from the commitment to perform repairs (Appendix A.2). The ASSESSOR's detection of fraud counts as an inspection response and thus discharges CC(A, Con, inspectReq, inspectRes) (Appendix A.3). 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 (Fig. 7(a)). The CONSULTANT propagating the detection of fraud to the COMPANY counts as the consulting service being provided.

4.4.3 *Step M3: Modify message meanings.* New messages are needed to propagate information about the detected fraud and to notify the policy holder. In general, we can model two roles apiece for detecting and responding to each exception: one sending a message (of a detected exception or a concomitant response), and the other receiving the message. In our present scenario, the ASSESSOR deals only with the CONSULTANT, and only the COMPANY deals with the OWNER when a fraudulent claim is detected. For this reason, we would need to introduce another message to convey this information from the CONSULTANT to the COMPANY.

The ASSESSOR may send an adviseFraud message to the CONSULTANT who may propagate it as a fraudulent message to the COMPANY. The COMPANY may notify the OWNER of the fraud detection via a fraud message. Each of these messages has a claimNO parameter for correlation. Appendix A.8 specifies *Fra*. The following axioms describe part of the composition of *Fra* and *Picp* into *Ficp*.

**FICP$_7$.** Fra.fraud(claimNO) →
Picp.cancel(Ir, CC(Rp, Id, serviceReq ∧ validClaim, claimService))

**FICP$_8$.** Fra.adviseFraud(claimNO) →
Picp.release(Con, CC(Rp, Con, acceptEstimate(claimNO, price), performRepair(claimNO)))

**FICP$_9$.** Fra.adviseFraud(claimNO) → Picp.inspectRes(claimNO)

**FICP$_{10}$.** Fra.fraudulent(claimNO) → Picp.consultingService(claimNO)

4.4.4 *Step M4: Modify message constraints.*

*Capture data flows among messages.* Clearly, the value of the claimNO parameter in the fraudulent and fraud messages flows from the adviseFraud message. However, because *Fra* does not open a new claim, the value of claimNO in adviseFraud must flow into *Fra* from other protocols. Only the claims approved during the claim reception can go to the inspectors. Thus, the following data flow is identified between *Picp* and *Fra*.

**FICP$_{11}$.** Picp.approved.claimNO ⇝ Fra.adviseFraud.claimNO

*Capture event orders.* No additional temporal constraints are needed between the messages of *Fra* and *Picp*.

4.4.5 *Step M5: Compose new protocols.* No other changes within existing protocols are needed and thus no existing protocols need to be replaced.

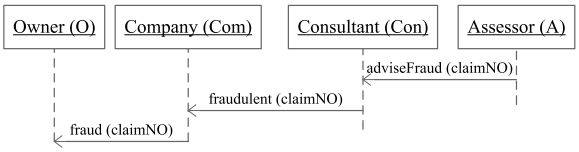


Fig. 13. A scenario of *Fra* (fraudulent claims detection)

4.4.6 *Result.* Fig. 13 illustrates the new protocol *Fra* for interactions relating to detecting frauds. The composition diagram of Fig. 14 shows how a composite protocol *Ficp*

(Fraud-resistant insurance claim processing) is constructed by composing *Picp* and *Fra*. Here INSURED (Id), REPAIRER (Rp), CALL CENTER (Ca), and ASSESSOR (A) are not shown for *Picp*.

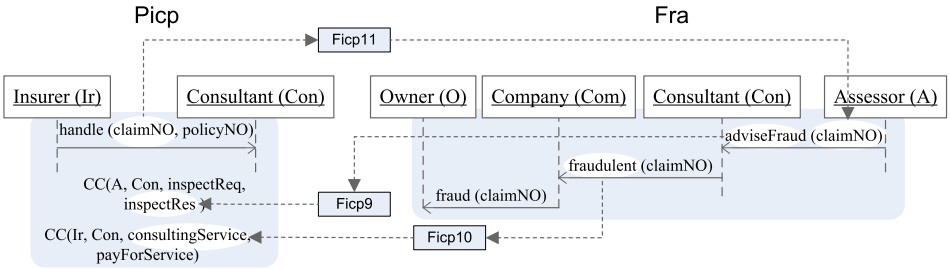


Fig. 14. Accommodating a contextual change: Incorporating handling of fraudulent claims

Notice how the resulting interaction supports AGFIL ending the process by canceling and releasing the commitments in case of fraud. Simply adjusting the commitments can yield greater flexibility while enabling the parties to continue to interoperate. Thus, fraud, which is an exception, is handled by applying contextual rules to adjust the commitments—AGFIL cancels its commitment to handle an otherwise insured subscriber's claim.

## 5. CASE STUDY: AEROSPACE AFTERMARKET SERVICES

To evaluate Amoeba further, we apply it to the modeling and evolution of cross-organizational processes developed under the European Union CONTRACT project [van Aart et al. 2007] in the domain of aerospace aftermarket services.

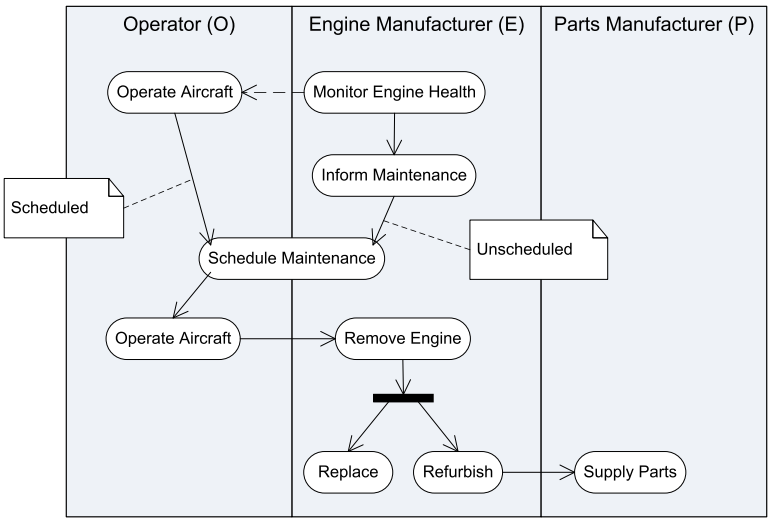


Fig. 15. A high-level model of the aerospace aftermarket process (verbatim from the CONTRACT project [van Aart et al. 2007])

Fig. 15 shows a high-level flow of a process in aerospace aftermarket services. This process involves three parties: an operator (i.e., an airline), an aircraft engine manufacturer, and a parts manufacturer. The engine manufacturer provides the required number of serviceable engines to keep the airline operator's aircrafts flying. The engine manufacturer is paid by the hour when the engines are available and suffers a penalty when planes are on the ground waiting for a serviceable engine. The operator regularly supplies engine health data to the manufacturer. Based on an analysis of the data, the manufacturer informs the operator of any required engine maintenance. Alternatively, the operator proactively requests maintenance. In the aerospace industry jargon, these are termed *unscheduled* and *scheduled* maintenance, respectively. In either case, the operator and the manufacturer schedule a time and place for the engine to be serviced. The manufacturer may either replace the engine or refurbish it. The engine manufacturer maintains a pool of serviceable engines via contracts with one or more parts manufacturers, who supply individual engine parts.

### 5.1 Modeling

Performing Amoeba steps M1–M5 of Section 3 yields several protocols (as in Figs. 16) and contractual relationships (as in 17). The contract between the operator and the manufacturer is created in *Msc*. A similar protocol (not shown in Fig. 16) would create the contract between the engine manufacturer and the parts manufacturer. Fig. 17 shows the contractual relationships that hold after *Msc* is added to the process model. No other contractual relationships need be assumed. The remaining protocols simply detach or discharge the commitments created in *Msc*. Message annotations within square brackets show the commitment conditions being brought about.

Let us observe a few important points about the commitments and the protocols of Figs. 16 and 17. In *Msc*, the creation of $C_1$ and $C_2$ would detach and discharge $C_0$, respectively. Also, as in $\textsc{Ins}_3$, as long as $C_1$ is active, $C_4$ and $C_5$ may be created and discharged multiple times. Similarly, $C_2$ keeps creating $C_3$ and $C_6$. In *Pma*, both refurbishEngine and replaceEngine count as serviceInTime. If the service was delayed, the operator would pay for the service and the manufacturer would pay the penalty for being delayed. The penalty for a delayed parts consignment in *Spa* is modeled similarly.

### 5.2 Evolution

We applied Amoeba to accommodate requirements changes in the aerospace aftermarket process. As in the insurance scenario, we consider a transactional, a structural, and a contextual change. For each change, we now describe the additional requirements and how to accommodate them. The following assumes a protocol *Ams* (Aerospace aftermarket services) defined as a composition of *Msc*, *Sma*, *Uma*, *Pma*, *Opa*, and *Spa*. The engine manufacturer plays CONSUMER in *Spa* and BUYER in *Opa*. The parts manufacturer plays SUPPLIER in *Spa* and SELLER in *Opa*. The other role identifications are self-explanatory.

5.2.1 *Transactional Change*. Flights are commonly delayed due to bad weather. In *Sma* and *Uma*, the operator and the manufacturer have agreed on an airport and a time for performing maintenance on a particular engine. It is quite possible that the aircraft with the engine to be serviced is delayed and is not available for service. As the weather improves, a large number of aircraft become available for service, possibly from different operators. This overloads the maintenance resources of the manufacturer, who thus ends up

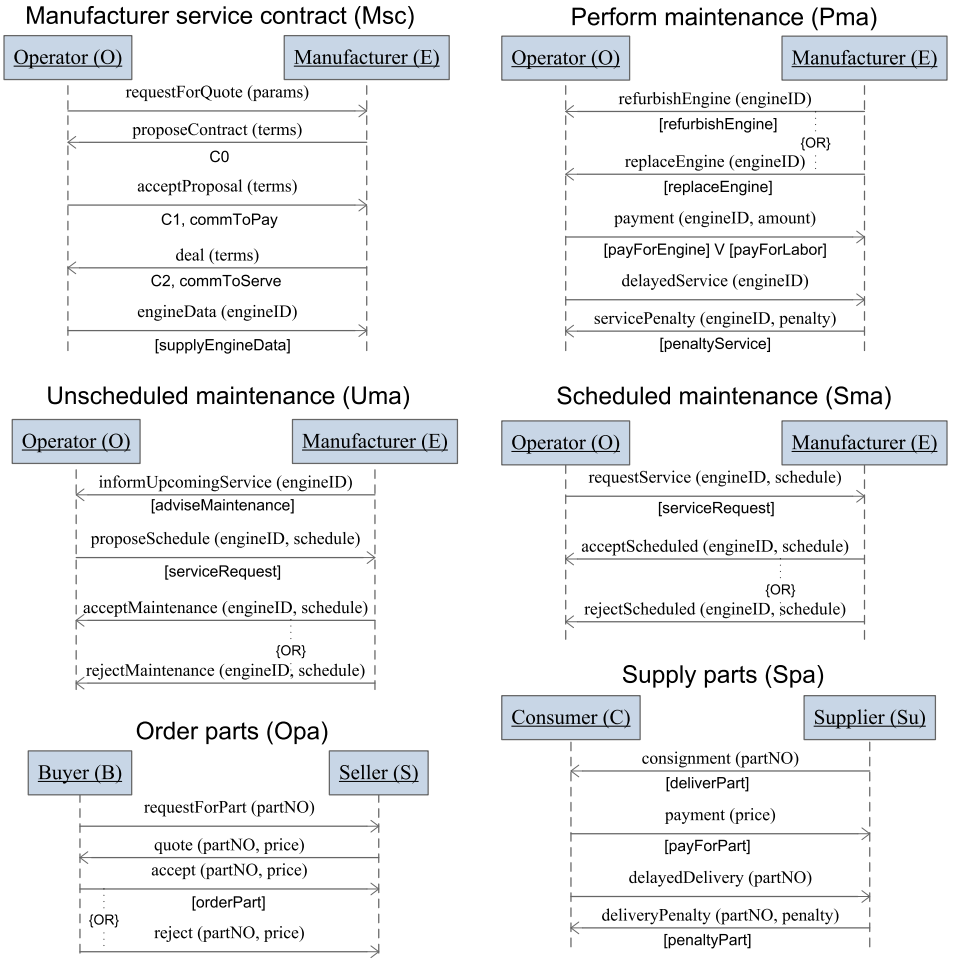


Fig. 16. Protocols in the aerospace aftermarket business process

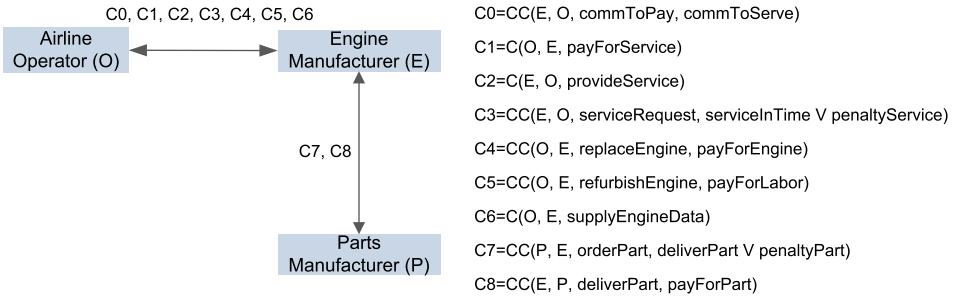


Fig. 17. The contractual relationships with *Msc* in the process model

paying penalties for service delays. To amend this situation, the manufacturer negotiates new conditions for its service contracts with the operators such that if the engine is not available at the agreed upon schedule, then the delay penalty is waived.

Following steps M1–M5 in Section 4.1, a new protocol *Wpe* (Waive penalty) is introduced wherein the manufacturer signals that an engine is unavailable to the operator. In the composition of *Wpe* with *Ams*, the effect of the "engine unavailable" message is captured via a message axiom that cancels the commitment to pay the penalty for delayed service.

5.2.2 *Structural Change.* The engine manufacturer decides to focus on its core competency of building and diagnosing engines and decides to outsource to a service company the work of performing timely maintenance on the various engines and aircraft.

Following steps M1–M5 in Section 4.1, a new partnership between the manufacturer and the service company is formed and a new protocol *Scm* (Subcontracted maintenance) is introduced wherein the manufacturer delegates $C_3$ to the service company and pays the service company for maintenance services in return. As a new interaction, the manufacturer informs the service company of the schedules of maintenance that it has agreed upon. In role identifications, the service company would play MANUFACTURER in *Pma*. (The role name may be changed from MANUFACTURER to SERVER to reflect the new usage.)

5.2.3 *Contextual Change.* A regulatory agency, such as the US FAA (Federal Aviation Administration) or similar, changes its safety regulations. Under the new regulations, in order to ensure safety, all engine manufacturers must be licensed by the agency. Also, an agency official may inspect aircraft at any time without prior warning, and may suspend or revoke the license of any manufacturer whose engine is found to perform inadequately.

Following steps M1–M5 in Section 4.1, we assume a commitment from the MANUFACTURER to the OPERATOR that the MANUFACTURER holds a valid license whenever it services an aircraft. Also, we introduce a new protocol *Ias* (Inspect aircraft safety) capturing the inspection interactions between the REGULATOR, the OPERATOR, and the MANUFACTURER. If the REGULATOR finds the MANUFACTURER has not informed the OPERATOR of a required maintenance of an engine it judges to defective, the REGULATOR suspends the license of the MANUFACTURER, releases the service company from its commitments, and cancels $C_1$, $C_4$, and $C_5$. This effectively ends the contracts between the MANUFACTURER and the OPERATOR.

## 6. DISCUSSION AND FUTURE WORK

Amoeba facilitates modeling cross-organizational business processes via commitment-based business protocols. The various requirements changes result in changes to the roles and the protocols in which they participate. The process model at each stage is driven by the business meanings of the interactions among the participants. The above case studies provide evidence of Amoeba's usefulness in business process modeling, especially in the face of evolving requirements.

A dominant paradigm for service-oriented business process modeling today is orchestration, as epitomized by the Business Process Execution Language (BPEL) [2007]. In orchestration, a process is represented from the perspective of a central engine that invokes various services and sets up desired data and control flows among them. Business processes have traditionally been modeled as workflows, and BPEL reflects this legacy. However, workflows inadequately model interactions, and cannot properly handle cross-

organizational settings in which the autonomy of the participants is crucial [Bussler 2001].

In contrast with orchestration, the emerging choreography approaches support a peer-to-peer metaphor for business processes. Choreography efforts include the Electronic Business Extensible Markup Language (ebXML) Business Process Specification Schema (BPSS) recently standardized by OASIS [ebBP 2006] and the Web Services Choreography Description Language being considered for recommendation by the W3C [WS-CDL 2005]. Because choreographies explicitly accommodate autonomous participants, they more readily support cross-organizational processes than orchestration does.

Existing orchestration and choreography approaches, even if formal, lack an appropriate encoding of *business* meaning. Traditional semantics reflects the occurrence and ordering of tasks (units of orchestrations) or messages (units of choreographies), but fails to identify the business interactions that these support. For example, they would specify that a quote message can be followed by an acceptance or a rejection message, but would ignore the business commitment that an acceptance of the quote creates. This limitation becomes all the more pronounced under requirements evolution, because absent a business meaning, there is no principled basis for validating a business process or modifying it in a reliable manner.

### 6.1 Treatment of Requirements in Agent-Oriented Software Engineering

Like existing agent-oriented software engineering (AOSE) methodologies [Bergenti et al. 2004; Henderson-Sellers and Giorgini 2005], Amoeba addresses the challenges of autonomy and heterogeneity. Amoeba complements existing AOSE methodologies by concentrating on requirements evolution, which they deemphasize. Also, unlike other AOSE methodologies, Amoeba provides guidelines for reverse engineering traditionally modeled processes. In principle, Amoeba could be incorporated into existing AOSE methodologies by enhancing them with protocols as reusable artifacts based on commitments.

Recent years have seen the emergence of common evaluation criteria for agent-oriented methodologies. Important evaluation efforts include those by Dam and Winikoff [2004] (of MaSE, Prometheus, and Tropos); Sturm and Shehory [2004] (of Gaia, Tropos, and MaSE); Tran and Low [2005] (of ten methodologies). Sudeikat *et al.* [2004] frame comparisons relative to a target platform. These studies have identified key criteria including concepts, modeling techniques, development processes, and tool support for methodologies. Table II applies these criteria to Amoeba.

Importantly, the vital criterion of handling requirements evolution does not feature in the existing studies. An evaluation with respect to requirements evolution would consider the guidance provided by a methodology in (and the complexity of) updating models for specific changes in interaction requirements. Table III compares Amoeba with Tropos, Gaia, Prometheus, and MaSE under this criterion.

Most of the current AOSE methodologies agree on protocols as reusable message patterns among roles. The key commonly missing pieces are a business-level abstraction such as commitments (or something analogous) and a mechanism for composing protocols as a way of accommodating requirements change. Below, we reflect on what it would take to accommodate changing requirements in Tropos and Gaia. In the interest of brevity, we omit similar arguments that can be made about the other methodologies.

Table II. Amoeba evaluated with respect to the established criteria for agent-oriented methodologies

| Criterion | Evaluation |
|---|---|
| **Concepts** | *Agent:* roles in protocols are adopted by agents |
| | *Role:* interactions are described among roles |
| | *Message:* roles interact via messages |
| | *Protocol:* logically related messages are grouped into protocols |
| **Properties** | *Proactivity:* agent business logics can be proactive |
| | *Reactivity:* Agents react to events according to the protocol |
| | *Sociality:* roles interact and create (social) commitments |
| | *Autonomy:* agents adopting roles are autonomous and are constrained only by their commitments |
| **Model Properties** | *Analyzability:* the specifications can be analyzed |
| | *Abstraction:* three levels: commitments, protocols, agents enacting protocols |
| | *Precision:* unambiguous due to formal specifications |
| | *Expressiveness:* due to formal representation |
| | *Modularity:* protocols are modular and can be composed |
| | *Testability:* implementations can be tested |
| **Development Process** | Guidelines and steps for analysis, design, reverse engineering, and requirements evolution |
| **Tools** | Tools supporting some Amoeba steps exist [Desai et al. 2005] |
| | *Composition:* given a set of protocols to compose and composition axioms, generate the composite protocol |
| | *Model generation:* Generate all possible conversations supported by a protocol specification |
| | *Skeleton generation:* Generate role skeletons from protocols |
| | *Enactment:* Augment role skeletons with business logic and generate implementations for roles, e.g., using messaging middleware |

*Requirements Evolution in Tropos.* In contrast with object-oriented models, Yu argues for an emphasis on the intentional aspects of actor relationships for early-phase requirements engineering [1996], which is in line with the notion of commitments in Amoeba. Tropos builds on Yu's approach. It employs the abstractions of goals, dependencies, and organizations to capture requirements. Transactional changes would correspond to newer subgoals, tasks, and resources for fulfilling goals. Structural changes would alter the actor diagram of the corresponding organization via changes in the stakeholders. Contextual changes would be accommodated via modeling the context as an actor. Even assuming that a methodology for accommodating such changes is available, Tropos offers limited support for such changes. Although prominent in the early and late requirements phases, the autonomous actors disappear in the detailed design phase and only a single information system actor remains. Thus a centralized information system is designed instead of one information system for each actor. Thus, Tropos seems to be targeted at applications where the actors collaborate via a central information system.

Also, mapping the dependencies to interactions is nontrivial; Mallya and Singh [2006] propose a set of guidelines to this end. Compared to commitments, goals deemphasize the participants' autonomy. For example, agents would not normally be able to manipulate the goals of other agents. However, delegation and assignment of commitments are quite common in business service engagements and hence it is crucial to support such operations. Lastly, it is nontrivial to map any changes to the models in the early phases to models in the later phases. For example, how would a new goal in the actor diagram affect the

means-ends analysis? Hence, Tropos would benefit from a decentralized perspective with an abstraction of commitments and guidelines for accommodating and tracing the changes in the various phases of the methodology.

*Requirements Evolution in Gaia.* Unlike Tropos, the newer versions of Gaia explicitly target open environments such as marketplaces. Thus, each agent adopts roles and has an independent interaction model instead of a centralized model for all agents. Explicit models of the environment and the organizations make it easier to accommodate structural and contextual changes. Gaia gives *responsibilities* first-class status, but describes them in terms of procedural coordination rather than declarative (contractual or goal-based) requirements. Thus, Gaia lacks a direct high-level abstraction to capture a transactional change. Gaia would benefit from role schemas described in terms of commitments being created and manipulated. Our technique of protocol composition can be readily adapted for Gaia protocols. Methodologies based on Gaia, such as ROADMAP [Juan et al. 2002], explicitly model goals in role schemas and are a step in the right direction. However, the arguments about goals versus commitments in Tropos also apply to ROADMAP.

Table III. Methodologies evaluated relative to requirements evolution

| Methodology | Evaluation |
|---|---|
| **Tropos** [Bresciani et al. 2004] | Requirements changes would map to the exclusion or inclusion of new actors, changed dependencies between actors, changed goals, and changed ways to achieve the goals in the early and late requirements analysis phases |
| **Gaia** [Juan et al. 2002; Zambonelli et al. 2003] | Requirements changes would cut across the analysis and architectural design phases with possible changes to the environmental, role, and interaction models; organizational rules; and organizational structure |
| **Prometheus** [Padgham and Winikoff 2005] | Requirements changes would result in changes in scenarios, goals, and potentially changes in interaction protocols and functionalities |
| **MaSE** [DeLoach 2004] | Requirements changes would cut across goal hierarchy, use cases, sequence diagrams, role models, and conversation diagrams |
| **Amoeba** | Requirements changes yield changes in roles and changed ways to fulfill commitments that are captured via composition axioms |

A crucial distinguishing feature of Amoeba is its foundation in commitments. Considering commitments explicitly enables us to give processes a suitable and precise business meaning, which is lacking in most existing approaches. Commitments are valuable because they support both flexibility and compliance: agents may flexibly choose their actions as long as they comply with their commitments. Traditional, low-level representations support compliance but only at the cost of flexibility. Winikoff [2007] concurs that conventional message-oriented protocols lack business meaning, thus leading to rigid implementations. Winikoff models commitments between agents to yield agent implementations whose flexibility derives from being able to plan. In a similar vein, Cheong and Winikoff [2009] advocate declaratively specifying interactions for achieving greater flexibility. They employ interaction goals, which can be understood in terms of social commitments, to drive the design of protocols. Narendra and Orriëns [2007] also use commitments to express functional requirements from which they derive service compositions.

Maamar *et al.* [2007] propose interactions to support design and development of composite services. They separate service interactions into two layers: business logic and support, which is similar in spirit to how Amoeba separates protocols from the business logics of the interacting participants. These recent works signal an increased interest in interactions and a recognition of commitments as a natural abstraction to capture the essence of the interactions among autonomous parties.

Kongdenfha *et al.* [2006] outline a taxonomy of possible adaptations that (business process) services might undergo to accommodate clients, but these adaptations are at the level of messages. For instance, a service might require only one of the many messages that a client sends to achieve a business functionality (*message merge*); another kind involves adaptation of the type of an incoming message to a type understood by a service (*type mismatch*). The adaptations themselves are achieved using aspect-oriented programming (AOP) in the services. It would be interesting to see such adaptations analyzed at the level of commitments. For example, a customer may discharge a commitment to make a payment of $100 in five steps of $20 each. Either AOP or Winikoff's planning-based approach may be used for implementing such agents.

Service composition has been extensively studied. The orchestration approaches discussed above mix interactions with local business logic, complicating the reuse and adaptation of interactions. OWL-S [DAML-S 2002], which includes a process model for Web services, facilitates dynamic composition. The Semantic Web Services Framework (SWSF) [2005] proposes an expressive formal language and an ontology for modeling services. Similarly Web Services Modeling Ontology (WSMO) [2004] employs a formal language and an ontology for modeling the behavior of services and of the mediators that facilitate interoperability among services. However, dynamic composition presumes perfect markup and an ontological matching of the services being composed. The semantic web services approaches focus on the representation and reasoning about services rather than on methodologies for the evolution of requirements. By contrast, Amoeba seeks only to guide a human designer in composing services. Semantic Annotations for WSDL (SAWSDL) [2007] is an approach for annotating and reasoning about the input, output, and metadata of Web services. Amoeba can benefit from SAWSDL for handling the data heterogeneity of protocols.

Vitteau and Huget [2004] compose protocols from microprotocols in a bottom-up manner, but only in limited ways, and do not support interleaving protocols. Mazouzi *et al.* [2002] compose protocols in a top-down manner by associating refinements with abstract transitions in Colored Petri Nets, but also do not support interleaving. By contrast, Amoeba provides composition axioms that offer indirection, which enables us to arbitrarily compose protocols including by interleaving them.

OMG's Model-Driven Architecture (MDA) [OMG 2006] promotes three kinds of models (from higher to lower levels of abstraction): computation independent, platform independent, and platform dependent. In MDA, development proceeds by transforming higher to lower models. Amoeba operates on protocol-based, i.e., computation-independent, models, thus specializing MDA for cross-organizational business processes. Krüger *et al.* [2006] motivate interaction patterns as first-class modeling elements for all levels of abstraction in MDA. They treat interaction patterns as aspects and explore the space of suitable service-oriented architectures. Being composable, protocols are similar in spirit to aspects albeit with a business-level focus. Thus, the techniques proposed by Krüger *et al.*

can be extended for cross-organizational engagements by adopting protocols as aspects.

### 6.2 Software Requirements Evolution

The software engineering community has long studied the challenges of handling evolving requirements. The main distinguishing aspect of Amoeba with respect to this body of work is that we focus on business-level requirements pertaining to interactions among organizations. Existing approaches differ in the style of handling requirements evolution. Zowghi and Offen [1997] propose a logical framework for reasoning about requirements evolution. They employ nonmonotonic reasoning and belief revision to relate successively refined requirement models, each expressed in a nonmonotonic theory. Etien and Salinesi [2005] address the evolution of requirements where a change impacts multiple aspects of a system, such as its teams, engineering domains, viewpoints, or components. The main challenge they address is that of maintaining consistency between the various aspects as they evolve together. Of the three styles of handling evolution of requirements that Etien and Salinesi describe, our approach addresses requirements evolution explicitly. However, unlike Amoeba, neither of the above works focus on business-level interaction requirements and it is not clear how they would treat these as first-class requirements. Unlike in our approach, they present neither a methodology nor any case studies.

Requirements evolution has been studied from the nonfunctional requirements perspective. Chung *et al.* [1995] propose a model in which nonfunctional requirements are represented as goals, which may be decomposed and analyzed in relation with one other. In addition, the model may be systematically annotated with design decisions and rationales. An important feature of Chung *et al.*'s model is that it captures the history of the evolving requirements. Cleland-Huang *et al.* [2005] propose an approach for understanding the impact of a functional requirements change on nonfunctional requirements. They capture nonfunctional requirements and their interdependencies via a soft-goal interdependency graph. Cleland-Huang *et al.* use a probabilistic model to identify subgraphs affected by a change. Although our emphasis is on business-level cross-organizational requirements rather than nonfunctional intraorganizational requirements, Amoeba can benefit both from maintaining the history of the evolving requirements and from understanding the relationship between the functional and the nonfunctional requirements.

Other approaches emphasize different perspectives on requirements. Lormans [2007] proposes a system for tracing requirements into other products of the software development life cycle as a means of monitoring and managing requirements evolution. For end users, the system presents different *views* on requirements, depending on the perspective selected. Lam and Loomes [1998] propose a conceptual model for requirement evolution, and a process that uses this model in analyzing changes. In particular, they classify requirements changes into four categories: environment, (functional) requirements, viewpoint, and design. These roughly correspond to the three classes considered in this paper. However, Lam and Loomes neither provide modeling abstractions nor comprehensive case studies. Anderson and Felici [2001] argue for a product-oriented instead of a process-oriented methodology for requirements evolution. They seek to characterize industrial settings so that specific methodologies can be developed taking product features into account. Anderson and Felici describe case studies where the requirements are related not to business contracts but to safety critical features in avionics and to security in smart cards. Also, in all of these works, the focus is on the requirements of a single software project rather than of a cross-organizational information system.

### 6.3 Future Work

Amoeba introduces key computational abstractions and primitives with which to model processes and their adaptations. This opens up a fruitful line of research for agent-oriented software engineering and more broadly for service-oriented computing. A comprehensive tool-suite to support Amoeba is essential. With tools, Amoeba can be employed in practical settings and can be applied to handle requirements changes in the context of various real-life processes. Such studies could uncover additional properties, benefits, and potential pitfalls in the methodology.

#### ACKNOWLEDGMENTS

We thank Nanjangud Narendra, Michael Winikoff, Pınar Yolum, and the anonymous reviewers for helpful comments on previous versions of this paper. This research was partially supported by the National Science Foundation under grant IIS-0139037 and by a gift from Intel. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the sponsors.

#### REFERENCES

ANDERSON, S. AND FELICI, M. 2001. Requirements evolution from process to product oriented management. In *Proceedings of the International Conference on Product Focused Software Process Improvement*. 27–41.

BERGENTI, F., GLEIZES, M.-P., AND ZAMBONELLI, F., Eds. 2004. *Methodologies and Software Engineering for Agent Systems*. Kluwer, Boston.

BPEL. 2007. Web services business process execution language, version 2.0. http://docs.oasis-open.org/wsbpel/2.0/.

BRESCIANI, P., PERINI, A., GIORGINI, P., GIUNCHIGLIA, F., AND MYLOPOULOS, J. 2004. Tropos: An agent-oriented software development methodology. *Journal of Autonomous Agents and Multi-Agent Systems 8,* 3 (May), 203–236.

BROWNE, S. AND KELLETT, M. 1999. Insurance (motor damage claims) scenario. Document Identifier D1.a, CrossFlow Consortium.

BUSSLER, C. 2001. The role of B2B protocols in inter-enterprise process execution. In *Proceedings of the 2nd International Workshop on Technologies for E-Services*. Lecture Notes in Computer Science, vol. 2193. Springer-Verlag, 16–29.

CCALC. 2004. The causal calculator CCalc. http://www.cs.utexas.edu/users/tag/cc/.

CHEONG, C. AND WINIKOFF, M. 2009. Hermes: Designing flexible and robust agent interactions. In *Multi-Agent Systems: Semantics and Dynamics of Organizational Models*, V. Dignum, Ed. IGI, Chapter 5. To appear.

CHUNG, L., NIXON, B. A., AND YU, E. 1995. Using non-functional requirements to systematically support change. In *Proceedings of the IEEE International Symposium on Requirements Engineering*. 132–139.

CLELAND-HUANG, J., SETTIMI, R., BENKHADRA, O., BEREZHANSKAYA, E., AND CHRISTINA, S. 2005. Goal-centric traceability for managing non-functional requirements. In *Proceedings of the International Conference on Software Engineering*. 362–371.

DAM, K. H. AND WINIKOFF, M. 2004. Comparing agent-oriented methodologies. In *Agent-Oriented Information Systems*, P. Giorgini, B. Henderson-Sellers, and M. Winikoff, Eds. Vol. 3030. Springer-Verlag, 78–93.

DAML-S. 2002. DAML-S: Web service description for the semantic Web. In *Proceedings of the 1st International Semantic Web Conference (ISWC)*. Lecture Notes in Computer Science, vol. 2342. Springer-Verlag, 348–363. Authored by the DAML Services Coalition, which consists of (alphabetically) Anupriya Ankolekar, Mark Burstein, Jerry R. Hobbs, Ora Lassila, David L. Martin, Drew McDermott, Sheila A. McIlraith, Srini Narayanan, Massimo Paolucci, Terry R. Payne and Katia Sycara.

DELOACH, S. A. 2004. The MaSE methodology. In *Methodologies and Software Engineering for Agent Systems*, F. Bergenti, M.-P. Gleizes, and F. Zambonelli, Eds. Kluwer, Boston, Chapter 6, 107–126.

DESAI, N., CHOPRA, A. K., AND SINGH, M. P. 2006. Business process adaptations via protocols. In *Proceedings of the 3rd IEEE International Conference on Services Computing (SCC)*. IEEE Computer Society Press, Los Alamitos, 103–110.

DESAI, N., CHOPRA, A. K., AND SINGH, M. P. 2007. Representing and reasoning about commitments in business processes. In *Proceedings of the 22nd Conference on Artificial Intelligence (AAAI)*. 1328–1333.

DESAI, N., MALLYA, A. U., CHOPRA, A. K., AND SINGH, M. P. 2005. Interaction protocols as design abstractions for business processes. *IEEE Transactions on Software Engineering 31,* 12 (Dec.), 1015–1027.

DESAI, N. AND SINGH, M. P. 2007. A modular action description language for protocol composition. In *Proceedings of the 22nd Conference on Artificial Intelligence (AAAI)*. 962–967.

DESAI, N. AND SINGH, M. P. 2008. On the enactability of business protocols. In *Proceedings of the 23rd Conference on Artificial Intelligence (AAAI)*. 1126–1131.

EBBP. 2006. Electronic business extensible markup language business process specification schema v2.0.4. docs.oasis-open.org/ebxml-bp/2.0.4/OS/.

ETIEN, A. AND SALINESI, C. 2005. Managing requirements in a co-evolution context. In *Proceedings of the International Conference on Requirements Engineering*. 125–134.

GIUNCHIGLIA, E., LEE, J., LIFSCHITZ, V., MCCAIN, N., AND TURNER, H. 2004. Nonmonotonic causal theories. *Artificial Intelligence 153,* 1-2, 49–104.

HARKER, S. D. P. AND EASON, K. D. 1993. The change and evolution of requirements as a challenge to the practice of software engineering. In *Proceedings of the IEEE International Symposium on Requirements Engineering*. 266–272.

HENDERSON-SELLERS, B. AND GIORGINI, P., Eds. 2005. *Agent-Oriented Methodologies*. Idea Group, Hershey, PA.

JUAN, T., PEARCE, A., AND STERLING, L. 2002. ROADMAP: extending the Gaia methodology for complex open systems. In *Proceedings of the 1st International Joint conference on Autonomous Agents and Multiagent Systems*. ACM Press, New York, 3–10.

KONGDENFHA, W., SAINT-PAUL, R., BENATALLAH, B., AND CASATI, F. 2006. An aspect-oriented framework for service adaptation. In *Proceedings of the 4th International Conference on Service Oriented Computing*. ACM Press, New York, 15–26.

KRÜGER, I. H., MATHEW, R., AND MEISINGER, M. 2006. Efficient exploration of service-oriented architectures using aspects. In *Proceeding of the 28th International Conference on Software Engineering*. IEEE Computer Society, Los Alamitos, 62–71.

LAM, W. AND LOOMES, M. 1998. Requirements evolution in the midst of environmental change: A managed approach. In *Proceedings of the Euromicro Conference on Software Maintenance and Reengineering*. 121–127.

LORMANS, M. 2007. Monitoring requirements evolution using views. In *Proceedings of the European Conference on Software Maintenance and Reengineering*. 349–352.

MAAMAR, Z., BENSLIMANE, D., AND SHENG, Q. Z. 2007. Towards a two-layered framework for managing web services interaction. In *Proceedings of the 6th IEEE/ACIS International Conference on Computer and Information Science*. IEEE Computer Society, Los Alamitos, 87–92.

MALLYA, A. U. AND SINGH, M. P. 2006. Incorporating commitment protocols into Tropos. In *Proceedings of the International Workshop on Agent Oriented Software Engineering*, J. P. Müller and F. Zambonelli, Eds. LNCS, vol. 3950. Springer-Verlag, 69–80.

MAZOUZI, H., SEGHROUCHNI, A. E. F., AND HADDAD, S. 2002. Open protocol design for complex interactions in multi-agent systems. In *Proceedings of the 1st International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*. ACM Press, New York, 517–526.

NARENDRA, N. C. AND ORRIËNS, B. 2007. Modeling web service composition and execution via a requirements-driven approach. In *Proceedings of the ACM Symposium on Applied Computing*. ACM Press, New York, 1642–1648.

OMG. 2006. The Object Management Group's Model Driven Architecture (MDA). http://www.omg.org/mda/.

PADGHAM, L. AND WINIKOFF, M. 2005. Prometheus: A practical agent-oriented methodology. In *Agent-Oriented Methodologies*, B. Henderson-Sellers and P. Giorgini, Eds. Idea Group, Hershey, PA, Chapter 5, 107–135.

SAWSDL. 2007. Semantic Annotations for WSDL–SAWSDL. http://www.w3.org/2002/ws/sawsdl/.

SINGH, M. P. 1999. An ontology for commitments in multiagent systems: Toward a unification of normative concepts. *Artificial Intelligence and Law 7*, 97–113.

SINGH, M. P., CHOPRA, A. K., AND DESAI, N. 2009. Commitment-based SOA. *IEEE Computer 42*. To appear. Draft available at http://www.csc.ncsu.edu/faculty/mpsingh/papers/.

SINGH, M. P. AND HUHNS, M. N. 2005. *Service-Oriented Computing: Semantics, Processes, Agents*. John Wiley & Sons, Chichester, UK.

SMITH, H. AND FINGAR, P. 2002. *Business Process Management: The Third Wave*. Megan-Kiffer Press, Tampa.

STURM, A. AND SHEHORY, O. 2004. A comparative evaluation of agent-oriented methodologies. In *Methodologies and Software Engineering for Agent Systems*, F. Bergenti, M.-P. Gleizes, and F. Zambonelli, Eds. Kluwer, Boston, Chapter 7, 127–150.

SUDEIKAT, J., BRAUBACH, L., POKAHR, A., AND LAMERSDORF, W. 2004. Evaluation of agent-oriented software methodologies: Examination of the gap between modeling and platform. In *Agent-Oriented Software Engineering*, P. Giorgini, J. P. Müller, and J. Odell, Eds. LNCS, vol. 3382. Springer Verlag, 126–141.

SWSF COMMITTEE. 2005. SWSF: Semantic web services framework (W3C submission). http://www.daml.org/services/swsf/.

TRAN, Q.-N. N. AND LOW, G. C. 2005. Comparison of ten agent-oriented methodologies. In *Agent-Oriented Methodologies*, B. Henderson-Sellers and P. Giorgini, Eds. Idea Group, Hershey, PA, Chapter 12, 341–367.

VAN AART, C. J., CHABERA, J., DEHN, M., JAKOB, M., NAST-KOLB, K., SMULDERS, J. L. C. F., STORMS, P. P. A., HOLT, C., AND SMITH, M. 2007. Usecase outline and requirements. Document Identifier D6.1, IST-CONTRACT Project. http://tinyurl.com/6adejz.

VITTEAU, B. AND HUGET, M.-P. 2004. Modularity in interaction protocols. In *Advances in Agent Communication*, F. Dignum, Ed. LNCS, vol. 2922. Springer-Verlag, 291–309.

WINIKOFF, M. 2007. Implementing commitment-based interaction. In *Proceedings of the 6th International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*. International Foundation for Autonomous Agents and MultiAgent Systems, Columbia, SC, 868–875.

WINIKOFF, M., LIU, W., AND HARLAND, J. 2005. Enhancing commitment machines. In *Proceedings of the 2nd International Workshop on Declarative Agent Languages and Technologies (DALT)*. LNAI, vol. 3476. Springer-Verlag, 198–220.

WOOLDRIDGE, M. 2002. *An Introduction to MultiAgent Systems*. John Wiley & Sons.

WS-CDL. 2005. Web services choreography description language version 1.0. www.w3.org/TR/ws-cdl-10/.

WSMO COMMITTEE. 2004. WSMO: Web services modeling ontology. http://www.wsmo.org/TR/d2/v1.2/.

YOLUM, P. AND SINGH, M. P. 2002. Flexible protocol specification and execution: Applying event calculus planning using commitments. In *Proceedings of the 1st International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*. ACM Press, New York, 527–534.

YU, E. S.-K. 1996. Modelling strategic relationships for process reengineering. Ph.D. thesis, University of Toronto, Toronto, Canada.

ZAMBONELLI, F., JENNINGS, N. R., AND WOOLDRIDGE, M. 2003. Developing multiagent systems: The Gaia methodology. *ACM Transactions on Software Engineering Methodology 12,* 3, 317–370.

ZOWGHI, D. AND OFFEN, R. 1997. A logical framework for modeling and reasoning about the evolution of requirements. In *Proceedings of the IEEE International Symposium on Requirements Engineering*. 247–257.

## A. PROTOCOL LISTINGS: MAY BE PLACED ONLINE

This section presents complete specifications of the protocols referenced above.

### A.1 Claim Reception and Verification (*Rec*)

A scenario of *Rec* is shown in Fig. 6. $\mathbf{REC}_2$ states that the CALL CENTER sending authReq should count as meeting the antecedent of the assumed commitment of $\mathbf{REC}_1$. Both the success and failure of authentication count as authentication responses ($\mathbf{REC}_3$ and $\mathbf{REC}_4$). When parameters are omitted, all parameters of the message on the left are bound to the corresponding parameters of the message on the right.

$\mathbf{REC}_1$. start → CC(P, C, reqAuth(claimNO, policyNO), authResponse(claimNO, policyNO))

$\mathbf{REC}_2$. authReq(claimNO, policyNO, info) → reqAuth(claimNO, policyNO)

**REC$_3$.** authOK(claimNO, policyNO) → authResponse(claimNO, policyNO)
if authReq(claimNO, policyNO, info)

**REC$_4$.** authNOK(claimNO, policyNO) → authResponse(claimNO, policyNO)
if authReq(claimNO, policyNO)

**REC$_5$.** report.policyNO ⇝ authReq.policyNO

**REC$_6$.** authReq ⇝ authOK

**REC$_7$.** authOK ⇝ approved

**REC$_8$.** authReq ⇝ authNOK

**REC$_9$.** authNOK ⇝ denied

**REC$_{10}$.** approved ⇝ notification

### A.2 Administering Repairs (*Rep*)

A scenario of *Rep* is shown in Fig. 9. The roles of OWNER and REPAIRER would be adopted by the policy holders and the mechanics, respectively. The OWNER requesting repairs counts as her committing to the mechanic to acknowledge any repair services provided (**REP$_1$**). The REPAIRER completing the repairs counts as him providing the repair service (**REP$_2$**). The OWNER responding either positively or negatively counts as her affirming the provision of repair services (**REP$_3$** and **REP$_4$**). The value of claimNO flows from repairReq to repaired and subsequently to repairOK or repairNOK (**REP$_6$** and **REP$_7$**). Messages repairOK and repairNOK are mutually exclusive (**REP$_8$**).

**REP$_1$.** repairReq(claimNO) → CC(O, Rp, repairServed(claimNO), affirm(claimNO))

**REP$_2$.** repaired(claimNO) → repairServed(claimNO) if repairReq(claimNO)

**REP$_3$.** repairOK(claimNO, approval) → affirm(claimNO) if repaired(claimNO)

**REP$_4$.** repairNOK(claimNO) → affirm(claimNO) if repaired(claimNO, policyNO)

**REP$_5$.** repairReq ⇝ repaired

**REP$_6$.** repaired.claimNO ⇝ repairOK.claimNO

**REP$_7$.** repaired ⇝ repairNOK

**REP$_8$.** repairOK XOR repairNOK

### A.3 Handling Filed Claims (*Han*)

A scenario of *Han* is shown in Fig. 9. The roles of HANDLER, GARAGE, and ASSESSOR would be adopted by Lee CS, the mechanics, and the inspectors, respectively. The assumed commitment to respond to inspection requests would have been created when the inspector and Lee CS formed their relationship (**HAN$_1$**).

The GARAGE estimating a price for repairs counts as a commitment to perform repairs if the HANDLER accepts the estimate (**HAN$_2$**). The HANDLER may ask the ASSESSOR to assess the cost of repairs and the value of the car, which counts as a request for inspection (**HAN$_3$**). The ASSESSOR responding with the necessary assessments counts as an inspection response (**HAN$_4$**). The HANDLER may accept to deal with the mechanic on a previous estimate. Doing so creates a commitment for the HANDLER to pay for the completed repairs (**HAN$_5$**). Also, if the price of the deal matches that of the estimate, then a deal counts as acceptance of the estimate (**HAN$_6$**). The mechanic performs repairs, obtains approval from the policy holder, and sends a bill to the consultant. Billing on a previously accepted estimate counts as completing the repairs (**HAN$_7$**). The HANDLER's paying the bill counts as payment for the repairs (**HAN$_8$**).

The value of claimNO flows from estimate to deal or noDeal, and to inspect ($\textbf{HAN}_9$, $\textbf{HAN}_{10}$, and $\textbf{HAN}_{11}$). Because deal does not depend on inspect, requesting inspections is not always necessary. The price in the bill must flow from the price specified in the deal ($\textbf{HAN}_{14}$). Similarly, the paid price must be the same as the billed price ($\textbf{HAN}_{16}$). Other data flows are self explanatory. Finally, deal and noDeal are mutually exclusive ($\textbf{HAN}_{17}$).

$\textbf{HAN}_1$. start → CC(A, H, inspectReq(claimNO), inspectRes(claimNO))

$\textbf{HAN}_2$. estimate(claimNO, price) →
CC(G, H, acceptEstimate(claimNO, price), performRepair(claimNO))

$\textbf{HAN}_3$. inspect(claimNO) → inspectReq(claimNO)

$\textbf{HAN}_4$. inspected(claimNO, cost, carValue) → inspectRes(claimNO) if inspect(claimNO)

$\textbf{HAN}_5$. deal(claimNO, price) → CC(H, G, performRepair(claimNO), payment(price))

$\textbf{HAN}_6$. deal(claimNO, price) → acceptEstimate(claimNO, price) if estimate(claimNO, price)

$\textbf{HAN}_7$. bill(claimNO, price, approval) → performRepair(claimNO) if deal(claimNO, price)

$\textbf{HAN}_8$. pay(claimNO, price) → payment(price) if bill(claimNO, price, approval)

$\textbf{HAN}_9$. estimate ⇝ deal

$\textbf{HAN}_{10}$. estimate ⇝ noDeal

$\textbf{HAN}_{11}$. estimate.claimNO ⇝ inspect.claimNO

$\textbf{HAN}_{12}$. inspect.claimNO ⇝ inspected.claimNO

$\textbf{HAN}_{13}$. deal.claimNO ⇝ bill.claimNO

$\textbf{HAN}_{14}$. deal.price ⇝ bill.price

$\textbf{HAN}_{15}$. bill.claimNO ⇝ pay.claimNO

$\textbf{HAN}_{16}$. bill.price ⇝ pay.price

$\textbf{HAN}_{17}$. deal XOR noDeal

### A.4 Monitoring (*Mon*)

A scenario of *Mon* is shown in Fig. 9. The roles of COMPANY and consultant would be adopted by AGFIL and Lee CS, respectively. The COMPANY assigns a case to the CONSULTANT who after taking necessary steps, returns with an invoice. The COMPANY authorizes payment for the consulting services provides by the CONSULTANT.

The assumed commitment represents an agreement between the COMPANY and the CONSULTANT reached *a priori* ($\textbf{MON}_1$).

The CONSULTANT returning back with an invoice of a handled claim counts as the CONSULTANT providing the consulting service ($\textbf{MON}_2$). The COMPANY authorizing a payment to the CONSULTANT counts as a payment for the consulting service ($\textbf{MON}_3$). The values of claimNO and policyNO flows from handle to invoice, and from invoice into authorizePay ($\textbf{MON}_4$, $\textbf{MON}_5$, $\textbf{MON}_6$, and $\textbf{MON}_7$). Also, the value of quote flows from invoice into authorizePay ($\textbf{MON}_8$).

$\textbf{MON}_1$. start → CC(Com, Con, consultingService(claimNO), payForService(claimNO))

$\textbf{MON}_2$. invoice(claimNO, quote, approval) → consultingService(claimNO) if handle(claimNO)

$\textbf{MON}_3$. authorizePay(claimNO, quote) → payForService(claimNO)
if invoice(claimNO, quote, approval)

$\textbf{MON}_4$. handle.claimNO ⇝ invoice.claimNO

$\textbf{MON}_5$. handle.policyNO ⇝ invoice.policyNO

$\textbf{MON}_6$. invoice.claimNO ⇝ authorizePay.claimNO

**MON$_7$.** invoice.policyNO $\rightsquigarrow$ authorizePay.policyNO

**MON$_8$.** invoice.quote $\rightsquigarrow$ authorizePay.quote

### A.5 Partial insurance claim processing (*Picp*)

*Picp* is composed of *Bas*, *Mon*, *Han*, and *Rep*. The following role identifications are self explanatory.

**PICP$_1$.** Picp.Insured $\doteq$ Bas.Insured, Rep.Owner

**PICP$_2$.** Picp.Insurer $\doteq$ Bas.Insurer, Mon.Company

**PICP$_3$.** Picp.CallCenter $\doteq$ Bas.CallCenter

**PICP$_4$.** Picp.Consultant $\doteq$ Han.Handler, Mon.Consultant

**PICP$_5$.** Picp.Repairer $\doteq$ Rep.Repairer, Han.Garage

**PICP$_6$.** Picp.Assessor $\doteq$ Han.Assessor

The INSURED requesting repairs from a REPAIRER counts as a request for repair service (**PICP$_7$**). INSURED approving the repairs counts as provision of claim service (**PICP$_8$**). The value of claimNO flows from approved into repairReq and from notification into handle (**PICP$_{10}$** and **PICP$_9$**). Similarly, the claimNO in estimate gets its value from the claimNO in repairReq (**PICP$_{11}$**).

**PICP$_7$.** Rep.repairReq(claimNO) $\rightarrow$ Bas.serviceReq(claimNO)

**PICP$_8$.** Rep.repairOK(claimNO, approval) $\rightarrow$ Bas.claimService(claimNO) if Rep.repaired(claimNO)

**PICP$_9$.** Bas.notification.claimNO $\rightsquigarrow$ Mon.handle.claimNO

**PICP$_{10}$.** Bas.approved.claimNO $\rightsquigarrow$ Rep.repairReq.claimNO

**PICP$_{11}$.** Rep.repairReq.claimNO $\rightsquigarrow$ Han.estimate.claimNO

Before the REPAIRER performs repairs, the repair estimate must have been accepted by the CONSULTANT (**PICP$_{12}$**). Similarly, before the REPAIRER sends a bill to the CONSULTANT, the approval of repairs from the INSURED must have been acquired (**PICP$_{13}$**). Finally, the CONSULTANT must have received a bill from the REPAIRER before he sends an invoice to the INSURER (**PICP$_{14}$**).

**PICP$_{12}$.** Han.deal $\prec$ Rep.repaired

**PICP$_{13}$.** Rep.repairOK $\prec$ Han.bill

**PICP$_{14}$.** Han.bill $\prec$ Mon.invoice

### A.6 Pay cash and scrap car (*Pcsc*)

A scenario of *Pcsc* is shown in Fig. 10. The roles of OWNER, COMPANY, and CONSULTANT would be adopted by the policy holders, AGFIL, and Lee CS, respectively.

The COMPANY offering cash in lieu of repairs creates a commitment to pay the offered amounts as settlement if the offer is accepted (**PCSC$_1$**). Similarly, the COMPANY notifying the OWNER of scrapping of the car and promising settlement money creates a commitment to pay the value of the car as settlement (**PCSC$_2$**). The OWNER accepting the offer counts as an acceptance of the cash offer (**PCSC$_3$**). The COMPANY paying the settlement amount counts as achievement of a settlement. The data flows are self-explanatory. Finally, the CONSULTANT can either advise to scrap the car or advise to pay cash in lieu of repairs, but not both (**PCSC$_{11}$**). Similarly, the cash offer can either be accepted or rejected, but not both (**PCSC$_{12}$**).

**PCSC$_1$.** cashOffer(claimNO, amount) $\rightarrow$ CC(Com, O, acceptCash(claimNO), settlement(amount))
**PCSC$_2$.** adviseScrap(claimNO, value) $\rightarrow$ C(Com, O, settlement(value))
**PCSC$_3$.** accept(claimNO, amount) $\rightarrow$ acceptCash(claimNO) if cashOffer(claimNO, amount)
**PCSC$_4$.** settle(claimNO, amount) $\rightarrow$ settlement(amount) if accept(claimNO, amount)
**PCSC$_5$.** settle(claimNO, value) $\rightarrow$ settlement(value) if adviseScrap(claimNO, value)
**PCSC$_6$.** adviseScrap $\rightsquigarrow$ settle
**PCSC$_7$.** adviseCash $\rightsquigarrow$ cashOffer
**PCSC$_8$.** cashOffer $\rightsquigarrow$ accept
**PCSC$_9$.** accept $\rightsquigarrow$ settle
**PCSC$_{10}$.** cashOffer $\rightsquigarrow$ reject
**PCSC$_{11}$.** adviseScrap XOR adviseCash
**PCSC$_{12}$.** accept XOR reject

### A.7 Outsourced insurance claim processing (*Out*)

A scenario of *Out* is shown in Fig. 12. The roles of INSURED, INSURER, CALL CENTER, and CONSULTANT would be adopted by the policy holders, AGFIL, Europ Assist, and Lee CS respectively. In *Out*, the details of how the CONSULTANT handles claims are hidden as business logic.

The role identification axioms are self explanatory. The CONSULTANT returning back with an invoice of a handled claim counts as provision of the claim service to the INSURED (**OUT$_5$**). The invoice is signed with an approval by the INSURED. The INSURED asking the CONSULTANT to handle a claim counts as a request for repair services (**OUT$_6$**). The condition serviceReq affects the commitments created via *Ins*. The data flows are self explanatory.

**OUT$_1$.** Out.Insured $\doteq$ Bas.Insured
**OUT$_2$.** Out.Insurer $\doteq$ Bas.Insurer, Mon.Company
**OUT$_3$.** Out.CallCenter $\doteq$ Bas.CallCenter
**OUT$_4$.** Out.Consultant $\doteq$ Mon.Consultant
**OUT$_5$.** Mon.invoice(claimNO, quote, approval) $\rightarrow$ Bas.claimService(claimNO) if Mon.handle(claimNO)
**OUT$_6$.** Mon.handle(claimNO) $\rightarrow$ Bas.serviceReq(claimNO)
**OUT$_7$.** Bas.approved.claimNO $\rightsquigarrow$ Mon.handle.claimNO
**OUT$_8$.** Bas.approved.policyNO $\rightsquigarrow$ Mon.handle.policyNO

### A.8 Fraudulent Claims Detection (*Fra*)

A scenario of *Fra* is shown in Fig. 13. The roles of OWNER, COMPANY, CONSULTANT, and ASSESSOR would be adopted by the policy holders, AGFIL, Lee CS, and the inspectors, respectively. Due to the absence of indigenous commitments, no message axioms are needed. The value of claimNO flows from adviseFraud into fraudulent, and from fraudulent into fraud.

**FRA$_1$.** adviseFraud.claimNO $\rightsquigarrow$ fraudulent.claimNO
**FRA$_2$.** fraudulent.claimNO $\rightsquigarrow$ fraud.claimNO

Received December 2007; revised May 2008, November 2008