Modeling and Reasoning about Service-Oriented Applications via Goals and Commitments

Amit K. Chopra, Fabiano Dalpiaz, Paolo Giorgini, and John Mylopoulos

Department of Information Engineering and Computer Science, University of Trento
chopra,dalpiaz,paolo.giorgini,jm@disi.unitn.it

Abstract. Service-oriented applications facilitate the exchange of business services among participants. Existing modeling approaches either apply at a lower of abstraction than required for such applications or fail to accommodate the autonomous and heterogeneous nature of the participants. We present a business-level conceptual model that addresses the above shortcomings. The model gives primacy to the participants in a service-oriented application. A key feature of the model is that it cleanly decouples the specification of an application's architecture from the specification of individual participants. We formalize the connection between the two—the reasoning that would help a participant decide if a specific application is suitable for his needs. We implement the reasoning in datalog and apply it to a case study involving car insurance.

Keywords: Business modeling, Commitments, Goals, Service engagements, Service-oriented architecture

1 Introduction

Service-oriented applications exemplify programming-in-the-large [8]: the architecture of the application takes precedence over the specification of services. An individual service may be designed using any methodology in any programming language as long as it structurally fits in with the rest of the system. Component-based systems embody this philosophy, but service-oriented applications are fundamentally different in that they represent *open systems* [16, 21]. A service-oriented application is characterized by the autonomy and heterogeneity of the participants. Application participants engage each other in a service enactment via interaction. Applications are dynamic implying that participants may join or leave as they please. The identity of the participants need not even be known when designing the application. In a sense, open systems take the idea of programming-in-the-large to its logical extreme.

An example of a service-oriented application are auctions on eBay. Multiple autonomous and heterogeneous participants are involved: eBay itself, buyers, sellers, payment processors, credit card companies, shippers, and so on. eBay (the organization) specified the architecture of the application in terms of the roles (seller, bidder, shipper, and so on) and the interaction among them without knowing the identity of the specific participants that would adopt those roles.

For service-oriented applications, it is especially useful to treat the architecture as being largely synonymous with the application itself. The auctions application on eBay exists whether some auction is going on or not. The application is *instantiated* when participants adopt (play) roles in the application. The application is *enacted* when participants interact according to the roles (see Figure 1). Moreover, the application is, in general, specified independently from the specification of the individual participants. Clearly, the notion of roles, participants, and interaction are key elements in the modeling of service-oriented applications.

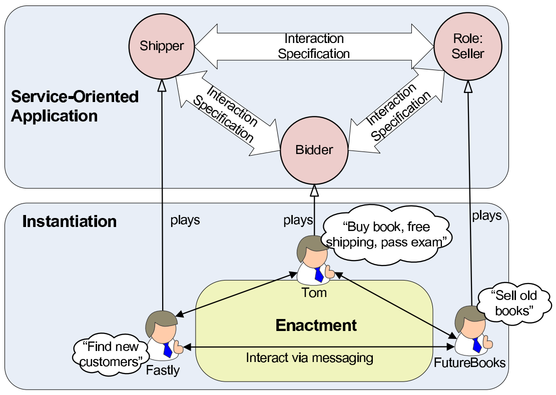

Fig. 1. A service-oriented application is specified in terms of roles. It is instantiated when participants adopt those roles; it is enacted when participants interact according to the adopted roles.

The real value of service-oriented computing is realized for applications in which participants engage each other in business transactions, for example, in an auction on eBay. Each individual participant has his own business goals, and he would need to interact flexibly with others so as to be able to fulfill his goals. Ideally, we would want to model both applications and participants in terms of business-level abstractions. We would also want to characterize and reason about properties critical to doing business, such as *goal fulfillment*, *compliance*, *interoperability*, and so on, in similarly high-level terms. This is key to alleviating the business-IT gap.

Existing conceptual modeling approaches either (i) lack the notion of roles, participants, and interactions altogether, or (ii) are lacking in business-level abstractions—they are typically rooted in control and data flow. Workflow-based modeling of applications, as is done using BPMN (the *Business Process Modeling Notation*), exemplifies the former; choreography-based modeling, as is done using WS-CDL (the *Web Services Choreography Description Language*), exemplifies the latter. Many approaches fall somewhere in between (discussed extensively in Section 5).

We propose a conceptual model for service-oriented applications that addresses both the above concerns. It gives primacy to the autonomy and heterogeneity of participants, and works at the business-level. The key insight behind the model is this. A participant will have business *goals* that he wants to achieve. However, given his autonomy, a participant cannot force another to bring about any goal. In fact, a participant wouldn't even know the internal construction—in the forms of business rationale, rules, goals, strategies, procedures, or however otherwise specified—of any other participant. In such situations, the best a participant can do is to deal in *commitments* (concerning the goals he wants to achieve) with other participants. For example, a bidder on eBay cannot force a seller to deliver even if he has won the auction; the interaction between them proceeds on the understanding that there is a commitment from the seller to deliver if the bidder has won.

This paper synthesizes results from two influential lines of research: goal-oriented requirements engineering [24] and agent communication [19]. Specifically, we specify participants in terms of their goal models, and application architecture in terms of commitments. The conceptual model enables reasoning about properties at a business-level. In this paper, we focus on axiomatizing the *supports* relation, which essentially formalizes the notion of whether adopting a role in a particular application is compatible with a participant's goals. We implement a prototype reasoning tool that encodes the supports relation in datalog. We evaluate the usefulness of our conceptual model by modeling a car insurance scenario, and describe how we may encode and reason about the model. We also report on experiments that show the scalability of the reasoning.

The rest of the paper is organized as follows. Section 2 describes our conceptual model in detail and shows the relation between individual participants and the application. Section 3 shows how we reason about compatibility between the commitments a participant might be party to and his goals. Section 4 evaluates our approach. We model elements of a car insurance application, and show some queries one may run. The section also reports on the scalability results. Section 5 summarizes our contribution, discusses the relevant literature, and highlights future directions.

2 Conceptual Model

From here on, we refer to the participants in an application as *agents*. This term is appropriate given their autonomous and heterogeneous nature. Figure 2 shows the proposed conceptual model: the left box concerns a service-oriented application, the right box is about an agent's requirements.

2.1 Specifying Agents via Goal Models

An agent is specified in terms of a goal model, as formalized in the Tropos methodology [2]. Goal modeling captures important aspects of requirements—not just what they are, but why they are there. An agent's goal model represents his motivations, and abstracts away from low-level details of control and data flow. We now briefly revisit the aspects of goal modeling relevant to this paper.

As shown in Figure 2, an Agent *has* some goals. A Goal may be a HardGoal or a SoftGoal. A softgoal has no clear-cut criteria for satisfaction (its satisfaction is subjectively evaluated). A goal *reflects* a state of the world desired by the agent. A goal may contribute to other goals: $++S(g, g')$ means g contributes positively to the achievement of g'; $--S(g, g')$ means g contributes negatively to the achievement of g'. Both hard and soft goals may be AND-decomposed or OR-decomposed into subgoals of the same type. Additionally, an agent may be capable of a number of hard-goals; the notion of capability abstracts the means-end relation in Tropos.

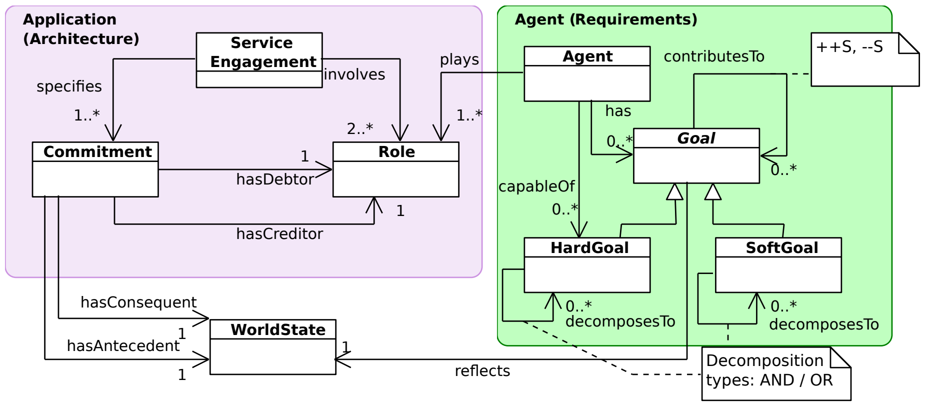

Fig. 2. Conceptual model for service-oriented applications and participating agents

2.2 Specifying Applications via Service Engagements

Conceptually, the *commitment* $\mathsf{C}(\mathsf{Debtor}, \mathsf{Creditor}, \mathsf{antecedent}, \mathsf{consequent})$ means that the debtor is committed to the creditor for the $\mathsf{consequent}$ if the $\mathsf{antecedent}$ holds. The *antecedent* and *consequent* are propositions that refer to the *states of the world* of relevance to the application under consideration. A commitment is *discharged* when its consequent is achieved; it is *detached* when the antecedent holds. An unconditional commitment is one where the antecedent is $\top$ (true).

For example, in an auction application, one can imagine a commitment $\mathsf{C}(\mathsf{Bidder}, \mathsf{Seller}, \mathsf{wonBid}, \mathsf{paymentMade})$. Informally, it means that the bidder commits to the seller that if the world state is such that he has won the bid, then he will bring about the world state where the payment has been made.

We use commitments as the basis of architectural connections. As Figure 2 shows, a Service Engagement *involves* two or more roles and *specifies* one or more commitments among the involved roles. A Role role can be debtor (creditor) in one or more commitments; each commitment has exactly one debtor (creditor). A commitment has an antecedent and a consequent, each representing some state of the world. Table 1 introduces the message types by which agents update commitments [5]. In the table, $x, y, \ldots$ are variables over agents, and $p, q, \ldots$ are variables over propositions.

Conceptually, a *service engagement* is a business-level specification of interaction. It describes the *possible* commitments that may arise between agents adopting the roles, and via the standard messages of Table 1, how the commitments are updated. An engagement should not be interpreted to mean that by simply adopting roles in this engagement the agents will become committed as stated. The commitments themselves would come about at runtime via exchange of messages. Moreover, whether an agent sends a particular message is solely his own decision.

Commitments are made in a certain sociolegal context and represent a contractual relationship between the debtor and the creditor. They yield a notion of compliance expressly suited for service-oriented applications. Agent compliance amounts to the agent not violating any of his commitments towards others. A service engagement specified

Message	**Sender**	**Receiver**	**Effect**	**Business Significance**
$\mathsf{Create}(x, y, r, u)$	x	y	$\mathsf{C}(x, y, r, u)$	brings about a relation
$\mathsf{Cancel}(x, y, r, u)$	x	y	$\neg\mathsf{C}(x, y, r, u)$	dissolves relation
$\mathsf{Release}(x, y, r, u)$	y	x	$\neg\mathsf{C}(x, y, r, u)$	dissolves relation
$\mathsf{Delegate}(x, y, z, r, u)$	x	z	$\mathsf{C}(z, y, r, u)$	delegates relation to another debtor
$\mathsf{Assign}(x, y, z, r, u)$	y	x	$\mathsf{C}(x, z, r, u)$	assigns relation to another creditor
$\mathsf{Declare}(x, y, p)$	x	y	p	informs about some aspect of state

Table 1. Messages and their effects; a commitment is understood as a contractual relation

in terms of commitments does not dictate specific operationalizations (runtime enactments) in terms of when an agent should send or expect to receive particular messages; as long as the agent discharges his commitments, he can act as he pleases [9].

Figure 3 shows the (partial) service engagement for an auction application. Figure 4 shows a possible enactment for the service engagement of Figure 3. The bidder first creates $\mathsf{c_B}$. Then he places bids, possibly increasing his bids (indicated by the dotted bidirectional $\mathsf{Bidding}$ arrow). The seller informs the bidder that he has won the bid, which detaches $\mathsf{c_B}$ and causes the unconditional commitment $\mathsf{c_{UB}} = \mathsf{C}(\mathsf{Bidder}, \mathsf{Seller}, \top, \mathsf{paymentMade})$ to hold. Finally, the bidder discharges his commitment by sending the payment.

$$\boxed{\begin{array}{l} \mathsf{c_B} = \mathsf{C}(\mathsf{Bidder}, \mathsf{Seller}, \mathsf{wonBid}, \mathsf{paymentMade}) \\ \mathsf{c_S} = \mathsf{C}(\mathsf{Seller}, \mathsf{Bidder}, \mathsf{paymentMade}, \mathsf{itemDelivered}) \\ \mathsf{c_A} = \mathsf{C}(\mathsf{Auctioneer}, \mathsf{Bidder}, \top, \mathsf{itemCheckedForAuthenticity}) \end{array}}$$

Fig. 3. A (partial) service engagement depicting an auction application. The labels are for reference purposes only. Figure 4 shows an enactment of this engagement between a bidder agent and a seller agent

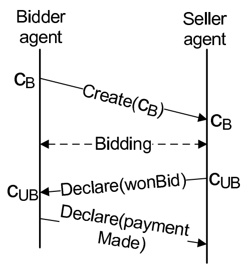

Fig. 4. An enactment

Where do service engagements come from? Domain experts specify these from scratch or by reusing existing specifications that may be available in a repository. In eBay's case, presumably architects, experts on the various kinds of businesses (such as payment processing, shipping, and so on) and processes (auctions) involved, and some initial set of stakeholders got together to define the architecture. How *application requirements* (as distinct from an individual participant's requirements) relate to the specification of service engagements is studied in [9].

2.3 Binding

As Figure 2 shows, an agent may choose to *play*, in other words, adopt one or more roles in a service engagement. Such an agent is termed an *engagement-bound agent*. Adopting a role is the key notion in instantiating an application, as shown in Figure 1.

However, before a bound agent may start interacting, he may want to verify that he is compatible with the engagement. The semantic relationship between a service engagement and an agent's goals is the following. To fulfill his goals, an agent would select a role in some service engagement and check whether adopting that role is compatible with the fulfillment of his goals. If it is compatible, then the agent would presumably act according to the role to fulfill his goals; else, he would look for another service engagement. For example, the bidder may have the requirement of a complete refund from the seller if the seller delivers damaged goods. The bidder must check whether the service engagement with the seller includes a commitment from the seller to that effect; if not, he may try a different service engagement. We formalize compatibility via the notion of *supports* in Section 3.

Notice in Figure 2 that both commitments and goals are expressed in terms of world states. This provides the common ontological basis for reasoning between goal and commitments.

3 Goal and Commitment Support

The conceptual model supports two kinds of compatibility reasoning. Given some role in a service engagement and some goal that the agent wants to achieve, *goal support* verifies whether an agent can *potentially* achieve his goal by playing that role. *Commitment support* checks if an agent playing a role is *potentially* able to honor the commitments he may make as part of playing the role.

Note the usage of the words *support* and *potentially*. Goal (commitment) support is a weaker notion than goal fulfillment; support gives no guarantee about fulfillment at runtime. And yet, it is a more pragmatic notion for open systems, where it is not possible to make such guarantees anyway. For instance, a commitment that an agent depends upon to fulfill his goal may be violated.

Goal support Agent x (at runtime, or his designer) may execute a query to check whether playing a role ρ in the service engagement under consideration supports x's goal g. Intuitively, a goal g is supported if (i) no other goal g' that x intends to achieve negatively contributes to g; and (ii) either of the following holds:

1. x is capable of g, or
2. x can get from some other agent playing role ρ' the commitment $\mathsf{C}(\rho', \rho, g', g)$ and x supports g' (akin to x requesting an offer from ρ'), or
3. x can make $\mathsf{C}(\rho, \rho', g, g')$ to some agent playing ρ'; in other words, x commits to g' if the other agent agent achieves g (akin to x making an offer to ρ'), or
4. g is and-decomposed (or-decomposed) and x supports all (at least one) subgoals, or
5. some other supported goal that x intends to achieve positively contributes to g.

Notice the difference between clauses 2 and 3. In clause 2, x can get a commitment to support a goal g only if x supports the antecedent; in other words, x cannot realistically hope that some agent will play ρ' and will benevolently bring about g. According to clause 3, x can support g without supporting the consequent g' of the commitment. Support of the consequent by the debtor (here x) is a matter of checking for commitment support, as explained below.

An important aspect in our reasoning is that of visibility (or scope). Visibility roughly amounts to the goals that an agent intends to achieve. The content of a goal query defines the reasoning scope, namely which are the goals that the agent intends to achieve.

Given a query for a goal g, the query scope consists of all the subgoals of the tree starting from the top-level goal ancestor of g. Visibility is important in order to rule out contributions from goals which are not intended.

Goal support is presented with respect to a single goal for the sake of exposition, but this notion is easily generalized to propositions. For instance, one might query for the support of a goal proposition $g_1 \wedge g_2$. In this case, the query scope is the union of the scopes of g_1 and g_2. Similarly, the antecedent and consequent of a commitment can be expressed using propositions.

-gs(X) :- goal(X), not v(X).	R1. Goals out of the scope cannot be supported.
do(X) v -do(X) :- cap(X).	R2. Capabilities can be exploited or not.
gs(X) :- do(X).	R3. Using a capability implies goal support.
gs(X) :- v(X), gs(Y), pps(Y,X). *-gs(X) :- v(X), gs(Y), mms(Y,X).*	R4. ++S and --S apply from and to visible goals.
v(X) :- v(Y), anddec(Y,L), goal(X), #member(X,L). *v(X) :- v(Y), ordec(Y,L), goal(X), #member(X,L).*	R5. A subgoal is visible if its parent is visible.
gs(X) :- ordec(X,L), #member(Y,L), gs(Y).	R6. An or-decomposed goal is supported if any of its subgoals is supported.
gs(X) :- anddec(X,Y), suppAllS(X,Y). *wand(X,Y) :- anddec(X,Y).* *wand(X,Y) :- wand(X,[A|Y]), goal(A).* *suppAllS(A,X) :- wand(A,X), #length(X,N), N=0.* *suppAllS(A,X) :- wand(A,X), #length(X,N), N>0, #memberNth(X,1,E), gs(E), #tail(X,X1), suppAllS(A,X1).*	R7. An and-decomposed goal is supported if all of its subgoals are supported. These rules split the subgoals list into atomic goals.
comm(X,Y,C) :- cc(A,B,X,Y,C), plays(B). *comm([],X,C) :- cc(A,B,X,Y,C), plays(A).*	R8. The agent can exploit only those commitments where it plays debtor or creditor.
e(X) v -e(X) :- comm(_,_,X). *-e(C) :- comm(L,Y,C), not suppAll(C,L).*	R9. Commitments can be exploited only if the precondition is supported.
gs(Y) :- comm(X,L,C), #member(Y,L), goal(Y), suppAll(C,X), e(C).	R10. A goal is supported by a commitment if it is in the consequent, the antecedent is supported, and the commitment is exploited.
suppAll(C,X) :- wcomm(X,_,C), #length(X,0). *suppAll(C,X) :- wcomm(X,_,C), #length(X,N), N>0, #memberNth(X,1,E), gs(E), #tail(X,X1), suppAll(C,X1).* *wcomm(L1,L2,C) :- comm(L1,L2,C).* *wcomm(L1,L2,C) :- wcomm([A|L1],L2,C), goal(A).*	R11. A commitment's antecedent is supported if all the goals in the antecedent are supported. These rules split the antecedent into atomic components.

Table 2. Datalog (DLV-complex) axiomatization of the supports relation

Table 2 axiomatizes the above rules as a logic program in datalog[1] (any sufficiently expressive logic programming language would have sufficed for our present purposes). A goal is expressed as an atomic proposition. Antecedent and consequent of commitments are expressed as lists of atomic propositions (the list is interpreted as a conjunction). Given (i) an agent's goal model; (ii) a service engagement; and (iii) the role played by the agent in the engagement, the predicate $gs(g_0 \wedge \ldots \wedge g_n)$ is true if and only if each of $g_0,\ldots,g_n$ is supported.

[1] www.mat.unical.it/dlv-complex

A goal model is defined by the following predicates: $goal(g)$ states that g is a goal; $anddec(g, [g_1, \ldots, g_n])$ ($ordec(g, [g_1, \ldots, g_n])$) denotes that g is and-decomposed (or-decomposed) to $g_1, \ldots, g_n$; $pps(g_1, g_2)$ ($mms(g_1, g_2)$) represents a ++S (- -S) contribution from g_1 to g_2; $cap(g)$ says that the agent is capable of goal g. The predicate $cc(r_1, r_2, [g_1, \ldots, g_l], [g_m, \ldots, g_n])$ indicates the commitment $\mathsf{C}(r_1, r_2, g_1 \wedge \ldots \wedge g_l, g_m \wedge \ldots \wedge g_n)$. The predicate $plays(r)$ states that the considered agent plays role r.

The query scope (the visibility predicate v) is manually defined for what concerns the top-level goals; then it is propagated top-down by rule R5. This may be automated by macros.

Commitment support. It makes sense to check whether an agent will be able to support the commitments it undertakes as part of a service engagement. In other words, let's say to support g, x makes $\mathsf{C}(x, y, g, g')$ to an agent y. Now if y brings about g, x will be unconditionally committed to bringing about g'. If x is not able to support g', then x will potentially be in violation of the commitment. Commitment support reduces to goal support for the commitment consequent.

A reckless or malicious agent may only care that his goals are supported regardless of whether his commitments are supported; a prudent agent on the other hand would ensure that the commitments are also supported.

Reasoning for support as described above offers interesting possibilities. Some examples: (i) x can reason that $\mathsf{C}(x, y, g_0, g_1)$ is supported by $\mathsf{C}(z, x, g_2, g_1)$ if x supports g_2; (ii) x can support a conjunctive goal $g_0 \wedge g_1$ by getting commitments for g_0 and g_1 from two different agents, (iii) to support g in a redundant manner, x may get commitments for g from two different agents; and so on.

4 Evaluation

First, we model a real-life scenario using our conceptual model, and show how we may reason about it. Second, we demonstrate the scalability of the *supports* reasoning.

4.1 Case study: insurance claim processing

We show how the model and the reasoning techniques can be used to model a real life setting concerning car insurance claim processing. We base our scenario on the documentation that the Financial Services Commission of Ontario (FSCO) provides online, specifically on the description of the claim process[2]. The process describes the perspective of a driver involved in a car accident in Ontario; it also highlights what happens behind the scenes. It describes a service engagement that is independent of specific insurance companies, car repairers, and damage assessors. We assume the car driver is not at fault and his policy has no deductible.

Figure 5 describes the service engagement in the car insurance claim processing scenario. The engagement is defined as a set of roles (circles) connected via commitments; the commitments are labeled c_i. Table 3 explains the commitments.

Figure 6 shows an agent model where agent Tony plays role repairer. The main goal of Tony is to perform a repair service. This is and-decomposed to subgoals car repaired,

[2] http://www.fsco.gov.on.ca/english/insurance/auto/after_auto_accident_ENG.pdf

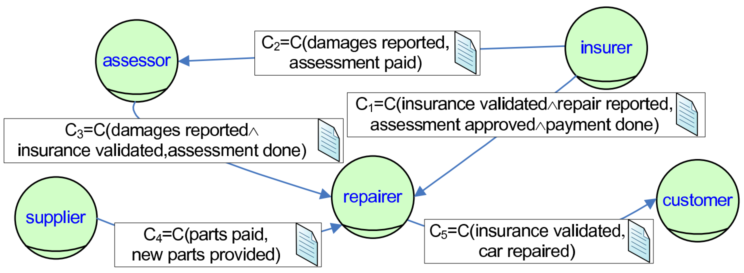
assessor
C2=C(damages reported, assessment paid)
insurer
C1=C(insurance validated∧repair reported, assessment approved∧payment done)
C3=C(damages reported∧ insurance validated,assessment done)
repairer
supplier
C4=C(parts paid, new parts provided)
customer
C5=C(insurance validated, car repaired)

Fig. 5. Role model for the insurance claim processing scenario. Commitments are rectangles that connect (via directed arrows) debtors to creditors

c_1	insurer to repairer: if insurance has been validated and the repair has been reported, then the insurer will have paid and approved the assessment
c_2	insurer to assessor: if damages have been reported, the assessment will have been paid
c_3	assessor to repairer: if damages have been reported and the insurance has been validated, a damage assessment will have been performed
c_4	supplier to repairer: if parts have been paid for, new parts will have been provided
c_5	repairer to customer: if the insurance has been validated, then the car will have been repaired

Table 3. Commitments in the car insurance service engagement

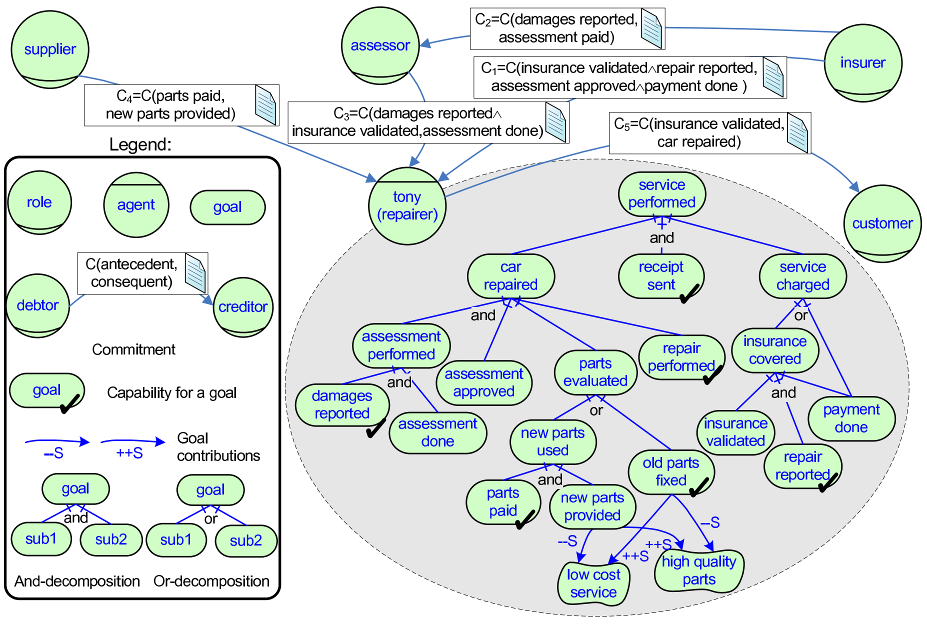

Fig. 6. Visual representation of Tony's insurance-engagement bound specification. Tony plays repairer.

receipt sent, and service charged. The goal model contains two variation points: the or-decompositions of goals parts evaluated and service charged. The former goal is or-decomposed to subgoals new parts provided and old parts fixed. Note the softgoals low cost service and high quality parts. Using new parts has a negative contribution to low cost service and a positive one to high quality parts, whereas fixing old parts contributes oppositely to those soft goals.

Table 4 show the insurance engagement-bound specification of Tony in datalog. Even though both the service engagement and Tony's requirements are in a single table, we remark that they are independently defined artifacts. The binding of Tony to the repairer role in the engagement in indicated by *plays(repairer)* at the beginning of the specification. We now demonstrate the reasoning.

```
%   AGENT-ROLE plays relation
plays(repairer).

%   GOALS: each goal node is declared, only three shown below
goal(servicePerformed). goal(carRepaired). goal(receiptSent).

%   CAPABILITIES
cap(receiptSent). cap(damagesReported). cap(partsPaid).
cap(oldPartsFixed). cap(repairPerformed). cap(repairReported).

%   GOAL MODEL: DECOMPOSITIONS
anddec(servicePerformed,[carRepaired,receiptSent,serviceCharged]).
anddec(carRepaired,[assessmentPerformed,assessmentApproved,
            partsEvaluated,repairPerformed]).
anddec(assessmentPerformed,[damagesReported,assessmentDone]).
ordec(partsEvaluated,[newPartsUsed,oldPartsFixed]).
anddec(newPartsUsed,[partsPaid,newPartsProvided]).
ordec(serviceCharged,[insuranceCovered,paymentDone]).
anddec(insuranceCovered,[insuranceValidated,repairReported,paymentDone]).

%   GOAL MODEL: CONTRIBUTIONS
mms(newPartsProvided,lowCostService). pps(newPartsProvided,highQualityParts).
pps(oldPartsFixed,lowCostService). mms(oldPartsFixed,highQualityParts).

%   COMMITMENTS IN THE SERVICE ENGAGEMENT
cc(insurer,repairer,[insuranceValidated,repairReported],
            [assessmentApproved, paymentDone],c1).
cc(insurer,assesser,[damagesReported],[assessmentPaid],c2).
cc(assesser,repairer,[damagesReported,insuranceValidated],[assessmentDone],c3).
cc(supplier,repairer,[partsPaid],[newPartsProvided],c4).
cc(repairer,customer,[insuranceValidated],[carRepaired],c5).
```

Table 4. Datalog representation of Figure 6

Table 5 shows some queries for support of particular goals and their solutions. The solutions represent the output that our implementation provides; each solution is a possible strategy to support a goal. A strategy consists of a set of exploited capabilities and a set of commitments that the agent can get or make to other agents. Below, we describe the posed queries and we provide some details to explain why the alternatives are valid solutions to the query. The queries pertain to the insurance engagement-bound Tony.

Query 1 Can Tony support service performed?

This query has four solutions (see Table 5). Solution 1 includes both options to fix cars (to support goal parts evaluated): new parts are bought and old parts are fixed.

Query 1: Can Tony support "service performed"?
v(servicePerformed).
gs(servicePerformed)?
Solutions:
1: {do(receiptSent), do(repairPerformed), do(damagesReported), do(oldPartsFixed), do(partsPaid), do(repairReported), e(c1), e(c3), e(c4), e(c5)}
2: {do(receiptSent), do(repairPerformed), do(damagesReported), do(partsPaid), do(repairReported), e(c1), e(c3), e(c4), e(c5)}
3: {do(receiptSent), do(repairPerformed), do(damagesReported), do(oldPartsFixed), do(partsPaid), do(repairReported), e(c1), e(c3), e(c5)}
4: {do(receiptSent), do(repairPerformed), do(damagesReported), do(oldPartsFixed), do(repairReported), e(c1), e(c3), e(c5)}

Query 2: Can Tony support "service performed" and "high quality"?
v(servicePerformed). v(highQualityParts).
gs(servicePerformed), gs(highQualityParts)?
Solutions:
1: {do(receiptSent), do(repairPerformed), do(damagesReported), do(partsPaid), do(repairReported), e(c1), e(c3), e(c4), e(c5)}

Query 3: Can Tony support "service performed" with "high quality" and "low cost"?
v(servicePerformed). v(highQualityParts). v(lowCostService).
gs(servicePerformed), gs(lowCostService), gs(highQualityParts)?
Solutions: none

Table 5. Queries (and their solutions) against Tony's insurance-engagement bound specification

Tony can make commitment c_5 to a customer in order to support insurance validated; he can get c_4 from supplier since Tony supports the antecedent by using his capability for parts paid. In order to get commitments c_1 and c_3, Tony has to chain commitments: get or make other commitments in order to support the antecedent of another commitment. Tony can get c_3 from an assessor by using his capability for damages reported and chaining c_5 to support insurance validated. Tony can get c_1 from insurer by using his capability for repair reported and chaining c_5 for insurance validated. Solution 1 contains redundant ways to achieve a goal, thus might be useful in order to ensure reliability. Solution 2 involves buying new parts only, and it has the same commitments of solution 1. Solution 4 involves fixing old parts only. Solution 3 includes fixing old parts and also paying new parts, but not c_4. This option is legitimate, even though not a smart one. Notice how solution 1 and solution 3 are not minimal: indeed, solution 2 is a subset of solution 1, while solution 4 is a subset of solution 3.

Query 2 Can Tony support service performed with high quality?

Verifying this query corresponds to checking goal support for the conjunction of the two goals. This means that goal high quality is in the scope. The effect of this modification is that three solutions for Query 1 are not valid for Query 2. The only valid solution is the former Solution 2. The reason why the other three solutions are not valid is simple: they include goal old parts fixed, which contributes negatively (--S) to high quality.

Query 3 Can Tony support service performed with high quality and low cost?

The third query adds yet another goal in the scope, namely low cost. The effect is that Tony cannot support the conjunction of the three queried goals. The reason for this is that goal new parts provided has a negative contribution to low cost, therefore the only valid solution for Query 2 is not valid for Query 3.

4.2 Scalability experiments

We evaluated the applicability of our reasoning to medium- and large-sized scenarios by performing some experiments on goal models and service engagements of growing size. Our tests are not intended to assess the absolute performance of the reasoner, rather they aim to check empirically if the query execution time grows linearly or exponentially with the size of the problem.

We create our experiments using a scenario cloning technique: a basic building block is cloned to obtain larger scenarios. The building block consists of a goal model with 9 goals (with one top-level goal, 3 and-decompositions and 1 or-decomposition) and a service engagement with 2 commitments. Cloning this scenario produces a new scenario with 2 top-level goals, 18 goals and 4 commitments; another cloning operation outputs 3 top-level goals, 27 goals and 6 commitments, and so on. The posed query consists of the conjunction of all the top-level goals in the cloned scenario.

Note that cloning linearly increases the number of goals and commitments, whereas it exponentially increases the number of solutions. Cloning is a useful technique to check scalability for our reasoning, given that the important thing is not the number of goals and commitments but the number of solutions. The cloned scenario is characterized by high variability.

# goals	# comms	# solutions	time (s)	$\frac{time}{\#sol}$ (μs)
9	2	5	0.009	1866
18	4	25	0.013	533
27	6	125	0.033	266
36	8	625	0.112	179
45	10	3125	0.333	107
54	12	15625	1.361	87
63	14	78125	7.017	90
72	16	390625	37.043	95
81	18	1953125	199.920	102

Table 6. Experiments evaluating the scalability of goal support reasoning

The experiments were run on a machine with an AMD Athlon(tm) 64 X2 Dual Core Processor 4200+ CPU, 2GB DIMM DDR2 memory, Linux version 2.6.31-15-generic kernel, DLV-Complex linux static build 20090727. We executed three runs for every experiment; we consider the average time; time was measured using the linux *time* utility and summing the *user* and *sys* values.

Table 6 present the results of the scalability experiments. The first three columns show the number of goals, commitments and solutions, respectively. Notice how the number of solutions grows exponentially: the biggest experiment has almost two millions solutions. The fourth column shows the total time needed to run the experiment; the reasoning is applicable at design time to medium-large models, given that 2 millions solutions are computed in 200 seconds on a desktop computer. The most significant result, however, is in the last column. It shows the average time to derive one solution in microseconds. Notice how the time per solution does not grow exponentially. The average time for smaller experiments is higher because initialization time has a strong impact; time grows pseudo-linearly for bigger experiments.

5 Discussion

The principal contribution of this paper lies in formulating a conceptual model for service-oriented applications that accommodates the autonomy and heterogeneity of their participants, and applies naturally at a business level. We accomplish this by specifying the participants in terms of goals, and the engagements between them in terms of commitments. Our conceptual model has the following salient features. (1) Application architecture is specified in terms of commitments. (2) Commitment are conditional and capture the reciprocal nature of business relationships. (3) Commitments decouple agents: if an agent has a commitment from another, it may not care what goals the other has. We also encoded the reasoning relationship between goals and commitments in datalog, and applied it to a real car insurance scenario. The reasoning helps determine if a chosen service engagement is suitable for a participant's requirements.

Prominent goal-oriented methodologies such as Tropos [2] and KAOS [23] do not distinguish between application architecture and the requirements of individual agents. The reason is their basis in traditional information systems development where stakeholders cooperate in building a fully specified system. Gordijn *et al.* [10] combine goal modeling with profitability modeling for the various stakeholders; however, their approach shares the monolithic system-development point of view.

One may understand dependencies between *actors* in i* [24] as an architectural description of the application. However, dependencies do not capture business relationships as commitments do. Guizzardi *et al.* [11] and Telang and Singh [22] highlight the advantages of commitments over dependencies for capturing relationships between roles. Both Telang and Singh [22] and Gordijn *et al.* [10] especially note that dependencies do not capture the reciprocal nature of a business transaction. Bryl *et al.* [3] use a planning-based approach to explore the space of possible alternatives for satisfying some goal; however, unlike us, they follow goal dependencies inside the dependee actors, thus violating heterogeneity. Castro *et al.* [4] highlight the lack of modularity in goal models. Since commitments decouple agents, they significantly alleviate the modularity problem.

Compatibility between a participant and a service engagement is a different kind of correctness criterion compared to checking for progress or safety properties over procedural specifications, as is done for example, in [7]. Mahfouz *et al.* [14] consider the alignment between the goal model of an application, in terms of both dependencies between the actors and their internal goals, and the choreography under consideration. Their approach could be applied in the design of choreographies, that might be then made available as architectural specifications.

Abstractions such as goals and intentions have been used to describe services [13, 18]; however such approaches violate heterogeneity by making assumptions about other participants' internals. Specifications of service engagements are eminently more reusable than the goal models of actors [9]. Liu *et al.* [12] formalize commitments in a weaker sense—as a relation between an actor and a service, not between actors, as in done in our approach.

Workflow-based approaches for business processes, for example, [15, 17], capture interaction from the viewpoint of a single participant. As such, they may be used to code up individual agents—either as an alternative to goal models or as their operationalization. Benatallah *et al.* [1] formalize properties such the similarity and replaceability for choreographies. Although such approaches are valuable, they are at a lower level of abstraction than service engagements. Such properties have begun to be formalized for service engagements [20]. Especially interesting is the formalization of interoperability in terms of commitments in completely asynchronous settings [5]. The formalization therein completely obviates the need for control flow constructs in service engagements, for example, that an *Accept* or *Reject* should follow the *Order*.

One feature that distinguishes our model from some others in the literature, for example [6], is the emphasis on roles and participants as opposed to on the service itself. In our approach, a service is something that is *realized* when participants interact according to the service engagement. Notice that there is no "service" entity in our conceptual model.

Our approach opens up interesting directions of work. An agent would ideally monitor both his goals and commitments. Compliance with legal and contractual requirements may be formulated directly in terms of commitments, instead of in terms of following a process. An agent would adapt in case some goal is threatened by adopting new strategies; however, in doing so it should ideally also consider his outstanding commitments, else it risks being noncompliant.

Acknowledgments. Research partially supported by FP6-EU project SERENITY contract 27587.

References

1. Boualem Benatallah, Fabio Casati, and Farouk Toumani. Representing, analysing and managing web service protocols. *Data and Knowledge Engineering*, 58(3):327–357, 2006.
2. Paolo Bresciani, Anna Perini, Paolo Giorgini, Fausto Giunchiglia, and John Mylopoulos. Tropos: An agent-oriented software development methodology. *Autonomous Agents and Multi-Agent Systems*, 8(3):203–236, 2004.
3. Volha Bryl, Paolo Giorgini, and John Mylopoulos. Designing socio-technical systems: From stakeholder goals to social networks. *Requirements Engineering*, 14(1):47–70, 2009.
4. Jaelson Castro, Manuel Kolp, Lin Liu, and Anna Perini. Dealing with complexity using conceptual models based on Tropos. In *Conceptual Modeling: Foundations and Applications*, LNCS 5600 , 335–362. Springer, 2009.
5. Amit K. Chopra and Munindar P. Singh. Multiagent commitment alignment. In *Proceedings of AAMAS 2009*, 937–944, 2009.

6. Massimiliano Colombo, Elisabetta Di Nitto, Massimiliano Di Penta, Damiano Distante, and Maurilio Zuccaiá. Speaking a common language: A conceptual model for describing service-oriented systems. In *Proceedings of ICSOC 2005*, LNCS 3826, 48–60, 2005.
7. Gero Decker and Mathias Weske. Behavioral consistency for B2B process integration. In *Proceedings of CAiSE'07*, LNCS 4495, 81–95. Springer, 2007.
8. Frank DeRemer and Hans H. Kron. Programming-in-the-large versus programming-in-the small. *IEEE Transactions on Software Engineering*, 2(2):80–86, June 1976.
9. Nirmit Desai, Amit K. Chopra, and Munindar P. Singh. Amoeba: A methodology for modeling and evolution of cross-organizational business processes. *ACM Transactions on Software Engineering and Methodology*, 19(2), 2010.
10. Jaap Gordijn, Eric Yu, and Bas van der Raadt. E-service design using i* and e^{3}value modeling. *IEEE Software*, 23(3):26–33, 2006.
11. Renata S. S. Guizzardi, Giancarlo Guizzardi, Anna Perini, and John Mylopoulos. Towards an ontological account of agent-oriented goals. In *Software Engineering for Multi-Agent Systems V*, LNCS 4408, 148–164. Springer, 2007.
12. Lin Liu, Qiang Liu, Chi-Hung Chi, Zhi Jin, and Eric Yu. Towards a service requirements modelling ontology based on agent knowledge and intentions. *International Journal of Agent-Oriented Software Engineering*, 2(3):324–349, 2008.
13. Amy Lo and Eric Yu. From business models to service-oriented design: A reference catalog approach. In *Proceedings of ER 2007*, 87–101, 2007.
14. Ayman Mahfouz, Leonor Barroca, Robin Laney, and Bashar Nuseibeh. Requirements-driven collaborative choreography customization. In *Proceedings of ICSOC-ServiceWave*, 144–158, 2009.
15. Dinh Khoa Nguyen, Willem-Jan van den Heuvel, Mike P. Papazoglou, Valeria de Castro, and Esperanza Marcos. GAMBUSE: A gap analysis methodology for engineering SOA-based applications. In *Conceptual Modeling: Foundations and Applications*, LNCS 5600 , 417–435. Springer, 2009.
16. Elisabetta Di Nitto, Carlo Ghezzi, Andreas Metzger, Mike P. Papazoglou, and Klaus Pohl. A journey to highly dynamic, self-adaptive service-based applications. *Automated Software Engineering*, 15(3–4):313–341, 2008.
17. Chun Ouyang, Marlon Dumas, Wil M. P. Van Der Aalst, Arthur H. M. Ter Hofstede, and Jan Mendling. From business process models to process-oriented software systems. *ACM Transactions on Software Engineering and Methodology*, 19(1):1–37, 2009.
18. Colette Rolland, Rim Samia Kaabi, and Naoufel Kraïem. On ISOA: Intentional services oriented architecture. In *Proceedings of CAiSE'07*, 158–172, 2007.
19. Munindar P. Singh. Agent communication languages: Rethinking the principles. *IEEE Computer*, 31(12):40–47, December 1998.
20. Munindar P. Singh and Amit K. Chopra. Correctness properties for multiagent systems. In *Proceedings of Workshop on Declarative Agent Languages and Technologies*, LNCS 5948 , 192–207. Springer, 2009.
21. Munindar P. Singh and Michael N. Huhns. *Service-Oriented Computing: Semantics, Processes, Agents*. John Wiley & Sons, Chichester, UK, 2005.
22. Pankaj R. Telang and Munindar P. Singh. Enhancing Tropos with commitments: A business metamodel and methodology. In *Conceptual Modeling: Foundations and Applications*, LNCS 5600, 417–435. Springer, 2009.
23. Axel van Lamsweerde. From system goals to software architecture. In *Formal Methods for Software Architectures*, LNCS 2804, 25–43. Springer, 2003.
24. Eric S.K. Yu. Towards modelling and reasoning support for early-phase requirements engineering. In *Proceedings of ISRE'97*, 226–235, 1997.