Requirements as Goals and Commitments too

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

Abstract In traditional software engineering research and practice, requirements are classified either as functional or non-functional. Functional requirements consist of all functions the system-to-be ought to support, and have been modeled in terms of box-and-arrow diagrams in the spirit of SADT. Non-functional requirements include desired software qualities for the system-to-be and have been described either in natural language or in terms of metrics. This orthodoxy was challenged in the mid-90s by a host of proposals that had a common theme: all requirements are initially stakeholder goals and ought to be elicited, modeled and analyzed as such. Through systematic processes, these goals can be refined into specifications of functions the system-to-be needs to deliver, while actions assigned to external actors need to be executed. This view is dominating Requirements Engineering (RE) research and is beginning to have an impact on RE practice. We propose a next step along this line of research, by adopting the concept of conditional commitment as companion concept to that of goal. Goals are intentional entities that capture the needs and wants of stakeholders. Commitments, on the other hand, are social concepts that define the willingness and capability of an actor A to fulfill a predicate φ for the benefit of actor B, provided B (in return) fulfills predicate ψ for the benefit of actor A. In our conceptualization, goals are mapped to collections of commitments rather than functions, qualities, or actor assignments. We motivate the importance of the concept of commitment for RE through examples and discussion. We also contrast our proposal with state-of-the-art requirements modeling and analysis frameworks, such as KAOS, MAP, *i** and Tropos.

1 Introduction

Colette Rolland is an eminent researcher, mentor and leader in the Information Systems community thanks to a distinguished career that spans more than three decades. Her plethora of contributions include novel concepts, methods and tools for building information systems, as well as dozens of young researchers who will carry the torch of her ideas for years to come. One of those ideas that has had tremendous impact on the field is the notion that system requirements are stakeholder

goals—rather than system functions—and ought to be elicited, modeled and analyzed accordingly [21, 27, 28]. In this paper, we take this idea one small step farther.

In traditional software engineering research and practice, requirements are classified either as functional or nonfunctional. Functional requirements consist of all functions the system-to-be ought to support, and have been modeled and analyzed in terms of box-and-arrow diagrams in the spirit of SADT [32]. Nonfunctional requirements include desired software qualities for the system-to-be and have been described either in natural language or in terms of metrics. This orthodoxy was challenged in the mid-90s by a host of proposals that had a common theme: all requirements—functional and non-functional—are initially stakeholder goals, rather than functions. Through systematic processes, these goals can be refined into specifications of functions the system-to-be needs to deliver, whereas actions assigned to external actors need to be executed. This view is dominating Requirements Engineering (RE) research and is beginning to have an impact on RE practice.

The main objective of this paper is to propose a next step along this line of research, by adding the concept of conditional commitment as companion concept to that of goal. Goals are intentional entities that capture the needs and wants of stakeholders. Commitments, on the other hand, are social concepts that define the willingness and capability of actors to contribute to the fulfillment of requirements. Specifically, a conditional commitment involves two actors A and B, where A has committed to fulfill a predicate φ for the benefit of actor B, provided B (in return) fulfills ψ for the benefit A. In our conceptualization, goals are mapped to collections of commitments rather than functions, qualities, and actor assignments.

Our work is motivated by RE frameworks such as *i** [43] which are founded on the notion of actor and social dependencies between pairs of actors; also on Agent-Oriented Software Engineering (AOSE) frameworks such as Tropos [4], where design begins with stakeholder goals and proceeds through a refinement process to identify and characterize alternative designs (plans) that can fulfill these goals. The Tropos framework has been formalized for goals and their refinements [18], but not for goal fulfillment in a multiagent setting where commitments form the primary vehicle for goal fulfillment. We have striven to keep our proposal generic so that it applies not only to Tropos but also other frameworks where there is a need to reason with a collection of agents along with their goals and commitments.

We motivate the importance of the concept of commitment for RE through examples and discussion. We also contrast our proposal with state-of-the-art requirements modeling and analysis frameworks, such as KAOS [10], MAP [29], *i** and Tropos.

Our proposal is intended primarily for the development of socio-technical systems. Unlike their traditional computer-based cousins, such systems include in their architecture and operation organizational and human actors along with software ones, and are regulated and constrained by internal organizational rules,

business processes, external laws and regulations [15, 31]. Among the challenging problems related to the analysis and design of a socio-technical system is the problem of understanding the requirements of its software components, the ways technology can support human and organizational activities, and the way in which the structure of these activities is influenced by introducing technology. In particular, in a socio-technical system, human, organizational and software actors rely heavily on each other in order to fulfill their respective objectives. Not surprisingly, an important element in the design of a socio-technical system is the identification of a set of dependencies among actors which, if respected by all parties, will fulfill all stakeholder goals, the requirements of the socio-technical system.

This paper is structured as follows. Section 2 provides a comprehensive overview on commitments, specifically on their usage in multiagent systems. Section 3 illustrates how commitments can be used with goals to specify requirements, and introduces some reasoning principles. Section 4 exemplifies how the reasoning may be applied in a travel agency setting. Section 5 compares our model to related work. Finally, Section 6 concludes with a summary of our approach.

2 Commitments in Multiagent Systems

The concept of commitment spans many disciplines, from Philosophy of Mind, to Psychology, Sociology and Economics. A review of the literature suggests that the concept has only been studied in the later half of the last century (it is true: Aristotle did not discover everything!).

Commitments as a computational abstraction have a long history in Computer Science. Bratman [3] and Cohen and Levesque [9] formulated the notion of an agent's commitment to his intentions. Singh [33] labeled commitments of this nature as psychological commitments, and instead stressed the notion of *social commitment*, that is, commitments among agents. In particular, Singh showed that social commitments are key to modeling communication among agents [34], and consequently to the development of large systems consisting of autonomous, interacting entities—in other words, multiagent systems. In the following, the term *commitment* is used solely in the sense of a *social commitment*.

Singh [35] also elucidated the key ontological aspects of commitments. Since then, commitments have been applied as a basis for flexible interaction among agents [41, 42]; towards the formulation of agent communication languages [17], as an abstraction for business process design [11, 14]; towards a type theory for protocols [6, 24]; towards understanding interoperability among agents [6, 7]; and towards formulating a service-oriented architecture [38]. Aspects related to reasoning about commitments have been addressed in [7, 13, 16, 36]. Commitments have also been recently applied in requirements engineering [39], and for monitoring in conjunction with goals [26].

Below, we characterize multiagent systems especially emphasizing the value of commitments.

2.1 Multiagent Systems

Multiagent Systems (MAS) are *open systems*: *autonomous and heterogeneous* entities known as agents participate in multiagent systems. An agent's autonomy means that no agent has control over it. An agent's heterogeneity means that an agent's internal construction is inaccessible to other agents. An agent may be a human, organization, or some stakeholder projected into the system as software. It is worth emphasizing that socio-technical systems are, first and foremost, multiagent systems.

The purpose of the *system*, specifically, is to provide a basis for coherent interactions among agents in spite of their autonomy. Indeed, the system may be specified independently of the agents [37]. The system itself serves as the specification, from a global perspective, of the legitimate expectations that agents adopting roles in the system would have of each other. In other words, the system is the *protocol* (MAS terminology), or *specification* (RE terminology).

We specify expectations in terms of commitments. An agent that does not fulfill its commitments to others is noncompliant. Compliance balances autonomy. An agent may do as it pleases, but from the system's perspective it may be noncompliant. Example 1 illustrates these concepts.

Example 1. A *housing contract* is a system that specifies the commitments that govern interaction between a tenant and the landlord, both agents. For example, the contract may say that the tenant may not accommodate other persons on the property unless he seeks permission from the landlord. However, the tenant, in noncompliance with the clause, may on occasion host visiting family members. It does not matter whether the landlord knows of the violation; what matters is that from the system perspective, there is a violation.

The question of the basis of compliance goes to the heart of multiagent systems research. The answer lies in how systems (protocols) are specified. Systems specified in terms of control and data flow impose strong ordering and synchronization constraints on interaction; compliance for such specifications amounts to not violating such constraints, as Example 2 shows.

Example 2. Consider a scenario where Alice wants to buy a book from the bookseller EBook. The protocol (the system) they employ specifies that the delivery of the book must precede payment. If Alice pays first, she would be noncompliant with the protocol.

Systems specified in terms of intentional abstractions such as goals and beliefs are brittle because they lead to strong assumptions about an agent's construction [34].

By contrast, system specification approaches based on commitments hit the right balance between over-abstraction (exemplified by goal-oriented approaches) and under-abstraction (exemplified by process-oriented ones). Goal-oriented approaches model desired states of the world without saying who is responsible for

doing what in achieving them. Process-oriented approaches, on the other hand, specify specific courses of action that are often violated by the actual actions undertaken by relevant agents. Commitments specify interaction at a high level of abstraction. They signify social relationships between agents and can be inferred solely from the observable communication between agents. Moreover, compliance for an agent simply means satisfying the commitments he has toward others [34]. We elaborate on commitments in the following.

2.2 The Concept of Commitment

A commitment is of the form C(debtor, creditor, antecedent, consequent), where debtor and creditor are agents, and antecedent and consequent are propositions. A commitment C(*x*, *y*, *r*, *u*) means that *x* is committed to *y* that if *r* holds, then it will bring about *u*. If *r* holds, then C(*x*, *y*, *r*, *u*) is *detached*, and the commitment C(*x*, *y*, T, *u*) holds (T being the constant for truth). If *u* holds, then the commitment is *discharged* and doesn't hold any longer. All commitments are *conditional*; an unconditional commitment is merely a special case where the antecedent equals T. Examples 3–5 illustrate these concepts. In the examples, EBook is a bookseller, and Alice is a customer; let *BNW* and $12 refer to the propositions *Brave New World has been delivered* and *payment of* $*12 has been made*, respectively.

Example 3. (Commitment) C(EBook, Alice, $12, BNW) means that EBook commits to Alice that if she pays $12, then EBook will send her the book *Brave New World*.

Example 4. (Detach) If Alice makes the payment, that is, if $12 holds, then C(EBook, Alice, $12, BNW) is detached. In other words, C(EBook, Alice, $12, BNW) $\wedge$ $12 $\Rightarrow$ C(EBook, Alice, T, BNW).

Example 5. (Discharge) If EBook sends the book (if BNW holds), then both C(EBook, Alice, $12, BNW) and C(EBook, Alice, T, BNW) are discharged. That is to say, BNW $\Rightarrow \neg$ C(EBook, Alice, T, BNW) $\wedge \neg$ C(EBook, Alice, $12, BNW).

Importantly, an agent can manipulate commitments by performing certain operations (technically, speech acts). The commitment operations are reproduced below (from [35]). Create, Cancel, and Release are two-party operations, whereas Delegate and Assign are three-party operations.

- Create(*x*, *y*, *r*, *u*) is performed by *x*, and it causes C(*x*, *y*, *r*, *u*) to hold.
- Cancel(*x*, *y*, *r*, *u*) is performed by *x*, and it causes C(*x*, *y*, *r*, *u*) to not hold.
- Release(*x*, *y*, *r*, *u*) is performed by *y*, and it causes C(*x*, *y*, *r*, *u*) to not hold.
- Delegate(*x*, *y*, *z*, *r*, *u*) is performed by *x*, and it causes C(*z*, *y*, *r*, *u*) to hold.
- Assign(*x*, *y*, *z*, *r*, *u*) is performed by *y*, and it causes C(*x*, *z*, *r*, *u*) to hold.

We introduce Declare(*x*, *y*, *r*) as an operation performed by *x* to inform *y* that *r* holds. This is not a commitment operation, but may indirectly affect commitments

by causing detaches and discharges. In relation to Example 4, when Alice informs EBook of the payment by performing Declare(Alice, EBook, $12), then the proposition $12 holds, and causes a detach of C(EBook, Alice, $12, BNW).

A deductive strength relation can be defined between commitments [7]: C(*x*, *y*, *r*, *u*) is stronger than C(*x*, *y*, *s*, *v*) if and only if *s* entails *r* and *u* entails *v*. So, for instance, a detached commitment C(*x*, *y*, T, *u*) is stronger than the commitment before detachment C(*x*, *y*, *r*, *u*).

A commitment arises in a social or legal context. The context defines the rules of encounter among the interacting parties, and often serves as an arbiter in disputes and imposes penalties on parties that violate their commitments. For example, eBay is the context of all auctions that take place through their service; if a bidder does not honor a payment obligation for an auction that it has won, eBay may suspend the bidder's account.

2.2 System Specification: Protocols

Traditional approaches describe a protocol in terms of the occurrence and relative order of specific messages.

The protocol of Fig. 1 begins with EBook sending Alice an offer. Alice may either accept or reject the offer. If she rejects it, the protocol ends; if she accepts it, EBook sends her the book. Next, Alice sends EBook the payment. Because an FSM ignores the meanings of the messages, it defines compliance based on low-level considerations, such as the order in which commitments are fulfilled. Moreover, this type of specification is often inflexible. As illustrated in Example 2, Alice fails to comply if she sends the payment before she receives the book. Note that this drawback applies to all process-oriented specification languages used for specifying rich social concepts such as business processes (e.g. BPMN [2] and BPEL [1]).

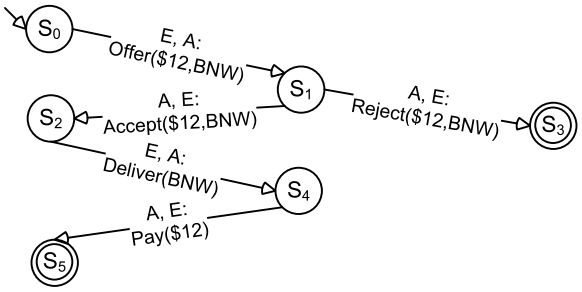

Fig. 1. A purchase protocol as a finite state machine, taken from [7]. Each message is tagged with its sender and receiver (here and below, E is EBook; A is Alice).

In contrast, we build on *commitment protocols* [42], which describe messages along with their *business meanings*. Commitment operations are realized in distributed systems by the corresponding messages. Commitment protocols are there-

fore defined in terms of the operations introduced above: Create, Cancel, Release, Delegate, Assign, and Declare. We introduce an abbreviation. Let c = C(x, y, r, u). Then, we Create(c) abbreviates Create(x, y, r, u).

Table 1 shows an alternative purchase protocol specified in terms of commitments. The semantics of domain-specific messages are explained in terms of commitment operations. For example, an *Offer* message is interpreted as a Create operation, whereas a *Reject* message releases the debtor from the commitment.

Table 1 A purchase protocol expressed in terms of commitments

Domain-Specific Message	Commitment-Oriented Message
Offer(E,A, $12, BNW)	Create(E, A, $12, BNW)
Accept(A,E,BNW, $12)	Create(A, E, BNW, $12)
Reject(E,A, $12,BNW)	Release(E, A, $12, BNW)
Deliver(E,A,BNW)	Declare(E, A, BNW)
Pay(A,E, $12)	Declare(A, E, $12)

Table 2 introduces the commitments used in Fig. 2 and Fig. 3.

Table 2. Commitments used as running examples in this paper

Name	Commitment
c_A	C(Alice, EBook, BNW, $12)
c_B	C(EBook, Alice, $12, BNW)
c_{UA}	C(Alice, EBook, T, $12)
c_{UB}	C(EBook, Alice, T, BNW)

Let us walk through the interaction of Fig. 2, which shows a possible enactment of the protocol described in Table 1. Upon sending Create(c_B), EBook infers c_B; upon receiving the message Alice infers c_B. Upon sending Declare($12), Alice infers that $12 holds. Consequently, she infers that c_B is detached, yielding c_{UB}. When EBook receives Declare ($12), it infers c_{UB}. EBook finally sends Declare (BNW), thus concluding that its commitment is discharged. When Alice receives Declare(BNW), she draws the same inference.

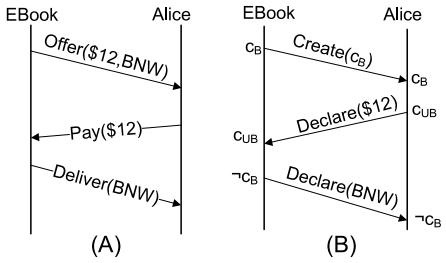

Fig. 2. An enactment of the protocol of Table 1 in terms of (A) domain-specific messages and (B) commitments. We show only the strongest commitments at each point. For example, because c_{UB} is stronger than c_B, c_{UB} is sufficient.

Notice that Table 1 does not specify any ordering constraints on messages. In effect, *each party can send messages in any order*. Fig. 3 shows some additional enactments of the purchase protocol of Table 1. Neither the enactments of Fig. 3(B) and Fig. 3(C) nor the one in Fig. 2 are legal according to the FSM in Fig. 1.

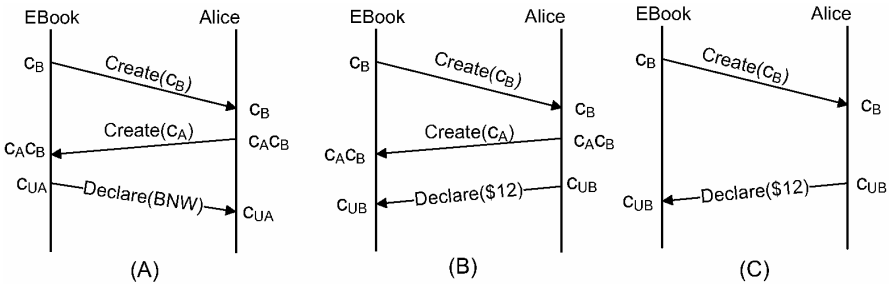

Fig. 3. Three possible enactments of the protocol of Table 1.

So when is an agent compliant with a protocol? The answer is simple: an agent complies if its commitments are discharged, no matter if delegated or otherwise manipulated. Traditional approaches force a tradeoff: checking compliance is simple with rigid automaton-based representations and difficult with flexible reasoning. Protocols specified using commitments find the golden mean, promoting flexibility by constraining interactions at the business level, yet providing a rigorous notion of compliance.

2.4 Architecture, Interoperability, and Middleware

In the discussion above, we used examples where the commitments are defined over specific agents (for example, Alice and EBook). General protocols can be defined by stating the commitments among roles instead of agents. For example, we can replace Alice with Customer and EBook with Vendor and use the commitments of the previous sections to specify a general protocol for commercial transactions. These generic protocols can then be used in a specific context by binding a specific agent to each role of the protocol.

Protocols are architectural specifications: they specify the interconnections between agents (via roles). Commitment protocols abstract away from considerations of control and data flow, instead focusing on the contractual relationships among agents. This affords agents flexibility in protocol enactment. However, flexibility poses challenges for interoperability: if an agent may send any message at any time, how do we ensure that they will come to the same conclusion about their commitments towards each other? Example 6 illustrates a case of *misalignment*.

Example 6. Assume both Alice and EBook infer c_B. Subsequently, Alice's payment for the book and EBook's cancellation of the offer c_B cross in transit (we are

dealing with distributed systems). When Alice receives EBook's cancellation, she considers it as having arrived too late; EBook considers Alice's payment late. Thus, Alice concludes c_{UB}, whereas EBook does not—they are misaligned.

Interoperability concerns are addressed in [6, 7] via the notion of commitment alignment. Alignment expresses the intuition that whenever a creditor computes (that is, infers) a commitment, the presumed debtor also computes the same commitment. If agents get misaligned, their interaction will potentially break down. Traditionally, interoperability among services has been captured in terms of whether agents can *send* and *receive* messages in a compatible manner—for example, in terms of (the absence of) deadlocks. Such formalizations of interoperability are useful, but work at a lower level than commitments. Two agents may be aligned commitment-wise, but deadlocked because they are both waiting for the other to act. Conversely, agents may be *live*, but misaligned.

Alignment motivates a middleware that maintains and monitors commitments, and transparently takes necessary actions to maintain alignment [8]. For example, the middleware would transparently notify the debtors when an event occurs that detaches a commitment; otherwise, in a distributed system where different agents have likely observed different events, agents could get easily misaligned. Compare this to what traditional middleware, for example, reliable message queues, do. They send acknowledgments, store messages until they are consumed, maintain message order, and so on, in other words, do the bookkeeping to maintain interoperability. A commitment-oriented middleware would do the bookkeeping at a high level, relegating messages queues to a lower level.

The middleware would ideally be able to monitor goals and commitments, reason about compliance and interoperability, and support adaptations. In essence, the middleware would encode a business semantics and form a common substrate for all kinds of business applications. The middleware would offer a new programming model: it will support writing services directly in terms of goals and commitments, and will alleviate greatly the burden of writing agents.

3 From Goals to Commitments

Let us begin by summarizing the above discussion about commitments.

- Commitments abstract over data and control flow.
- Commitments are a social abstraction—being grounded in interaction, they encode publicly verifiable relationships among agents.
- Commitments support a notion of compliance that enables an agent to act flexibly.
- Protocols, and thus systems, may be specified as the commitments that may arise among the agents participating in the system.
- Commitments may be supported in middleware: this includes monitoring and reasoning for the purposes of compliance and interoperability.

The parallel with the notion of *goals* as studied in RE may already be obvious.

- Goals abstract over data and control flow specifications.
- Goals represent the particular states of the world an agent wants to achieve.
- Goals are also used in reasoning about flexibility and adaptability, especially in terms of the *variants* supported by a goal model.
- Agents may be specified in terms of abstractions such as goals, capabilities, and so on.
- Goals may also be supported in middleware: an agent can monitor its goals and act in order to achieve them.

Goals and commitments are complementary. An agent has certain goals that it wants to satisfy, and in doing so it typically must make (to others) or get (from others) commitments about certain goals. Alternatively, an agent has commitments to others (and a goal to comply), and it then adopts specific goals in order to discharge its commitments.

Thus, there are two things that an agent designer or the agent itself, by introspection at runtime, may do.

First, an agent may induce a protocol—the set of commitments—that are necessary to supports the goals it wants to achieve. The agent would additionally publish the protocol along with the role it has adopted in the protocol, and possibly invite others to adopt the other roles in the protocol or just wait to be discovered. Example 7 illustrates this method.

Example 7. Alice has the goal *BNW*. Alice figures that to get the book, it must interact with a bookseller and pay the bookseller for the book. So Alice induces a protocol with two roles, customer and merchant, with the commitment C(customer, merchant, BNW, payment). She adopts customer, and publishes the protocol as her interface. Eventually, a seller may sell *BNW* to Alice by playing role merchant.

Second, an agent may select a protocol from a repository. This recognizes the fact that protocols are reusable specifications of interaction [14]. Indeed, this is the case with many standardized protocols such as for financial transactions [12]. An agent would naturally want to verify if a protocol selected from some repository were suitable for the achievement of the agent's goals. The agent would also want to verify that if he makes a certain commitment, then his goals support fulfillment of the commitment.

The notion of compliance with a protocol helps decouple one agent's specification from another agent's. For example, a merchant would only care (perhaps modulo other properties deriving from interaction such as trust and reputation) that Alice is committed to payment for the book, irrespective of whether Alice actually intends to pay. In other words, if an agent commits to another for something, from the perspective of the latter, it does not matter much what the former's goals are or how the former will act to bring about the goal he committed to.

We now sketch some elements of the reasoning one can perform with goals and commitments. Given some role in a protocol 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 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 We illustrate the basic intuitions with examples.

An agent's goal is supported if the agent has a capability for it (Example 8).

Example 8. Consider Alice's goal payment. Alice supports the goal if she has a capability for it.

An agent's goal is supported if it can get an appropriate commitment from some other agent about the state of affairs that the goal represents (Example 9).

Example 9. Consider Alice's goal BNW. The commitment C(merchant, Alice, payment, BNW) from some merchant supports the goal, but only if Alice supports payment. The intuition is that Alice won't be able to exploit the merchant's commitment unless she pays.

An agent's goal is supported if it can make a commitment to some other agent for some state of affairs (presumably one that the latter would be interested in) if the latter brings about the state of affairs that the goal represents (Example 10).

Example 10. Consider EBook's goal payment. He can support this goal by making an offer to some customer, that is, by creating C(EBook, customer, payment, BNW).

The intuitions may be applied recursively for decomposition in goal trees. Thus for example, if an agent wants to support g, and g is and-composed into g_0 and g_1, then the agent would want to verify support for both g_0 and g_1, and so on.

Commitment support It makes sense to check whether an agent will be able to support the commitments it undertakes towards others.

Commitment support reduces to goal support for the commitment consequent (Example 11).

Example 11. Consider that C(EBook, customer, payment, BNW) holds. EBook supports his goal payment by the commitment; however, if he does not support BNW, then if the customer pays, he risks being noncompliant.

We consider goal and commitment support as separate notions. 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) [**chaining**] x can reason that C(x, y, g_0, g_1) is supported by C(z, x, g_2, g_1) if x supports g_2; (ii) [**division of labor**] 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) [**redundancy**] to support g, x may get commitments for g from two different agents; and so on.

4 Applying Goals and Commitments

We show how the conceptual model and the reasoning techniques can be used to represent and analyze a setting concerning flight tickets purchase via a travel agency. Four main roles participate in this protocol: travel agency, customer, airline, and shipper. Customers are interested in purchasing flight tickets for some reason (e.g. holidays or business trips), travel agencies provide a tickets-selling service to customers by booking flight tickets from airlines, shippers offer a ticket delivery service.

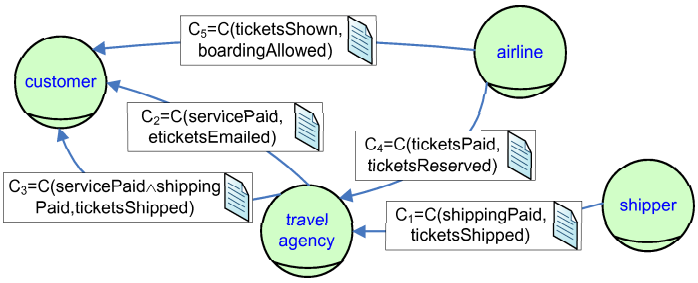

Fig. 4. Role model for the travel agency scenario. Commitments are rectangles that connect (via directed arrow) a debtor to a creditor

Fig. 4 describes the protocol in the travel agency scenario. The protocol is defined as a set of roles (circles) connected via commitments; the commitments are labeled (C_i). Table 3 explains the commitments.

Fig. 5 shows the situation where agent Fly has adopted the role travel agency in the protocol of Fig. 4; the other roles are not bound to agents. Fly has one top-level goal: selling tickets (ticketsSold). In order to support it, three sub-goals should be supported: tickets should be obtained, tickets should be delivered to the customer, and the service should be paid. Tickets can be obtained if the tickets are reserved and they have been paid. Fly is capable of goal ticketsPaid. There are two ways to deliver tickets: either electronic tickets are e-mailed or tickets are posted. In order to send tickets via mail, Fly has to ship the tickets and pay for the shipping. Fly is capable of eticketsEmailed. E-mailing tickets contributes positively (++S) [18] to softgoal costsKeptLow, whereas sending via shipping contributes negatively (--S) to such softgoal.

Table 3. Commitments in the travel agency protocol

Label	Description
C_1	shipper to travel agency: if the shipping cost have been paid, the flight tickets will be shipped
C_2	travel agency to customer: if the booking service has been paid, the electronic tickets will be e-mailed
C_3	travel agency to customer: if the booking service and the shipping cost have been paid, flight tickets will be shipped
C_4	airline to travel agency: if flight tickets have been paid, tickets will be reserved
C_5	airline to customer: if tickets have been shown, flight boarding will be allowed

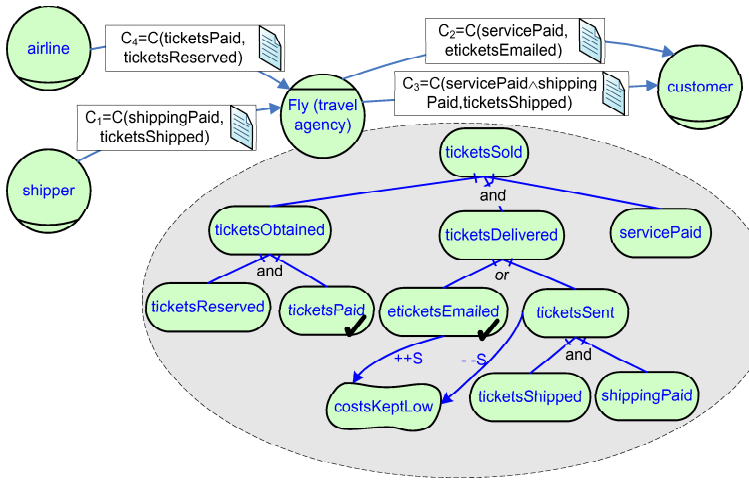

Fig. 5. Visual representation of Fly's travel agency-bound specification

We present now some queries concerning goal and commitment support that can be run against the specification of Fig. 5.

Query 1 Can Fly support goal ticketsSold?

The answer to this query is yes. Fly can support ticketsObtained by using its capability for ticketsPaid and getting C_4 from some airline. Fly supports ticketsDelivered via its capability for eticketsEmailed. Fly can support servicePaid by making C_2 to some customer.

An alternative solution involves sending tickets via shipping. Fly could support ticketsShipped and shippingPaid if it makes C_3 to a customer (which supports servicePaid and shippingPaid) and get C_1 from some shipper (to support ticketsShipped).

Another solution includes supporting both eticketsEmailed and ticketsSent: both C_2 and C_3 are made to the customer.

Query 2 Can Fly support goals ticketsSold and costsKeptLow?

This query adds an additional constraint to Query 1: supporting softgoal costsKeptLow. The only solution is when tickets are e-mailed: eticketsEmailed contributes positively to costsKeptLow and the softgoal gets no negative contribution. Posting tickets does not work: ticketsSent contributes negatively to costsKeptLow.

Query 3 Can Fly support commitment C_3 to customer?

As observed before, commitment support reduces to goal support. Thus, let's check whether Fly can support ticketsShipped if the antecedent of C_3 (servicePaid and shippingPaid) holds. Given the goal tree hierarchy of Fig. 5, the three goals that relate to C_3 are children of the top-level goal ticketsSold. The second solution of Query 1 tells us that Fly can support C_3 as it contains all such goals.

5 Discussion

Goal-oriented requirements engineering methodologies have been conceived with a traditional view of software in mind. They are adequate to design systems where stakeholders cooperate in a fully specified environment, but they are not thought for open systems composed of autonomous and heterogeneous participants.

The MAP approach [29] is describes processes in terms of *intentions* and *strategies*: a map is a directed graph where nodes are intentions and directed arrows represent strategies. A strategy explains how to achieve one intention starting from another intention. Maps have been recently used to define the concept of Intentional Services Oriented Architecture (ISOA) [30], where the authors conceive services in terms of intentional abstractions such as goals. In our approach, we model agents as goal-driven entities. However, we place emphasis on the modeling of the system itself via the social abstraction of commitments.

The *i** framework [43] starts from the identification of the stakeholders in the analyzed organizational setting and model these stakeholders—actors—in terms of their own goals and the dependencies between them. However, as concerns open settings such as socio-technical systems, *i** suffers from two primary drawbacks.

One, dependencies do not capture business relationships as commitments do. Guizzardi *et al.* [20] and Telang and Singh [39] highlight the advantages of commitments over dependencies for capturing relationships between roles. Both Telang and Singh and Gordijn *et al.* [19] especially note that dependencies do not capture the reciprocal nature of a business transaction.

Two, the strategic rationale model violates the heterogeneity principle by making assumptions about the goals of others actors. Commitments, by contrast, obviate looking inside an actor; as mentioned above, they completely decouple agents.

*i** has been recently used to describe services [23]; this approach violates agents heterogeneity by making assumptions about other participants' internals. Commitment protocols are more reusable than the goal models of actors [14].

Tropos [4] builds on top of *i** and adds models and concepts to be used in the development phases that follow requirements engineering. Being a derivative of *i**, Tropos suffers of the same problem concerning dependencies. Tropos provides an architectural model for the agents to develop, but exploits a weak notion of agency. Agents are designed and implemented under the assumption that they will cooperate with others. Our proposal differs in that cooperation is guaranteed by mutual interest in a commitment: the agents playing debtor and creditor have their own reasons to interact via commitments, but they don't (and can't) know the other party's motivations. Penserini *et al.* [25] have extended Tropos to design web services that support the stakeholders' goals. The main limitation of this approach is that it assumes that requirements engineers have a global view on all the actors.

KAOS [10] exploits a system-oriented perspective to specify requirements. Stakeholders are essential to gather system goals, but they are not explicitly represented in KAOS models. Leaf level goals are assigned to agents on the basis of a responsibility principle; van Lamsweerde has also discussed how KAOS requirements models can be mapped to software architecture [40]. KAOS is effective for the development of traditional software systems, but lacks of the proper abstractions to design autonomous and heterogeneous agents in open systems.

Gordijn *et al.* [19] combine *i** goal modeling with profitability modeling for the various stakeholders to design e-services. In such a way, the authors consider not only the intentions of the agents, but also the economic value of a service. Their approach is less generic than ours: economic value exchanges are a very important criteria but not the only one; moreover, they assume a monolithic system-development point of view which does not suit well in open systems.

Liu *et al.* [22] propose an *i** extension intended for the design of open systems, and propose some reasoning techniques that can be executed against these models. The authors formalize commitments in a weaker sense—as a relation between an actor and a service, not between actors, as is done in our approach.

Bryl *et al.* [5] use a planning-based approach to design socio-technical systems. The main intuition behind this work is 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.

6 Conclusions

The power of any technique for eliciting, modeling and analyzing requirements rests on the primitive concepts used to conceptualize them. The advent of goal-orientation in RE twenty years ago brought about a shift from a functional to an intentional view of software systems. The implications of this shift are still being worked out.

This paper advocates a further shift from an intentional to a social view of requirements for socio-technical systems. The proposal continues along a path originally defined by *i** in Eric Yu's PhD thesis. Our new proposal is founded on the

concept of commitment and related social concepts; it calls for a new form of system specification that prescribes a system's course of action more concretely than goal-oriented techniques, but more abstractly than process-oriented ones. We see this proposal as yet another step towards an agent-oriented view of socio-technical systems, their conceptualization, design, and evolution.

References

1. BPEL: Business process execution language for web services, version 1.1 (May 2003) www-106.ibm.com/developerworks/webservices/library/ws-bpel.
2. BPMN: Business process modeling notation, v1.1 (January 2008) http://www.bpmn.org/.
3. Bratman ME (1987) Intention, Plans, and Practical Reason. Harvard University Press, Cambridge, MA
4. Bresciani P, Perini A, Giorgini P, Giunchiglia F, Mylopoulos J (2004) Tropos: An agent-oriented software development methodology. Autonomous Agents and Multi-Agent Systems **8**(3): 203–236
5. Bryl V, Giorgini P, Mylopoulos J (2009) Designing socio-technical systems: From stakeholder goals to social networks. Requirements Engineering **14**(1): 47–70
6. Chopra AK, Singh MP (2008) Constitutive interoperability. In: Proceedings of the Seventh International Conference on Autonomous Agents and Multiagent Systems, pp.797–804
7. Chopra AK, SinghMP (2009) Multiagent commitment alignment. In: Proceedings of the Eighth International Conference on Autonomous Agents and MultiAgent Systems, pp.937–944
8. Chopra AK, Singh MP (2009) An architecture for multiagent systems: An approach based on commitments. In: Proceedings of the Seventh international Workshop on Programming Multi-Agent Systems
9. Cohen PR, Levesque HJ (1990) Intention is choice with commitment. Artificial Intelligence **42**: 213–261
10. Dardenne A, van Lamsweerde A, Fickas S (1993) Goal-directed requirements acquisition. Science of Computer Programming **20**(1–2): 3–50
11. Desai N, Mallya AU, Chopra AK, Singh MP (2005) Interaction protocols as design abstractions for business processes. IEEE Transactions on Software Engineering **31**(12): 1015–1027
12. Desai N, Chopra AK, Arrott M, Specht B, Singh MP (2007) Engineering foreign exchange processes via commitment protocols. In: Proceedings of the 4th IEEE International Conference on Services Computing, Los Alamitos, IEEE Computer Society Press, pp.514–521
13. Desai N, Chopra AK, Singh MP (2007) Representing and reasoning about commitments in business processes. In: Proceedings of the 22nd Conference on Artificial Intelligence. pp.1328–1333
14. Desai N, Chopra AK, Singh MP (2010) Amoeba: A methodology for modeling and evolution of cross-organizational business processes. ACM Transactions on Software Engineering and Methodology **19**(2)
15. Emery FE (1959) Characteristics of sociotechnical systems. London: Travistock Institute of Human Relations
16. Fornara N, Colombetti M (2002) Operational specification of a commitment-based agent communication language. In: Proceedings of the 1st International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS), ACM Press, pp.535–542
17. Fornara N, Colombetti M (2004) A commitment-based approach to agent communication. Applied Artificial Intelligence **18**(9-10): 853–866
18. Giorgini P, Mylopoulos J, Nicchiarelli E, Sebastiani R (2003) Reasoning with goal models. In: Conceptual Modeling—ER 2002. LNCS pp.167–181, Springer

19. Gordijn J, Yu E, van der Raadt B (2006) E-service design using *i** and e3value modeling. IEEE Software **23**(3): 26–33
20. Guizzardi RSS, Guizzardi G, Perini A, Mylopoulos J (2007) Towards an ontological account of agent-oriented goals. In: Software Engineering for Multi-Agent Systems V. LNCS 4408, pp.148–164, Springer
21. Kaabi RS, Souveyet C, Rolland C (2004) Eliciting service composition in a goal driven manner. In: Proceedings of the 2nd International Conference on Service Oriented Computing, pp.308–315
22. Liu L, Liu Q, Chi CH, Jin Z, Yu E (2008) Towards a service requirements modelling ontology based on agent knowledge and intentions. International Journal of Agent-Oriented Software Engineering **2**(3): 324–349
23. Lo A, Yu E (2007) From business models to service-oriented design: A reference catalog approach. In: Proceedings of the 26th International Conference on Conceptual Modeling (ER 2007), pp.87–101
24. Mallya AU, Singh MP (2007) An algebra for commitment protocols. Journal of Autonomous Agents and Multi-Agent Systems **14**(2): 143–163
25. Penserini L, Perini A, Susi A, Mylopoulos J (2006) From stakeholder needs to service requirements. In: Workshop on Service-Oriented Computing: Consequences for Engineering Requirements (SOCCER'06)
26. Robinson WN, Purao S (2009) Specifying and monitoring interactions and commitments in open business processes. IEEE Software **26**(2):72–79
27. Rolland C, Souveyet C, Achour CB (1998) Guiding goal modeling using scenarios. IEEE Transactions on Software Engineering **24**(12):1055–1071
28. Rolland C, Grosz G, Kla R (1999) Experience with goal-scenario coupling in requirements engineering. In: Proceedings of the IEEE International Symposium on Requirements Engineering
29. Rolland C, Prakash N, Benjamen A (1999) A multi-model view of process modelling. Requirements Engineering **4**(4): 169–187
30. Rolland C, Kaabi RS, Kraïem N (2007) On ISOA: Intentional services oriented architecture. In: Proceedings of the 19th International Conference on Advanced Information Systems Engineering, (CAiSE 2007), pp.158–172
31. Ropohl G (1999) Philosophy of socio-technical systems. Society for Philosophy and Technology **4**(3):55–71
32 Ross DT (1977) Structured analysis (SA): A language for communicating ideas. IEEE Transactions on Software Engineering **3**(1): 16–34
33. Singh MP (1991) Social and psychological commitments in multiagent systems. In: AAAI Fall Symposium on Knowledge and Action at Social and Organizational Levels, pp.104–106
34. Singh MP (1998) Agent communication languages: Rethinking the principles. IEEE Computer **31**(12): 40–47
35. Singh MP (1999) An ontology for commitments in multiagent systems: Toward a unification of normative concepts. Artificial Intelligence and Law **7**, pp.97–113
36. Singh MP (2008) Semantical considerations on dialectical and practical commitments. In: Proceedings of the 23rd Conference on Artificial Intelligence, pp.176–181
37. Singh MP, Chopra AK (2009) Programming multiagent systems without programming agents. In: Proceedings of the 7th International Workshop on Programming Multiagent Systems, (ProMAS 2009), invited paper
38. Singh MP, Chopra AK, Desai N (2009) Commitment-based service-oriented architecture. IEEE Computer **42**(11): 72–79
39. Telang PR, Singh MP (2009) Enhancing Tropos with commitments: A business metamodel and methodology. In Borgida A, Chaudhri V, Giorgini P, Yu E 8eds) Conceptual Modeling: Foundations and Applications. LNCS 5600, pp. 417–435, Springer
40. van Lamsweerde A (2003) From system goals to software architecture. In: Formal Methods for Software Architectures. LNCS 2804, pp. 25–43, Springer

41. Winikoff M, Liu W, Harland J (2005) Enhancing commitment machines. In: Proceedings of the 2nd International Workshop on Declarative Agent Languages and Technologies (DALT). LNAI 3476, pp.198–220, Springer
42. Yolum P, Singh MP (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, ACM Press, pp.527–534
43. Yu ES (1997) Towards modelling and reasoning support for early-phase requirements engineering. In: Proceedings of the Third IEEE International Symposium on Requirements Engineering, pp.226–235