Contracts + Goals = Roles?*

Lam-Son Lê and Aditya Ghose

School of Computer Science and Software Engineering
Faculty of Informatics, University of Wollongong
New South Wales 2522, Australia
{lle,aditya}@uow.edu.au

Abstract. The concept of role has been investigated in various fields of computer science as well as social sciences. While there is no clear consensus on how roles should be represented, a survey of the literature suggests that we should address both responsibilities and rights in the modeling of roles [1]. Based on this, we argue that the responsibilities and rights of roles can be captured by leveraging the notions of business contract and goal (in the sense of the goals of an actor being constrained by the rights associated with the role played by the actor) in the realm of requirements engineering. We leverage existing work on the formalization of business contracts [2] and the formulation of goals in the i^* modeling framework [3]. We devise formal techniques for reasoning about the composition and substitutability of roles and illustrate them through a running example.

Keywords: i^*, Business Contracts, Role Modeling, Formal Methods.

1 Introduction

Role modeling is a mechanism to separate concerns in the early phases of systems development (e.g. analysis) when developers have not actually built the system of interest. At this stage, model elements (e.g. components, agents, actors) that represent components of the system to be built do not yet exist. A role captures a coherent piece of behavior in the system. Roles may interact with one another in a process (in which case they are called *process roles*). A role could be regarded as an interface through which one might access the system or its subsystems (in which case it is called an *interface role*). In subsequent phases of the development process, designers assign roles to components. The representation of roles in the analysis and design phases thus provides critical input to the subsequent phases of the development lifecycle. This rationale also applies to social settings such as institutional design, where roles capture the expected behavior of positions that need to be filled by specific individuals or business entities. Systems (or institutions) can be designed/developed from scratch or can be obtained via the composition of pre-existing components (or via the composition of socio-technical

* Funding of this research was provided by the Smart Services CRC Initiative http://www.smartservicescrc.com.au/

P. Atzeni, D. Cheung, and R. Sudha (Eds.): ER 2012, LNCS 7532, pp. 252–266, 2012.

systems containing both system components and people, obtained through a combination of outsourcing, crowd-sourcing and recruitment). Either way, the components (or entities/people) must be capable of playing the roles that were assigned to them.

The so-called principle of separation of concerns matters in the requirements phase too. Unfortunately, the notion of role has received relatively little attention in the literature on requirements engineering (RE). For example, in the literature on the i^* framework, roles are simply regarded as abstract actors [4]. We take as a starting point the approach of Zhu et al [1] to role modeling where roles are associated with both responsibilities and rights. Given that systems often involve collaboration between roles, we need to understand the semantics of putting their responsibilities and rights together. Specifically, we need to answer the following two questions well before the system (or organization) of interest is built: (a) do the component roles of a system conjointly deliver what the system is expected to deliver? (b) how do we know if a role can safely substitute for another role without affecting the requirements of the system it belongs to?

To address these questions, we propose a formal framework for modeling and reasoning about process roles. We are inspired by the notion of socially-enhanced actors in the early-phase requirements that was made popular by the i^* framework [3], as well as the contract-based enterprise specification defined in the Reference Model of Open Distributed Processing (RM-ODP) standard [5]. Modeling roles in such a context involves representing (i) a contract a role is committed to (i.e. role's responsibility); (ii) the rights associated with a role that constrains the space of goals that an actor assuming that role can pursue. We leverage our previous work on semantic business processes, the work of Governatori et al [2] on business contracts and the well-established field of goal-oriented modeling in RE.

Paper Structure. Section 2 presents the background of our work. We discuss the representation of roles in RE and provides a running example in Section 3. Section 4 formally describes the framework. Section 5 surveys related work. Section 6 concludes the paper and outlines our future work.

2 Background

We base our work on semantic business processes, formalization of business contracts and the notion of role in a broad sense such as information systems.

2.1 Semantically Annotated Business Processes

An semantically annotated business process model is a process in which every task has been annotated with immediate effects. To determine the functionality delivered up to a given point of time during the occurrence of an annotated process, we reason about the cumulative effect. We suppose that analysts can associate context-independent effect to each step represented in the process. There

exists a technique that contextualizes these effects, i.e., to compute cumulative effects. The technique, called ProcessSEER, involves doing two stages of computation [6]. In the first stage, we derive a set of possible *scenario label*(s) for the given point in the process view. Each scenario label is a precise lits of steps that define a path leading from the start point to a the point being considered. In the second stage, the contiguous sequence of steps in each scenario label is taken into account to accumulate effects annotated to steps along this scenario in a pair-wise fashion.

A functionality annotation states the effect of having functionality delivered at a specified task. The effect can be textual. Alternatively, it could be written in first-order logic (FOL) or some computer-interpretable form. The total functionality delivered up to a certain task is the accumulation of all effects of the precedent tasks. We assume that the delivery annotations have been represented in conjunctive normal form (CNF) where each clause is also a prime implicate (this provides a non-redundant canonical form) [7]. The cumulative effect of tasks can inductively be defined as follows. The cumulative effect of the very first task is equal to its delivery annotation. Let $\langle Tk_i, Tk_j \rangle$ be an ordered pair of consecutive tasks such that Tk_i precedes Tk_j; let e_i be an effect scenario associated with Tk_i and e_j be the delivery annotation associated with Tk_j. Without loss of generality, we assume that e_i and e_j are sets of clauses. The resulting cumulative effect, denoted by $acc(e_i, e_j)$ is defined as follows.

- $acc(e_i, e_j) = e_i \cup e_j$ if $e_i \cup e_j$ is logically consistent
- Otherwise $acc(e_i, e_j) = e'_i \cup e_j$ whereby $e'_i \subseteq e_i$ such that $e'_i \cup e_j$ is consistent and we do not have any $e''_i \subseteq e'_i \subseteq e_i$ such that $e''_i \cup e_j$ is consistent

The task of accumulating functionality annotations is non-trivial since there might be various paths that can be traversed during the occurrence of the process up to the point of time being considered. We call the path leading to a certain point of time a *scenario label*. A scenario label can either be a sequence, denoted by the $\langle\rangle$ delimiters, or a set denoted by the $\{\}$ delimiters or combinations of both. The set delimiters are used to deal with parallel splits, and distinct elements in a set can be performed in any order [6]. Elements in a scenario label could be tasks (which have delivery annotations) or control elements (e.g. mutually exclusive split, parallel split). In addition to pair-wise effect accumulation across scenario labels, we need to make special provision for the following: (i) accumulation across AND-joins, and (ii) accumulation of effects over input/output flows.

2.2 Business Contract Modeling

Business contracts specify obligations, permissions and prohibitions as mutual agreements between business parties [8], as well as actions to be taken when a contract is violated. Governatori et al [2] have proposed such a contract modeling language which includes a non-boolean connective, $\otimes$, to represent contrary-to-duty obligations (i.e., what should be done if the terms of a contract are

violated). Deontic operators capture the contractual modality (i.e. obligations, permissions and prohibitions) [9]. Governatori et al represent a contractual rule as $r : A_1, A_2 \ldots A_n \vdash C$ where each A_i is an antecdent of the rule and C is the consequent. Each A_i and C may contain deontic operators but connectives can only appear in C.

As an example, $r : \neg p, q \vdash O_{seller}\alpha \otimes O_{seller}\beta$ is a contractual rule (identified by r) stating that if antecedents $\neg p$ and q hold, then a seller is obliged to make sure that α is brought about. Failure to do so results in a violation, for which a reparation can be made by bringing about β (the connective $\otimes$ can therefore be informally read as "failing which").

Definition 1. *[2] Contractual rules r and r' can be merged into rule r'' as follows where X denotes either an obligation or a permission.*

$$\frac{r : \Gamma \vdash O_s A \otimes (\bigotimes_{i=1}^{n} O_s B_i) \otimes O_s C \qquad r' : \Delta, \neg B_1, \neg B_2, \ldots, \neg B_n \vdash X_s D}{r'' : \Gamma, \Delta \vdash O_s A \otimes (\bigotimes_{i=1}^{n} O_s B_i) \otimes X_s D}$$

The $\otimes$ operator is associative but not commutative. This property matters when reasoning about the subsumption and merging of contractual rules. Definition 1 defines how contract rules might be merged. Governatori et al also devise a machinery for determining if one contractual rule subsumes another as presented in Definition 2.

Definition 2. *[2] Let's consider two rules $r_1 : \Gamma \vdash A \otimes B \otimes C$ and $r_2 : \Delta \vdash D$ where $A = \bigotimes_{i=1}^{m} A_i$, $B = \bigotimes_{i=1}^{n} B_i$ and $C = \bigotimes_{i=1}^{p} C_i$. Then r_1 subsumes r_2 (i.e. r_2 can safely be discarded if we have r_1) iff*

1. *$\Gamma = \Delta$ and $D = A$; or*
2. *$\Gamma \cup \{\neg A_1, \ldots, \neg A_m\} = \Delta$ and $D = B$; or*
3. *$\Gamma \cup \{\neg B_1, \ldots, \neg B_n\} = \Delta$ and $D = A \otimes \bigotimes_{i=0}^{k \leq p} C_i$*

2.3 Roles in Information Systems

Zhu et al. [1] provide a comprehensive survey of role modeling in various fields including object-oriented modeling, multi-agent systems, role-based access control, computer-supported cooperative work, social psychology and management. They suggest that role modeling should address both the responsibilities and rights of a role. They also define two categories of roles: interface roles and the process roles. The former is used in systems analysis and design while the latter in systems implementation.

Steimann [10] has studied role modeling mainly from the perspective of object-oriented modeling and identified 15 features of roles in relation to object types and relationships. He has also suggested merging the concepts of interface and role since they both refer to externally-visible behavior of an object that plays a certain role by realizing an interface [11]. In revisiting Bachman's role concept

in data modeling [12], Steimann [13] observes that role expectations capture not only behavior (e.g. a scientific reviewer is expected to comment on and evaluate papers she was assigned to) but also "role qualities" (e.g. a scientific reviewer should conduct her reviews thoroughly, constructively and objectively).

These views suggest that we should not be confined by the bounds of action-oriented behavior modeling when it comes to role modeling.

3 Roles in the Requirements Phase

We make our standpoint based on the two modeling frameworks that recently became international standards - the i^* and the RM-ODP.

i^* is a popular notation for capturing the rationale and intention in the early-phase requirements engineering. It defines the concepts of actor, agent and role. Being socially-motivated and inspired of multi-agent systems, i^* adopts the notion of autonomy and intentionality when it comes to the representation of actors in systems analysis and design [3]. The authors of this framework argue that, to capture the social aspects in RE, actors and roles should be represented as being intentional and autonomous, as opposed to being programmable and mechanistic[1]. This framework defines strategic rationale business models where we model a role in an RE phase as something that has internal objectives and activities.

RM-ODP, a standardization effort that defines essential concepts for modeling distributed enterprise systems, positions the concept of role in relation with contract, objective and policy of an enterprise specification, which can be thought of as the requirements for an ODP system [5]. In RM-ODP, we can make the coarse-grained organizational representation of an enterprise system using the enterprise concept of community. A community object is supposed to have (component) enterprise objects. These component objects are said to play roles described in the enterprise specification, in order to fulfill objectives of the community in question. The contract specified for a community object states how to assign the component enterprise objects to roles.

The analysis we make on these two standards would suggest that, in RE, roles might better be described in terms of objectives and contracts instead of plain action-oriented behavior. Contract-oriented behavior accounts for role's commitments (or role's qualities by Steimann [13]). Goal-oriented behavior addresses the intentionality and the autonomy of roles. This vision also coincides with those that have been around in the realms of Information Systems and Conceptual Modeling presented in Subsection 2.3. To reason about role's goal, we leverage the well-established field of goal-oriented modeling. To reason about role's contract, we need to consider all processes in which the role in question participates in order to fulfill the contracts it commits to.

[1] Many object-oriented analysis & design methodologies share a vision whereby objects' behavior is defined (by the designer) at design-time. At run-time, objects are instantiated and their operations are invoked on a presumably single thread of control in ways they were designed [14].

3.1 Running Example

Let us consider the business model of a car rental company (as a service). It has roles that are expected to provide the following services: identity check & deposit, cars pickup & return and cars maintenance. We call them `Receptionist` (or rl_{11} for short to be referred to formally later on), `Fleet Manager` (or rl_{12}) and `Mechanics` (or rl_{13}) respectively. Note that the rental car company as a whole also plays a role.

Figure 1 gives an overview of this business model using the i^* notation. In this diagram, i^* actors whose text is in bold represent roles that are considered in this example. Each role have their own goals and contracts. Their goals are represented under the i^* notation of hard goal inside their boundaries. Their

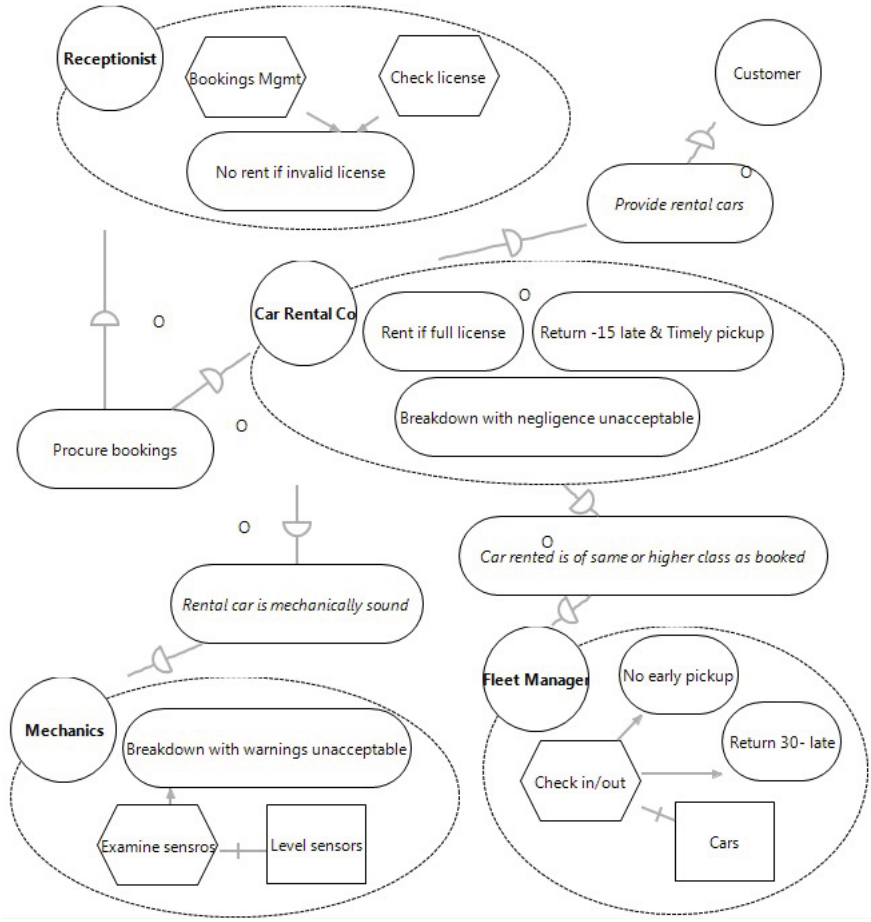

Fig. 1. Representation of roles in the car rental using the i^* notation

Table 1. Informal description of roles participating in the car rental business

Role	Role's Goal	Role's Contract
Role played by the car rental company	To reserve the right not to serve customers who don't have a full driver license; to avoid accepting car return of 15+ minutes late and early pickup; to hold customers responsible for breakdowns caused by their negligence in checking oil/coolant.	The customer expects to be handed in a car of which class was specified in her reservation. If the rental car company fails to do so, they are obliged to provide her with an alternative car of a comparable class. If this option is not available either, the company will offer her some discount on her next rent.
`Receptionist` (rl_1)	To cancel bookings made by customers who don't have a valid driver license.	We assume that `Receptionist` always delivers its functionality as expected. No contractual rules are specified for this role.
`Fleet Manager` (rl_2)	To allow customers to return their rental cars 30- minutes late; to avoid accepting early pickup.	If `Fleet Manager` can't provide the customer with a car of the class she requested in her reservation, an alternative car of a comparable class shall be provided. If this obligation is violated, `Fleet Manager` will be charged an amount of money.
`Mechanics` (rl_3)	To avoid taking responsibility for breakdowns caused by customers who ignored dashboard warnings about the engine oil and coolant during their prolonged renting.	If the car requested by the customer has not been properly serviced, and it is impossible to provide an alternative car of a comparable class, `Mechanics` is obliged to do a quick service to the selected car. Failure to do so will result in `Mechanics` being charged.

contracts are also represented under the i^* goal notation but have italicized text[2]. Each contract features a dependency between a pair of roles that engage in it. Table 1 informally describes these goals and contracts.

To formulate the contracts, let us consider a process that starts when a customer shows up to pick up a rental car she has previously booked and ends when she returns her rented car. At a stage of this process, each of these roles is expected to deliver certain functionality. Figure 2 is a diagram (presented in side view to be fit in a single page) that represents this process using the Business Process Model and Notation (BPMN)[3]. `Receptionist` processes the registration for customers who have booked in advance and checks the customer's driver license. `Fleet Manager` deals with pick-up & return procedures (and checking for any damages the customers might do to their rental cars). `Mechanics` takes care of the rental cars and makes sure that any they are in sound condition before the customers pick them up. In this diagram, each task[4] has a name that starts with a prefix telling which role conducts the task.

`Receptionist`, `Fleet Manager` and `Mechanics` each enter in a contract with the role played by the car rental company. They are expected to deliver the aforementioned functionality. In case they fail to deliver it, they are supposed to deliver alternative functionality as a reparation for their violation.

[2] i^* does not support the modeling of business contracts. We use this visual trick to make contracts and goals look differently in the diagram of Figure 1.

[3] OMG Business Process Management Initiative `http://www.bpmn.org/`

[4] We also annotate each task with functionality drawn under a rectangular callout, which is formally represented and useful for reasoning about contracts in the next section.

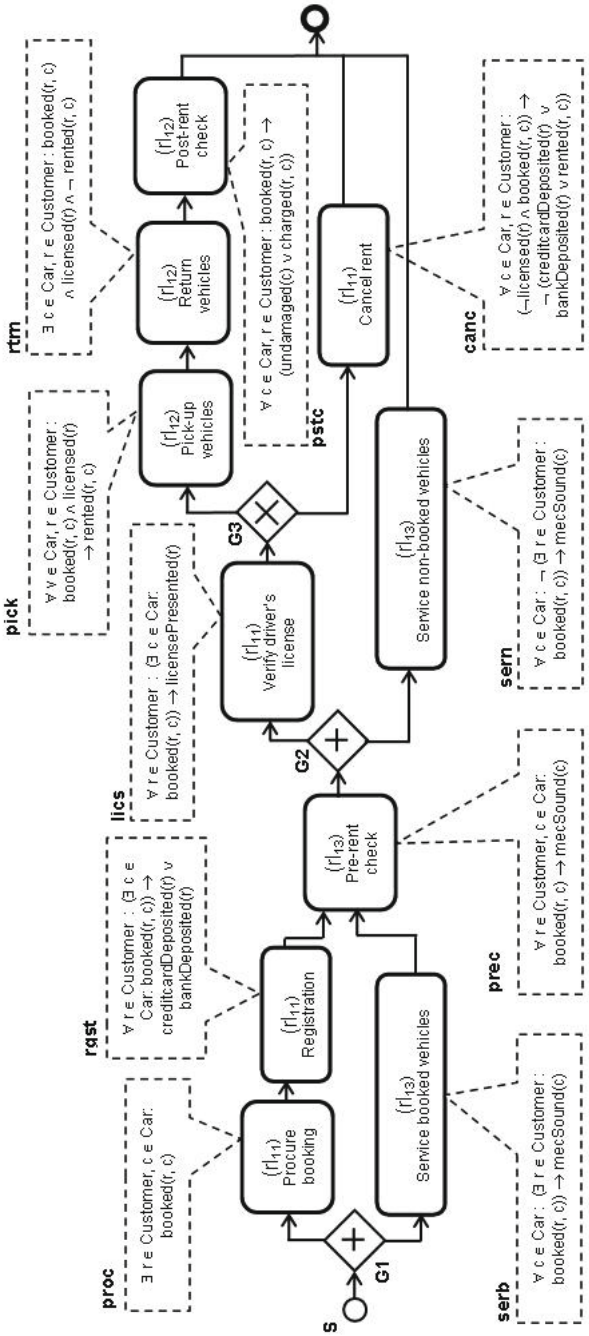

Fig. 2. Roles `Receptionist`, `Fleet Manager` and `Mechanics` in the process of renting a car

4 A Framework for Reasoning about Roles in RE

In this section, we provide a formal machinery for reasoning about the composition and extension of roles, which addresses the research statement presented in the previous section.

4.1 Role Composition

Let $SysBiz$ denote a system or a business that performs a function to deliver value to external or internal actors. Let RL_{sb} be the role played by $SysBiz$ as a whole. This role interacts with roles played by the external actors (hence, external roles). We formally represent RL_{sb} as a tuple $\langle R_{sb}, G_{sb}\rangle$ where R_{sb} denotes a contractual rule that RL_{sb} and the external roles have agreed upon; G_{sb} denotes RL_{sb}'s goal.

In the following, we will use $r_1 \sqsubseteq r_2$ to denote that r_2 subsumes r_1, for contractual rules r_1 and r_2. Let $rl_1, rl_2 \ldots rl_n$ be the component roles that collaborate in order for $SysBiz$ to deliver its function. These roles are assumed by actors that are internal to $SysBiz$, and RL_{sb} can be viewed as a role obtained via the composition of the component roles. Each role rl_i is a tuple $\langle r_i, g_i\rangle$ where r_i denotes a contractual rule agreed upon by rl_i and RL_{sb}; g_i denotes rl_i's goal. Definition 3) formalizes role composition.

Definition 3. *Role RL_{sb} is the composition of roles $rl_1, rl_2 \ldots rl_n$ (and roles $rl_1, rl_2 \ldots rl_n$ constitute role RL_{sb}) if and only if the following hold.*

- *$R_{sb} \sqsubseteq q$ (i.e. rule q subsumes rule R_{sb}) where q is the result of merging $r_1, r_2, \ldots r_n$ according to Definition 1.*
- *RL_{sb}'s goal entails all goals of roles $rl_1, rl_2 \ldots rl_n$. Formally, we have $G_{sb} \models g_1 \wedge g_2 \wedge \ldots \wedge g_n$ and $g_1 \wedge g_2 \wedge \ldots \wedge g_n \not\models \bot$.*

The intuition of Definition 3 is that we relax goals while strengthening rules when we compose roles. Example 1 illustrates this definition. To reason about the contractual behavior of roles, we annotate tasks in Figure 2 with semantic effect written in FOL. Each task has a delivery annotation that is drawn under a rectangular callout. The delivery annotations are written in FOL. We will refer to these annotations by 4-character nick names that appears next to their callout pictograms. The start event and gateways are enumerated as S, G1, G2 and G3 as can be seen in the figure.

Example 1. Let us consider the moment in the rental process (see Figure 2) when a customer is about to pick up a rental car. Roles rl_1, rl_2 and rl_3 are expected to deliver `lics`, `pick` and `prec` respectively. The contractual rule for delivering `pick` of role rl_2 (see Table 1) can be formally expressed as

$\boxed{r_2 : \neg pick \vdash O_2 AltCar \otimes O_2 Charged}$ where *pick* is as depicted in Figure 2 and $AltCar \equiv \forall c \in Car, r \in Customer : booked(r, c) \wedge licensed(r) \wedge \neg rented(r, c) \rightarrow (\exists ac \in Car : comparableClass(ac, c) \wedge mecSound(ac) \wedge rented(r, ac))$.

The formal representation of the contractual rule for role rl_3 is as follows.

$\boxed{r_3 : \neg prec, \neg AltCar \vdash O_2 QuickServ \otimes O_3 Charged}$

Given that role rl_1 does not have a contractual role, merging r_2, r_3 will yield $\boxed{r_{23} : \neg pick, \neg prec \vdash O_2 AltCar \otimes O_3 QuickServ \otimes O_3 Charged}$ according to Definition 1.

Now, let us formally represent the contractual rule for the role played by the rental car company in the following.

$\boxed{r_{co} : \neg delivery \vdash O_{co} AltCar \otimes O_{co} DiscountNextRent}$ where *delivery* denotes the cumulative effect of the rental process at the aforementioned moment.

According to the definition of function *acc* in Subsection 2.1, $\{\neg prec, \neg pick\} \cup \neg QuickServ = \neg delivery$ (see Appendix B for a proof). If the amount of money charged to rl_3 exceeds the mount of discount offered to the customer then according to criterion 3 of Definition 2, r_{23} subsumes r_{co}. To reason about the entailment of goals, we formalize them in FOL. The goals of rl_1, rl_2, rl_3 and the role played the rental car company are formalized as follows.

$g_1 \equiv \forall r \in Customer : \neg hasValidLicense(r) \rightarrow serviceDenied(r)$

$g_2 \equiv \forall c \in Car, r \in Customer : booked(r, c) \rightarrow$
$(timelyPickup(r, c) \wedge returnWithin30(r, c))$

$g_3 \equiv \forall c \in Car, r \in Customer : breakdownWithWarnings(r, c) \rightarrow heldResponsible(r)$

$g_{co} \equiv \forall c \in Car, r \in Customer : (\neg hasFullLicense(r) \rightarrow serviceDenied(r)) \wedge (booked(r, c) \rightarrow (timelyPickup(r, c) \wedge returnWithin15(r, c)) \wedge$
$(breakdownWithoutCheck(r, c) \rightarrow heldResponsible(r))$

Using theorem proving techniques, we can conclude that goal g_{co} entails g_1, g_2 and g_3. A proof for this can be found in Appendix A.

4.2 Role Subtyping

Subtyping is a binary relation on roles. A role is a subtype of another role if the former has weaker goal but a stronger contractual rule than the latter. We call the former the subtyping role. Definition 4 formally captures this point.

Definition 4. *Let $rl_a = \langle r_a, g_a \rangle$ and $rl_b = \langle r_b, g_b \rangle$ be two roles. Role rl_a is a subtype of role rl_b if $r_b \sqsubseteq r_a$ (i.e. rule r_a subsumes rule r_b) and $g_b \models g_a$ (i.e. goal g_b entails goal g_a).*

Example 2. Let us consider an alternative fleet manager (denoted as rl_{2a}) for `Fleet Manager` that would allow customers to return their rental cars within 30 minutes after the end of their rent and accept slightly early pickup (e.g. within 10 minutes before the start of their rent). In case neither the car selected nor an on-premise alternative car could be provided, this fleet manager may offer an alternative car sourced from other car fleets under its management. If this obligation is violated, it will be charged the same amount of money as specified in the contract of the original fleet manager.

We formalize the contract of role $rl_{2a} = \langle r_{2a}, g_{2a} \rangle$ as follows.

$\boxed{r_{2a} : \neg pick \vdash O_{2a} AltCar \otimes O_{2a} AltCarFromNetwork \otimes O_{2a} Charged}$ where *pick* and *AltCar* are the same as in Example 1.

$g_{2a} = \forall c \in Car, r \in Customer : booked(r, c) \rightarrow (timelyPickup(r, c) \vee earlyPickup(r, c)) \wedge returnWithin30(r, c))$

According to criterion 3 of Definition 2, g_{2a} subsumes g_2 (in a similar way to Example 1). In addition, we can deduce that $g_2 \models g_{2a}$ using the distribution property of $\wedge$ over $\vee$. Thus, according to Definition 4, role rl_{2a} is a subtype of role rl_2.

Role subtyping is essential for reasoning about the substitutability of roles in a role composition. The intuition is that a subtyping role can substitute for another role (i.e. the former can safely replace the latter in processes where the latter is expected) if it expresses more commitments but reserves less rights than the other does.

Theorem 1. *Let $R = \{rl_1, rl_2 \dots rl_n\}$ be the component roles that constitute role RL_{sb}. If role rl_s is a subtype of role $rl_k \in R$, then roles $rl_1 \dots rl_{k-1}, rl_s, rl_{k+1} \dots rl_n$ constitute a composite role that is a subtype of RL_{sb}.*

Proof. Let $rl_i = \langle r_i, g_i \rangle, RL_{sb} = \langle r_{sb}, g_{sb} \rangle$. Let p and q be contractual rules such that p is the result of merging $r_1, r_2, \dots r_n$ and q is is the result of merging $r_1 \dots r_{k-1}, r_s, r_{k+1} \dots r_n$ according to Definition 1

According to Definition 4, $r_k \sqsubseteq r_s$ and $g_k \models g_s$. Applying these to what is stated in Definition 3, we have $R_{sb} \sqsubseteq p \sqsubseteq q$ and $G_{sb} \models g_1 \wedge \dots \wedge g_k \wedge \dots \wedge g_n \models g_1 \wedge \dots \wedge g_{k-1} \wedge g_s \dots \wedge g_{k+1} \dots \wedge g_n$. According to Definition 4, $rl_1 \dots rl_{k-1}, rl_s, rl_{k+1} \dots rl_n$ constitute a subtype role of RL_{sb}.

Theorem 1 is useful for checking whether a composite role can substitute for another composite role from the perspective of roles that are external to both. Example 3 illustrates this point.

Example 3. Let us consider another rental company that deals with an alternative fleet manager (rl_{2a}) discussed in Example 2. It also deals with the very receptionist and mechanics described in our running example (Subsection 3.1). As such, this rental company is able to offer early pickup (10 minutes maximum), late return (20 minutes maximum) and a wide range of alternative rental cars. It is a subtype role of the one described in the running example according to Definition 4. In other words, we have $\langle rl_1, rl_{2a}, rl_3 \rangle$ is a subtype of $\langle rl_1, rl_2, rl_3 \rangle$ because rl_{2a} is a subtype of rl_2.

5 Related Work

As discussed in Section 3, the RM-ODP standard defines the enterprise specification, which could be regarded as requirements for the system to be built. Roles

are specified in this requirements specification. To reach international consensus for becoming a standard, RM-ODP is sometimes exceedingly generic and avoids providing patterns. Definitions of role in this standard range from an identifier of behavior, a subset of the total behavior (of an object) to an abstraction of the behavior that belongs to collaborative behavior [15]. In our work, we come up with a definition of roles that is separated from concrete components, agents, entities, etc. who may come into existence in the subsequent phases of development. This is in line with existing work on the semantics of roles and role-related methodological issues [16,10,14]. However, we do not relate the subtyping hierarchy of roles to that of entities like Steimann's work [10].

With respect to relationships between roles, our work shares the relaxing/strengthening principle with the notion of behavioral subtyping [17] whereby preconditions are eased and postconditions are strengthened in a subtype. Our notion of role substitutability in Theorem 1 is nevertheless not in line with the concept of object aggregation in contemporary object-oriented modeling where substitutability between component objects does not imply substitutability between composite objects.

Regarding the i^* standard, we provide an alternative conceptualization of roles in RE by introducing the concept of contractual rules [2] to the representation of roles. A role not only has goals (i.e. rights) but also features contracts (i.e. responsibility). We explicitly represent the responsibilities and rights of roles who are considered intentional and autonomous in i^*.

6 Conclusion

Are business contracts plus goals equal to roles in RE? In this paper, we propose a formal approach for modeling intentional and autonomous roles in the realm of RE. We base our work on a unified view in role modeling that suggests we explicitly represent both the responsibilities and the rights of a role [1]. We argue that the role's responsibility and rights can be captured by contractual rules and goals, respectively. We leverage existing work on the formalization of business contracts [2] and the formulation of hard goals in the i^* modeling framework [3] in order to devise techniques for reasoning about the composition and the substitutability of roles. Technically, goals are relaxed by means of entailment whereas contracts are strengthened via subsumption in a subtype role or a role composition.

Discussions. The contribution of our work is twofold. First, we propose to conceptually model RE roles in terms of goals and contracts. Second, we deal with the composition and substitutability of roles by leveraging existing work on business contracts and goal-oriented behavior. However, our framework may not cope with scenarios where role's contractual rules cannot effectively be merged and/or

entailment between role's goals cannot be formulated. We have not aligned our work to contemporary enterprise modeling frameworks such as ArchiMate[5].

Future Investigations. Further work includes investigating the non-functional properties of roles in RE, which can be done by leveraging the i^* concepts of soft goal and belief while making logic-based contractual rules more quantifiable. Another direction of future work is to detect conflicts between constituent roles when composing them (e.g. their goals contradict with one another). Yet another direction for further work would be feeding our role model to a design phase. The research question here is on the semantics and methodological issues of designing entities (or agents) so that they can play roles specified in the requirements phase.

References

1. Zhu, H., Zhou, M.: Roles in Information Systems: A Survey. IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews 38(3), 377–396 (2008)
2. Governatori, G., Milosevic, Z.: A Formal Analysis of a Business Contract Language. International Journal of Cooperative Information Systems 15(4), 659–685 (2006)
3. Yu, E., Giorgini, P., Maiden, N., Mylopoulos, J.: Social Modeling for Requirements Engineering. The MIT Press (January 2011)
4. Liu, L., Yu, E., Mylopoulos, J.: Security and Privacy Requirements Analysis within a Social Setting. In: Proceedings of 11th IEEE International Requirements Engineering Conference, pp. 151–161 (September 2003)
5. ISO/IEC: ITU-T X.903 | ISO/IEC 10746-3 Information Technology - Open Distributed Processing - Reference Model - Architecture. International Standard, SC 7 and ITU (2010)
6. Hinge, K., Ghose, A., Koliadis, G.: Process SEER: a Tool for Semantic Effect Annotation of Business Process Models. In: Proceedings of the 13th IEEE International Conference on Enterprise Distributed Object Computing, pp. 49–58. IEEE Computer Society, Auckland (2009)
7. Raut, M., Singh, A.: Prime Implicates of First Order Formulas. International Journal of Computer Science and Applications 1(1), 1–11 (2004)
8. Linington, P., Milosevic, Z., Cole, J., Gibson, S., Kulkarni, S., Neal, S.: A Unified Behavioural Model and a Contract Language for Extended Enterprise. Data & Knowledge Engineering 51(1), 5–29 (2004)
9. Gabbay, D.M., Woods, J.: Logic and the Modalities in the Twentieth Century. Handbook of the History of Logic, vol. 7. North-Holland (July 2006)
10. Steimann, F.: On the Representation of Roles in Object-Oriented and Conceptual Modelling. Journal of Data & Knowledge Engineering 35(1), 83–106 (2000)
11. Steimann, F.: Role = Interface: A merger of concepts. Journal of Object Oriented Programming 14(4), 23–32 (2001)

[5] Archimate Homepage `http://www.archimate.nl/`

12. Bachman, C.W.: The Role Data Model Approach to Data Structures. In: Proceedings of International Conference on Databases, pp. 1–18. University of Aberdeen: Heyden & Son (1980)
13. Steimann, F.: The Role Data Model Revisited. Applied Ontology Journal 2(2), 89–103 (2007)
14. Zambonelli, F., Jennings, N.R., Wooldridge, M.: Developing Multiagent Systems: the Gaia Methodology. ACM Transaction on Software Engineering Methodology 12, 317–370 (2003)
15. Genilloud, G., Wegmann, A.: A Foundation for the Concept of Role in Object Modelling. In: Proceedings of 4th International Enterprise Distributed Object Computing Conference, pp. 76–85 (September 2000)
16. Guarino, N.: Concepts, Attributes and Arbitrary Relations: Some Linguistic and Ontological Criteria for Structuring Knowledge Bases. Journal of Data & Knowledge Engineering 8(3), 249–261 (1992)
17. Liskov, B.H., Wing, J.M.: A Behavioral Notion of Subtyping. ACM Transactions on Programming Languages and Systems 16(6), 1811–1841 (1994)

A Goals Entailment

We have $hasFullLicense(r) \equiv hasValidLicense(r) \wedge probationPassed(r)$ (i.e. full driver's license is a valid license that has successfully undergone a probation), $breakdownWithoutCheck(r,c) \equiv breakdownWithWarnings(r,c) \vee levelsDropped(c)$ (i.e. breakdown was caused by levels dropped significantly, of which the customer might be warned if the rental car was equipped with sensors) and $returnWithin15(r,c) \equiv returnWithin30(r,c) \wedge within15$. To make the proof easier to follow, we denote predicates that appear in the formalization of these goals as follows. $hasFullLicense(r) \equiv F$, $hasValidLicense(r) \equiv V$, $probationPassed(r) \equiv P$, $breakdownWithoutCheck(r,c) \equiv C$, $breakdownWithWarnings(r,c) \equiv W$, $serviceDenied(r) \equiv D$, $timelyPickup(r,c) \equiv T$, $booked(r,c) \equiv B$, $returnWithin30(r,c) \equiv R$, $returnWithin15(r,c) \equiv Q$, $heldResponsible(r) \equiv H$, $within15 \equiv X$, $levelsDropped(c) \equiv L$.

We can safely add quantifier $\forall c \in Car$ to g_1 because g_1 does not matter on rental cars. This is to unify goals g_1, g_2, g_3 and g_{co} in terms of quantifiers. Syntactically, they now all start with quantifiers $\forall c \in Car, r \in Customer$. For the sake of simplicity, we ignore these quantifiers in the proof. Given the aforementioned denotation, we have $g_{co} \equiv (\neg F \rightarrow D) \wedge (B \rightarrow (T \wedge Q) \wedge (C \rightarrow H)$, $g_1 \equiv \neg V \rightarrow D$, $g_2 \equiv B \rightarrow (T \wedge R)$ and $g_3 \equiv W \rightarrow H$.

Note that $\neg F \rightarrow D$ equals to $F \vee \neg Dequiv(V \wedge P) \vee \neg Dequiv(V \vee \neg D) \wedge (P \vee \neg D) equiv(\neg V \rightarrow D) \wedge (\neg P \rightarrow D)$, which obviously entails g_1. In addition, since $Q \equiv R \wedge X$, we have $B \rightarrow (T \wedge Q) \equiv \neg B \vee (T \wedge R \wedge X) \equiv (\neg B \vee (T \wedge R)) \wedge (\neg B \vee X) \equiv (B \rightarrow (T \wedge R) \wedge (B \rightarrow X)$, which entails g_2. Finally, $C \rightarrow H = \neg C \vee H = \neg(W \vee L) \vee Hequiv(\neg W \wedge \neg L) \vee H \equiv (\neg W \vee H) \wedge (\neg L \vee H) \equiv (W \rightarrow H) \wedge (L \rightarrow H)$, which entails g_3. Thus $g_{co} \models g_1 \wedge g_2 \wedge g_3$.

B Accumulating Effects

Negating the semantic annotations given in Figure 2 will yield $\{\neg prec, \neg pick\} \equiv \{\exists r \in Customer, c \in Car : booked(r, c) \wedge \neg mecSound(c); \exists r \in Customer, c \in Car : booked(r, c) \wedge licensed(r) \wedge \neg rented(r, c)\}$. In addition, we have $QuickServ \equiv \forall r \in Customer, c \in Car : booked(r, c) \wedge \neg mecSound(c) \rightarrow serviced(c)$ (i.e. all cars, booked by the customers, that are not mechanically sound shall be serviced quickly). Thus, $\neg QuickServ \equiv \exists r \in Customer, c \in Car : booked(r, c) \wedge \neg mecSound(c) \wedge \neg serviced(c)$.

The scenario label for the three constituent roles at the moment when task `Pick-up vehicles` has been executed is $\langle S, G1, \{\langle proc, rgst, prec\rangle, servb\}, prec, G2, lics, G3, pick\rangle$. We proceed in cumulating semantic annotations along this label scenario (see Subsection 2.1). Since *proc*, *rgst*, *serb*, *prec*, *lics* and *pick* do not contain contradictory clauses (actually *serb* and *prec* have the same clause), we can simply collect their clauses. Negating these clauses would yield $\exists r \in Customer, c \in Car : booked(r, c) \wedge \neg mecSound(c)$; $\exists r \in Customer, c \in Car : booked(r, c) \wedge \neg mecSound(c)$; $\exists r \in Customer, c \in Car : booked(r, c) \wedge licensed(r) \wedge \neg rented(r, c)$.

So, we have $\{\neg prec, \neg pick\} \cup \neg QuickServ = \neg delivery$.