An Analyzable Model of Automated Service Negotiation

Aida Čaušević, Cristina Seceleanu, Paul Pettersson
Mälardalen Real-Time Research Centre (MRTC),
Mälardalen University, Västerås, Sweden
Email:[aida.causevic,cristina.seceleanu,paul.pettersson]@mdh.se

***Abstract*—Negotiation is a key aspect of Service-Oriented Systems, which is rarely supported by formal models and tools for analysis. Often, service negotiation proceeds with timing, cost and resource constraints, under which the users and providers exchange information on their respective goals, until reaching a consensus. Consequently, a mathematically driven technique to analyze various ways to achieve such goals is beneficial. In this paper, we propose an analyzable negotiation model between service clients and providers, in our recently introduced language REMES and its corresponding textual service composition language HDCL. The model can be viewed as a negotiation interface for different negotiation strategies and protocols, which iterates until an agreement is reached. We show how to analyze the negotiation model against timing, cost and utility constraints, by transforming it into the Timed Automata formal framework. We illustrate our approach through an insurance scenario assuming a form of the Contract Net Protocol for web services.**

I. INTRODUCTION

Recently, service orientation has become a standard paradigm that accommodates building distributed applications from autonomous entities called services, independent of any specific implementation platform. As opposed to the traditional approach of software system construction, in Service-Oriented Systems (SOS)[1] the same service may be offered at various prices, Quality-of-Service (QoS), and other conditions, depending on the customer needs. In such a setting, the interaction between involved parties might involve the *negotiation* of what is possible at request time, aiming at meeting needs dynamically.

Within SOS, services can have one of the following roles: client, provider, or mediator. The role of a client service is to require a service that performs a specific task within given resource limits. The mediator initiates and steers the communication, that is, the negotiation process between the client and the provider, helping them to reach an agreement. The service provider creates an offer, based on the client's request and on available services. The negotiation process is an iterative process that, if successful, ends up in a Service Level Agreement (SLA), that is, a contract between the client and the provider. The contract sets boundaries on both the functional and extra-functional properties of a service, which are to be guaranteed, defines the cost of a service delivery, and possible penalties in case the contract is broken. For instance, a SLA between a car insurance company and its clients might include the cost of a service provision, the guaranteed time for repairing some car damage, and availability of the offered repair shop at the specific time.

With the growing number of services that are offered by different service providers, the need to formally define and analyze the service negotiation process has increased. The offered services may have the same functionality, but might differ in extra-functional qualities, such as response time, price, time-to-serve, or reputation. Given that service provision is being negotiated upon the client's demand, establishing guarantees of provided QoS, under the changing conditions of service composition and possible negotiation constraints, becomes a challenging task.

Taking into account the above, a mathematically driven technique to analyze various ways to achieve the client's and provider's goals is beneficial. For instance, one can compute the minimum price for reaching an agreement within a given time constraint, while maximizing the utility function for all involved parties. Such analysis might expose trade-off configurations that can provide valuable inputs to both service designers as well as service users. Assuming particular offer and counter-offer strategies of the involved services, as well as a negotiation protocol, the formalization of the negotiation between clients and providers basically results in a "negotiation interface" that iterates until an agreement is reached.

In this paper, we propose an analyzable negotiation model between service clients and providers, based on an iterative form of the Contract Net Protocol (CNP) for web services, which we describe in Section III. The service model is time- and price-aware, being described in REMES [1], a behavioral language intended for modeling and analysis of interacting embedded components and services, briefly recalled in Section II-A. The negotiation model is obtained by composing REMES services, within a corresponding textual service composition language called Hierarchical Dynamic Composition Language (HDCL), via operators that have been defined formally in our previous work [2]. The salient point of the approach proposed in this paper is the fact that the obtained negotiation model can be analyzed against safety, timing, and utility constraints, for all possible behaviors. This can be achieved after transforming the negotiation model into the

[1]In this paper we write SOS meaning only service-oriented software, more precisely services and their compositions.

Timed Automata (TA) formal framework (Section II-B), which has a precise underlying semantics. One possible analysis goal could be to compute a path in which an agreement is reached within some prescribed time, with maximized utility.

In the related literature, there already exist some SOS approaches that enable service negotiation [3], [4], and also propose ways to support its formal analysis. However, these approaches show limited support in building the formal system model, out of formalized services. In comparison, our REMES service models can be deductively reasoned about, or can be (automatically) translated to TA [5], and analyzed with UPPAAL tools, for functional and extra-functional behaviors (timing and resource-wise behaviors) [6].

Our approach for service negotiation, described in Section III, is illustrated by a car insurance scenario, in Section IV. The analysis of the described service negotiation model for the insurance scenario is presented in IV-C. Last but not least, we compare our approach with some relevant work in Section V, before concluding the paper in Section VI.

II. PRELIMINARIES

A. REMES HDCL *modeling language*

To address functional but also extra-functional behavior such as timing and resource usage of SOS, in this paper, we use the dense-time hierarchical modeling language called REMES [1]. The language has been initially intended as a meaningful basis for modeling and analysis of embedded systems in a component-based fashion. To make it suitable for modeling SOS too, we have recently extended REMES with constructs fit for a SOS description [2]. To enable formal analysis, REMES models can be semantically transformed into Timed Automata (TA) [5], [7].

REMES language is well-suited for abstract modeling, it is hierarchical, has an input/ouput distinction, a well-defined formal semantics, and tool support for both component-based [2] and service-oriented [3] system modeling and analysis [6], [8].

The REMES service contains a list of attributes (i.e., service type (a web, network, or embedded service), capacity (the maximum ability to serve a given number of messages per time unit), time-to-serve (the worst-case time needed to respond and serve a given request), status (the current service status, i.e., passive, idle, active), service precondition, and postcondition exposed at the interface of the REMES service. A service precondition is a predicate that constrains the start of the service's execution, and must be true at the time a REMES service is invoked. A postcondition is the output guarantee and must hold at the end of a REMES service execution for a service to be correct.

In order to manipulate services REMES supports service creation, deletion, composition, and replacement via REMES interface operations. One can also think about creating, deleting, reordering a list of services (s_list) that can be useful when services are being composed.

[2]The REMES tool-chain is available at http://www.fer.hr/dices/remes-id.

[3]More information available at http://www.idt.mdh.se/personal/eep/reseide/.

REMES is accompanied by a hierarchical dynamic composition language (HDCL) that facilitates modeling of nested sequential, parallel or synchronized services and their compositions.

$$\mathsf{DCL} \ ::= \ (\mathsf{s\_list}, \mathsf{PROTOCOL}, \mathsf{REQ})$$

$$\mathsf{HDCL} \ ::= \ (((\mathsf{DCL}^{+}, \mathsf{PROTOCOL}, \mathsf{REQ})^{+}, \mathsf{PROTOCOL}, \mathsf{REQ})^{+} \ldots)$$

In the equations given above DCL (Dynamic Composition Languages) stands for the basic service composition, while HDCL describes the hierarchical composition. The positive closure operator is used to express that one or more DCLs are needed to form an HDCL. The PROTOCOL defines the type of binding between services and here we will consider only binary operators as follows:

$$\mathsf{PROTOCOL} \ ::= \ service_m \ \mathsf{binary\_operator} \ service_n$$

The requirement REQ is a predicate ($\Gamma \rightarrow$ Bool, where Γ is the set of states) that can include both functional and extra-functional properties/constraints of the composition. It identifies the required attribute constraints, capability, characteristics, or quality of a system, such that it exhibits the value and utility requested by the user. HDCL allows creating new services by composing existing services via binary operators, as well as adding and/or deleting services from lists. The above binary operator is defined as follows:

$$\mathsf{Binary\_operator} \ ::= \ ; \ | \ \| \ | \ \|_{SYNC\text{-}and} \ | \ \|_{SYNC\text{-}or}$$

In the above ; denotes the sequential composition of services, whereas $\|$, $\|_{SYNC\text{-}and}$, and $\|_{SYNC\text{-}or}$ denote their parallel composition. To model synchronized behavior in REMES we use a special kind of mode, called AND/OR mode. By the semantics of the mode, in an AND or an OR mode, the services finish their execution simultaneously, from an external observer's point of view. However, if the mode is employed as an AND mode, the subservices are entered at the same time, and their incoming edges do not contain any guard (a boolean enabling condition), while an OR mode assumes that one or all subservices are entered based on the guards annotated on the incoming edges. Services that belong to this type of REMES mode and that have to synchronize their behavior in the end of their execution communicate via $\|_{SYNC\text{-}or}$, or $\|_{SYNC\text{-}or}$ operators, respectively.

To verify the service correctness, we use the forward analysis technique based on the computation of the strongest postcondition of a REMES service w.r.t. a given precondition [9]. To prove the correctness of a REMES service in isolation, we check that the calculated strongest postcondition is no more than the given requirement. Assume that $\{p\}\mathcal{S}\{q\}$ is a Hoare triple denoting the partial correctness of service $\mathcal{S}$ with respect to precondition p and postcondition q. According to Dijkstra and Sholten [9], the strongest postcondition transformer, denoted by $(sp.\mathcal{S}.p)$, is the set of final states for which there exists a computation controlled by $\mathcal{S}$, which belongs to class "initially p". Assuming that p holds, the execution of a service $\mathcal{S}$ results in $sp.\mathcal{S}.p$ true, if $\mathcal{S}$ terminates. Proving the Hoare triple, that is, proving the correctness of service S, reduces to showing that $(sp.\mathcal{S}.p \Rightarrow q)$ holds. For analysis purposes, REMES models can be transformed to TA via a well-defined set of rules [6], [7], [10]. For a more thorough description of the REMES language, we refer the reader to our previous work [1], [2].

B. Timed Automata

A Timed Automaton (TAn) [5], [11] is a finite-state machine enriched with a set of clocks. All clocks are synchronized and they are assumed to be real-valued functions of time elapsed between events.

Formally, let us assume a finite set of real valued variables C ranging over x, y, etc., standing for clocks, and a finite alphabet Σ ranging over a, b, etc., standing for actions. A clock constraint is a conjunctive formula of atomic constraints of the form $x \sim n$ or $x - y \sim n$ for $x, y \in C, \sim \in \{<, \leq, =, \geq, >\}$ and $n \in N$. Clock constraints are used as guards (g) for timed automata. $\chi(C)$ is used to denote the set of clock constraints, ranged over by guards g.

Definition 1: A Timed Automaton A is a tuple (L, l_0, E, I) where: L is a finite set of locations, l_0 is the initial location, $E \subseteq L \times \chi(C) \times \Sigma \times L$ is the set of edges, $I : L \rightarrow \chi(C)$ assigns invariants to locations. In the case of $(l, g, a, r, l') \in E$, we write $l \overset{g,a,r}{\rightarrow} l'$, where g is a guard, a boolean condition that must hold in order for the edge to be taken, a is an action, and r is a simple clock reset. ■

Fig. 1 depicts an example of a Timed Automaton (TAn). The TAn models a client in the car insurance scenario, described in details in Section IV, which can report three types of the car damage to the insurance company. The TAn description consists of four locations: Start, DamRepSent0, DamRepSent1, and DamRepSent2 (with Start as the initial location), and edges, which are directed lines connecting locations. The timing behavior is controlled by the clock variable t. For each location, it is possible to assign an invariant that must hold in order to stay in that location (e.g., invariant in location DamRepSent0 is (t ≤ 20)), which also enforces a location change in case it ceases to hold. Further, each edge may be decorated with guards, that is, boolean expressions that must hold in order for the respective edge to be taken (e.g., edge from location DamRepSent1 to Start contains the guard RS01 == true).

The semantics of TA is defined in terms of a labeled transition system. A state of TA depends on its current location and on the current values of its clocks. So, a state of a TA is a pair (l, u), where l is a location, and $u : C \rightarrow R_+$ is a clock valuation. The initial state (l_0, u_0) is the starting state where all clocks are zero. There are two kinds of transitions: delay transitions and discrete transitions.

*Delay transition*s are the result of time passage and do not cause a change of location. More formally, we have

$$(l, u) \overset{d}{\rightarrow} (l, u \oplus d)$$

if $u \oplus d' \models I(l)$ for $0 \leq d' \leq d$. The assignment $u \oplus d$ is the result obtained by incrementing all clocks of the automata with the delay d.

Discrete transitions are the result of following an enabled edge in a TAn. Consequently, the destination location is changed from the source location to the new target location, and clocks may be reset. More formally, a discrete transition

$$(l, u) \overset{a}{\rightarrow} (l', u')$$

corresponds to taking an edge $l \overset{g,a,r}{\rightarrow} l'$ for which the guard g is satisfied by u. The clock valuation u' of the target state is obtained by modifying u according to updates r such that $u' \models I(l')$.

Reachability analysis is one of the most useful analyses to perform on a given timed automaton. The reachability problem can be defined as follows: Given two states of the system, is there an execution starting at one of them that reaches the other? The reachability analysis can be used to check that an error state is never reached, or just to check the sanity of the model.

Definition 2: We write $(l, u) \rightarrow (l', u')$ if $(l, u) \overset{\sigma}{\rightarrow} (l', u')$ for some $\sigma \in \Sigma \cup R_+$. For an automaton with initial state (l_0, u_0), (l, u) is reachable iff $(l_0, u_0) \rightarrow^* (l, u)$. More generally, given a constraint $\phi \in \chi(C)$ we say that the state (l, ϕ) is reachable if (l, u) is reachable for some u satisfying ϕ. ■

A network of TA, $A_1 \| ... \| A_n$, over χ and Σ, is defined as the parallel composition $A_1 \| ... \| A_n$ over χ and Σ. Semantically, a network again describes a timed transition system obtained from the components, by requiring synchrony on delay transitions, and requiring discrete transitions to synchronize on complementary actions (i.e., $a?$ (receive synchronization) is complementary to $a!$ (send synchronization)).

Properties of TA can be specified in the Timed Computation Tree Logic (TCTL), which is an extension of Computation Tree Logic (CTL) with clocks. CTL is a specification language for finite states systems. Using CTL one can reason about sequence of events. Let AP be a set of atomic propositions, $a \in AP$. In this paper, a CTL formula ϕ is defined as follows:

$$\phi \quad ::= \quad \bot \mid \top \mid a \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \rightarrow \phi_2 \mid EF\phi \mid AF\phi \mid AG\phi \tag{1}$$

Each CTL well-defined formula is a pair of symbols. The first operator is either A ("for All paths"), or E ("there Exists a path"). The latter operator is one of the following: F ("in a Future state"), or G ("Globally in the future"). For example $EF\phi$ means that there exists a path such that ϕ is eventually satisfied and it is called a reachability property. More details on CTL and TCTL can be found in earlier work of Alur et al. [12], [13]. In order to specify properties of Priced Timed Automata (PTA), the Weighted CTL (WCTL) logic has been introduced [14]. WCTL extends TCTL with resets and testing of cost variables.

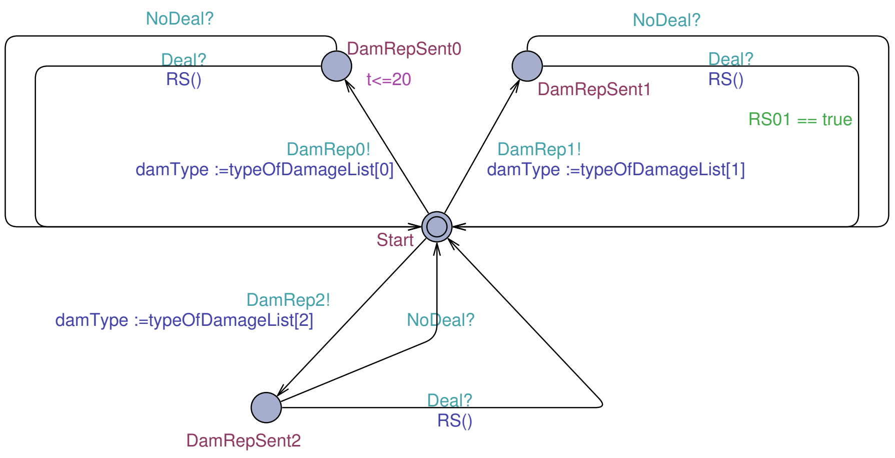

Fig. 1. The TAn model of a REMES service

III. OUR SERVICE NEGOTIATION MODEL

Pruitt describes a negotiation process as a process by which a joint decision is made by two or more parties [15]. As a prerequisite to the negotiation, the parties first verbalize contradictory demands and then move towards agreement, via a process of concession, making, or search for a new alternative. Within the negotiation process in SOS, one might think of a number of both quantitative (e.g., price of the service, the penalty for contract violation, etc.) and qualitative issues (e.g., a final goal). Each successful negotiation process includes a set of such issues to be resolved w.r.t. demands and offers provided by involved parties. It might be the case that the involved parties have to make trade-offs between issues under negotiation in order to reach an agreement. In most cases, time to reach an agreement is a crucial factor, meaning that the process of negotiation is time bounded from above. One might think about several levels of timing constraints:(i) the time it takes to reach an agreement; (ii) the time by which a negotiated service has to be executed (in cases when multiple services have to be composed and coordinated); (iii) the timing constraints on a negotiated service given by the service client (a deadline that a service has to fulfill). In case that both a service client and a service provider have agreed on an offer, they bound to a SLA, i.e., a contract among the involved parties that includes a set of relations to be evaluated over a set of attributes, where each attribute has an assigned value to be satisfied. Attributes in SLA usually have the same priority, but it might be the case that some attribute gets a higher priority, meaning that some value is added to the relation.

Next we will introduce our negotiation model that could be seen as automatically iterating over client-provider choices until a consensus is reached.

A. *Modeling Service Negotiation in* REMES HDCL

To provide a systematic and analyzable way to model a negotiation process, in this paper we propose the REMES HDCL negotiation model, described in the following. The model is based on the set of REMES interface operations and the hierarchical language HDCL, which supports REMES service composition, which we have recently proposed [2]. The model's benefit is that once the negotiation process is finished and a service has been provided, one can easily check if the service really delivers the original qualities. Also the model can be seen as a "negotiation service" that iterates over client-provider choices, automatically, until a consensus is reached. The verification is rigorous, and it requires the computation of the service's strongest postcondition (sp), which we can provide automatically [16], and then checking if the strongest postcondition implies the client requirements:

$$sp.Service.s\_pre \quad \Rightarrow \quad s\_post$$

In the above implication, (s_pre) is a service precondition required by the service client, and (s_post) is the postcondition of a service offered by the service provider that should be guaranteed after the service is executed. If the implication is verified, one can ensure that the provided service fits with the client requirement, before the service has been executed. Additionally, the fact that the REMES model can be translated to TA and analyzed with the UPPAAL [4] model checker brings more insight into the negotiation process and its possible outcomes.

Although our negotiation model is general, here we assume an iterative form of the Contract Net Protocol (CNP) between web services, described in REMES HDCL. In the CNP the roles of the manager that can be seen as a negotiation mediator, and the contractor (a service provider) are assumed. The manager gets a request from a client and aims at finding an appropriate contractor to fulfill the request via call for proposals (CFP). Further on, CFP is evaluated by contractors and a response is being sent back. The manager evaluates the received proposals and based on the evaluation decides which proposals to accept and which to reject. In the iterative CNP, the communication between manager and possible contractors repeats several times until the consensus is reached. In each round the

[4]More information available at http://www.uppaal.org.

contractors aim to improve on their previous proposals, in order to make them more suitable to the manager.

Let us assume that we want to model a service negotiation process using HDCL. The steps to follow are enumerated bellow.

(i) First, all service types have to be declared and instantiated using REMES interface operations.

```
**declare** $Service ::=<$
        $service\ type : \{web\ service\},$
        $capacity : N,$
        $time\_to\_serve : N,$
        $status : \{passive, idle, active\},$
        $precondition : predicate,$
        $postcondition : predicate >$
$\forall i \in N$ **create** $Manager_i : Service$
$\forall i \in N$ **create** $Client_i : Service$
$\forall i \in N$ **create** $Contractor_i : Service$
```

As shown above, the type Service is declared first, describing the attributes of the service. Next, we create three actual instances (Manager, Client, and Contractor) of the declared data type.

(ii) Second, one needs to add the services to appropriate lists to model the composition of services. For this, a type List is declared, initialized with two lists, list_request and list_offer, which are then populated by the actual services.

```
**declare** $List ::=< [service\_name_0 : Service, \ldots,$
                $service\_name_n : Service] >$
    **create** $list\_request : List$
    **create** $list\_offer : List$
    **add** $Client\ list\_request$
    **add** $Manager\ list\_request$
    **add** $Contractor\ list\_offer$
    **add** $Manager\ list\_offer$
```

In HDCL the service negotiation can be modeled as a service composition via the parallel with synchronization protocol modeled by the operator $\|_{SYNC\text{-}and}$. Services communicating via $\|_{SYNC\text{-}and}$ operator belong to the special type of REMES mode, called AND mode. Respecting the semantics of an AND mode, the services connected by this operator start and finish their execution simultaneously.

(iii) Last but not least, our model of service negotiation is defined by the following:

```
$DCL\_req ::= (list\_request, \|_{SYNC\text{-}and}, req_{client})$
$DCL\_offer ::= (list\_offer, \|_{SYNC\text{-}and}, req_{provider})$
**DO**
    $p\_offer := negotiation(param_p)$
    $c\_request := negotiation(param_c)$
    $neg\_cost\_client := a * t$
**OD** $((t \leq t_{max}) \wedge (c\_request < p\_offer))$
**if** $(c\_request \geq p\_offer)$ **then**
  $contract := true$
  $neg\_cost\_client := b * t$
**else**
  $contract := false$
  $neg\_cost\_client := neg\_cost\_client + penalty$
  $neg\_cost\_provider := penalty$
**fi**
```

The requirements req_{client} (a requirement defined from the client's side) and $req_{provider}$ (a provider defined requirement) are predicates that might include both functional and extra-functional properties of service compositions as part of the negotiation process. The content of the requirement might include different negotiable parameters (denoted by $param_p$ for the provider, and $param_c$ for the client), such as price, time, or location at which a provided service should be available. The provider's offer is calculated via function $negotiation(param_p)$ and stored in variable p_offer similar to the approach presented by Kumpel et al. [17]. In case that the provided offer has not met the client's expectation, so that the client is willing to continue the negotiation (given that the timing constraints permit this), his/her request ($c\_request$) can be changed/updated using the same function (as the one used in the provider's case) but with a different parameter $negotiation(param_c)$.

To describe the function let us assume n negotiable parameters $p_1, \ldots, p_n$, and for each parameter a set of acceptable values predefined as $v_{p_i} = \{v_1, \ldots, v_k\}$. The set of acceptable values can contain real, integer, or natural number values depending of the parameter type. In case of a real-valued range, the value set consists of two values only, the minimum and the maximum, respectively. In order to provide an offer, we calculate the utility function $U(O)$ as follows:

$$U(O) \quad = \quad \sum_{i=1}^{n} U(v_{p_i}) = \sum_{i=1}^{n} (w(p_i) * pref(v_{p_i})) \tag{2}$$

where $w(p_i)$ is the weight of v_{p_i} compared to other involved parameters, $pref(v_{p_i})$ represents the degree of importance of value v_{p_i} as compared to all values of parameter p_i, and the set of acceptable values per parameter $O = \{v_{p_1}, \ldots, v_{p_n}\}$. To provide the most suitable value of parameter p_i such that the utility function is maximized for all participants in the negotiation, one has to define an objective function Z [17]. Assuming two parties in the negotiation, the objective function Z has the following form:

$$\begin{aligned} Z(v_{p_i}) &= U_1(v_{p_i}) + U_2(v_{p_i}) - \mid U_1(v_{p_i}) - U_2(v_{p_i}) \mid \\ &= 2 * min(U_1(v_{p_i}), U_2(v_{p_i})) \end{aligned} \tag{3}$$

The optimal value to be offered for parameter p_i is the one for which the objective function Z is maximum. In case that more than one value is found, one can choose the one for which the sum $U_1(v_{p_i})$ + $U_2(v_{p_i})$ is the highest. The value calculated in this way is offered to the client.

Since we are interested in time-constrained negotiation models, we have introduced in this model t_{max} that defines the maximum allowed time for the negotiation, present in the constraint $t \leq t_{max}$, where t is an implicit time variable. Once t_{max} is reached, the negotiation process finishes.

The negotiation process might finish with a signed contract ($contract := true$) if the Boolean condition ($c\_request \geq p\_offer$) is satisfied, or without any agreement ($contract := false$) if the given condition does not hold ($c\_request < p\_offer$). In our model, we assume that each participant in the negotiation pays a price, captured by two variables, neg_cost_client, and $neg\_cost\_provider$, respectively. The negotiation cost payed by the client is a linear function of time, with constant rates, i.e., a, b, respectively. In case that the negotiation ends without any agreement both the client and the provider pay a penalty price that is a constant modeled by $penalty$.

In our REMES HDCL negotiation model, one might consider different types of costs, as part of the negotiation model. For example, one might think about the cost of service provision, which is negotiated between the service client and the service provider. Another interest might be in the cost of the negotiation process for a client (neg_cost_client), which increases linearly with the elapsed time, provided that the negotiation process has more than one iteration. In case that the negotiation process finishes without any agreement within the required time (i.e., time for negotiation elapsed) the cost of negotiation is increased by a given penalty (i.e., constant $penalty$ in the model) paid by both involved parties.

The negotiation process aims at bringing benefits to all participants. The client benefits in terms of being offered a service that fits the provided requirements, while the service provider benefits in terms of money to be paid for a service, and reputation based on the negotiation flow (i.e., it might be the case that several service providers can offer the same service, then the reputation of a service provider plays a significant role when deciding which service provider to choose to deliver the service).

In the following section, we show what types of analysis are supported by our REMES HDCL model.

B. Analysis of the Proposed Negotiation Model

For analysis purposes, we give TA semantics to the REMES HDCL model [1], [2]. The actual transformation is explained informally in the example of Section IV-C. In the proposed negotiation model one could think about different types of analyses. Since the model is time-constrained, it is possible to perform timing analysis, e.g., to compute what is the response time of the service provider. The described negotiation model is equipped with the cost model that enables analyzing the maximum and minimum cost of the negotiation process. If we consider the TAn or Priced Timed Automaton (PTAn) model as the semantic translation of a REMES model, then some analysis goals might be expressed by the following WCTL [14] properties that TAn or PTAn model can be checked against:

$$AF_{t \leq t_{max}}\ final\_deal \quad (4)$$
$$AG\ (negotiation \Rightarrow AF_{cost \leq n} final\_deal) \quad (5)$$
$$EF_{cost \leq n}\ final\_deal \quad (6)$$
$$AG\ (negotiation \Rightarrow EF_{cost \leq n} final\_deal) \quad (7)$$
$$EF\ (t \leq\ t_{max} \wedge\ contract \wedge\ u \geq val) \quad (8)$$

Properties (4), (5), (7) are liveness properties, while properties (6) and (8) are reachability properties. We say that the first two properties specify *strong feasibility* of the model: property (4) requires that for all execution paths, the target location *final_deal* is eventually reached within time t less or equal to the maximum allowed time t_{max}; property (5) states that, for all paths, it is always the case that, once in location $negotiation$, the *cost* of eventually reaching location *final_deal* will be no more than n, regardless of how location *final_deal* is reached. Property (6) models *weak feasibility*, meaning that the target location *final_deal* may be reached within a total cost of n. Property (7) states that for all paths, it is always the case that once location $negotiation$ is reached, there exists a way by which location *final_deal* will be eventually reached within cost n. The last property (8) states that there exists a path in which a contract can be signed ($contract$ holds) within the given timing constraints, that is, $t \leq t_{max}$, and maximized utility function ($u \geq val$).

In the following section, we introduce an insurance scenario example to illustrate our approach.

IV. EXAMPLE: AN INSURANCE SCENARIO

In the car insurance branch the biggest concern is handling claims filed by the insured customers. In most cases the process still relies on traditional methods that involve input and effort of several specialized persons, usually working in different departments of an insurance company. The claims handling process becomes both time consuming and expensive. Therefore, insurance companies show interest in finding more automated, less expensive, and quicker ways to serve their customers.

In this paper, we describe an imaginary car insurance company called Damage Secure Center similar to the one presented by Paurobally et al. [18]. The company is specialized in managing tasks related to car damage claims on behalf of insurance companies. The goal is to enhance the quality, efficiency, and to reduce the cost of claims handling between client, the car damage repair shops, and insurance companies. The involved parties have different preferences regarding the car repair claims. A customer wants to get her/his car repaired as soon as possible. An insurance company that has a duty to pay for a repair (claim settlement) would like to do it under the best circumstances, that is the lowest price, as close as possible to the clients current location, as soon as possible, etc. In this example we assume several repair shops, representing themselves and negotiating on the price of the requested repair. The benefit of such a scenario is twofold: (i) the process becomes more efficient since it is not manual anymore; (ii) the prices offered by repair shops most likely will drop since it is possible to negotiate the final price.

The example assumes a client being insured by an insurance company. In case of the car damage the client sends a damage report to an insurance company that contacts Damage Secure Center company (the mediator) that negotiates the prices of the car damage repair with repair shops on behalf of the insurance company. The Damage Secure Center selects several available repair shops, based on their competence and performs an iterative CNP negotiation in order to choose the most suitable one. When the most suitable repair shop is chosen, the insurance company signs the contract and pays for the provided service.

In this example we are interested to model the negotiation process with respect to the following properties:

- *Price* - represents several different types of costs such as labor cost, repair cost, and a profit margin calculated

on the repair shop side. On the other hand for Damage Secure Center, the price is the maximum price that an insurance company is willing to pay for a car damage repair.
- *Location* - describes the distance of the repair shops from an insured client that has reported a car damage.
- *Speed of repair* - is described in terms of the time of the repair in which the repair shop is fixing the damage.

For parties involved in the negotiation process we have defined negotiation preferences that are used to determine which values to offer, accept, or respond with. For example, a repair shop starts the negotiation by offering a price within the predefined minimum and maximum values for each car damage repair type. In case that the negotiation process does not go as expected, i.e., takes too much time without reaching any agreement, a repair shop can offer a marginal price that is lower than previously defined minimum cost, but still acceptable by that repair shop (as compared to abandoning the negotiation).

In the following section we present negotiation strategies of interest for the described insurance scenario.

A. Negotiation strategies

The goal of the negotiation strategy is to determine the best course of actions that will lead to an agreement. The essence of negotiation strategies is creating offers/counter-offers in the acceptable range of values predefined by the involved negotiation parties, sometimes defining some acceptable trade-offs to steer the negotiation towards convergence. In this paper we describe two negotiation strategies, *price-driven negotiation strategy* and *time dependent negotiation strategy with marginal cost*, suitable in the context of the presented insurance scenario example.

1) Price-driven strategy: Each involved party during the negotiation process has some desired values for the prices to be required or offered. In our example the mediator Damage Secure Center would like to get as low as possible price, to serve the interests of the insurance company. On the other hand, the price that a repair shop is willing to offer lays in the interval [min_{cost}, max_{cost}] calculated based on the price of the spare part for the specific type of the car damage (partCost[damType]) and labor cost (minLaborCost[damType] and maxLaborCost[damType]), as follows:

$$min_{cost} = partCost[damType] + minLaborCost[damType]$$
$$max_{cost} = partCost[damType] + maxLaborCost[damType]$$

The goal of a repair shop is to get the highest possible price for the repair, as close as possible to its maximum price, yet in the worst case not below its minimum price.

In case that the price offered by a repair shop from the interval [min_{cost}, max_{cost}] is greater than the price that Damage Secure Center is willing to agree upon, the negotiation goes to the next iteration. In the next iteration the interval of possible prices offered by the repair shop is narrowed with the maximum price value offered in the previous iteration. In this way, the range of possible prices gets as close as possible to the desired values, but still satisfies the price restrictions of all involved parties. In case that the negotiation continues in a large number of iterations, the model could be bounded by a maximum number of allowed iterations.

2) Time-driven strategy with marginal cost: A time-driven negotiation strategy has similar principles of offering and counter-offering as the price-driven strategy. The prices proposed by the repair shop are again from the interval [min_{cost}, max_{cost}] with the interval being narrowed in each iteration of the negotiation process. In this strategy, each repair shop defines a marginal cost that is the maximal allowed deviation from the minimum price in the given interval.

This strategy assumes that there exists a time bound (t_{max}) on the negotiation process. The idea is that by this time some agreement should be made, or the involved parties should be very close to agree upon something with the final agreed price in the range [min_{cost}, max_{cost}]. In case that this time is exceeded with no agreement settled, and the current offered price (offered_price) is greater than the price required by the Damage Secure Center (req_price) with an amount less or equal to the marginal cost (marginal_cost), then the contract can be signed and the agreed price is favoring the Damage Secure Center.

In the following section, we present the HDCL-based negotiation model for the above described insurance scenario.

B. Modeling Negotiation for the Insurance Scenario

Let us consider a model that consists of one client (C), an insurance company (IC), a Damage Secure Center (DSC), and three repair shops (RS01, RS02, and RS03). In HDCL each involved party has to be introduced through the declarative part as described in Section III-A. Through the same declarative part the global variables used in this example are first declared. In this example we make use of three lists for the negotiation model (list_car_damage_report, list_forward_report, and list_neg).

In this model we have defined three types of the car damage that a client can report (damType). Repair shop RS01 can offer its services in case that the car damage is of type 0 or 1, repair shop RS02 can serve car damage types 1 and 2, and the last repair shop RS03 can repair all three possible types of the car damage. These repair shops are located at two different locations. Repair shops RS01 and RS02 are at location0, while repair shop RS03 is at location1, assumed far enough from each other. In the code below, services are first declared, followed by their creation and adding to the respective lists in order to model service composition.

```
int $damType, location, ISprice, price\_RS[3], price_{client}$
bool $car\_status, contract$
declare $Client ::=<$
    $web\ service,$
    $3,$
    $20,$
    $passive,$
    $(car\_status == false \wedge damType == 1$
    $\wedge\ location == 0 \wedge\ price_{client} \leq\ 7),$
    $(car\_status == true \wedge location == 0) >$
    **declare** $IC$ ::=<
        $web\ service$,
        5,
        20,
        $passive$,
        $(damType == 0 \vee damType == 1 \vee damType == 2)$,
        $(ISprice \leq 12)$ >
    **declare** $DSC$ ::=<
        $web\ service$,
        3,
        20,
        $passive$,
        $(damType == 0 \vee damType == 1 \vee damType == 2)$,
        $(ISprice \leq 12)$ >
    **declare** $RS01$ ::=<
        $web\ service$,
        7,
        20,
        $passive$,
        $(5 \leq price\_RS[1] \leq 14 \wedge location == 0$
        $\wedge\ (damType == 0 \vee damType == 1))$,
        $(location == 0 \wedge\ (damType == 0 \vee damType == 1))$ >
    **declare** $RS02$ ::=<
        $web\ service$,
        6,
        20,
        $passive$,
        $(5 \leq price\_RS[2] \leq 14 \wedge location == 0$
        $\wedge\ (damType == 1 \vee damType == 2))$,
        $(location == 0 \wedge\ (damType == 1 \vee damType == 2))$ >
    **declare** $RS03$ ::=<
        $web\ service$,
        5,
        20,
        $passive$,
        $(4 \leq price\_RS[3] \leq 10 \wedge location == 1$
        $\wedge\ (damType == 0 \vee damType == 1 \vee damType == 2))$,
        $((damType == 0 \vee damType == 1 \vee damType == 2) \wedge$
        $\wedge\ location == 1)$ >
**create** $Client, IC, DSC, RS01, RS02, RS03$
**create** $list\_car\_damage\_report, list\_forward\_report, list\_neg$
**add** $Client\ list\_car\_damage\_report$
**add** $IC\ list\_car\_damage\_report$
**add** $IC\ list\_forward\_report$
**add** $DSC\ list\_forward\_report$
**add** $DSC\ list\_neg$
**add** $RS01\ list\_neg$
**add** $RS02\ list\_neg$
**add** $RS03\ list\_neg$
```

When a car damage occurs, the client (C) sends a request for car repair to the insurance company (IC) that contains the car location and the damage type. Let us assume that the client has to report the car damage type 1 and that the car is at location0. The client also states the maximum price he/she is willing to pay for the car repair ($price_{client} \leq\ 7$).

$$DCL\_req ::= (list\_car\_damage\_report, \|_{SYNC\text{-}and}, damType == 1 \wedge \wedge\ location == location0 \wedge price_{client} \leq\ 7)$$

This request is forwarded to the Damage Secure Center (DSC) together with information on the maximum price that the insurance company is willing to pay for the car damage repair (ISprice).

$$DCL\_req ::= (list\_forward\_report, \|_{SYNC\text{-}and}, ISprice \leq 12 \wedge damType == 1 \wedge location == location0)$$

In terms of the negotiation model the Damage Secure Center (DSC) acts as a coordinator between the insurance company as the client and repair shops as contractors. After receiving a request, (DSC) asks all repair shops to offer their prices. Even if some repair shop is not at the same location as the client, it will be asked to create an offer for a repair since the offered price might be lower than the prices offered by other two repair shops including the traveling cost to that location. It might be the case that it is acceptable to pay more to tow the car to this location, but still make a significant saving in the repair cost. For this example, the negotiation model is then as follows:

```
**bool** $contract := false$;
**int** $n := 1, m := 1$
**int** $price\_min\_RS[1] := 5, price\_max\_RS[1] := 14$
**int** $price\_min\_RS[2] := 5, price\_max\_RS[2] := 14$
**int** $price\_min\_RS[3] := 4, price\_max\_RS[3] := 10$
**DO**
  $DCL\_req ::= (list\_neg, \|_{SYNC\text{-}and}, damType == 1 \wedge$
              $\wedge\ location == location0 \wedge\ price_{client} \leq\ 7)$
  $DCL\_offer ::= (list\_neg, \|_{SYNC\text{-}and},$
              $price[m] \in [price\_min\_RS[m], price\_max\_RS[m]])$
  **DO**
    $price\_RS[m] := negotiation(price)$
    $price\_max\_RS[m] := price\_RS[m]$
    $m := m + 1$
  **OD** $(m \leq\ 3)$
  $m := 1$
  **DO**
    **if** $(price_{client} \geq\ price\_RS[m])$ **then**
    $contract := true$
    $m := m + 1$
    **fi**
  **OD** $(m \leq\ 3)$
  $n := n + 1$
**OD** $(n \leq\ 5 \wedge\ \neg contract)$
**DO**
  **if** $(price_{client} < price\_RS[m])$ **then**
  $contract := false$
  **fi**
**OD** $(m \leq\ 3)$
```

After receiving a call for offers from the (DSC), each repair shop calculates the minimum and the maximum price for a given damage type (i.e., price_min_RS[m] and price_max_RS[m], where m is an integer variable that represent number of available repair shops). This sets the initial bound for a offer to be created. In each iteration of the negotiation process each repair shop provides an offer value on price that is negotiated (i.e., price_RS[m]) via function negotiation(price), which is always a value between minimum and maximum price for a given damage type. The maximum value of the price price_max_RS[m] is being updated in each negotiation iteration by the previously offered price meaning that the price interval is progressively narrowed.

The negotiation process iterates as long as the given condition holds (n $\leq$ 5 $\wedge$ contract == false). Depending on the strategy used, the negotiation process might have different outcomes. One outcome might be that both sides agree on some price in few iterations and that the contract (contract := true) is quickly signed, or that the process iterates as long as it is allowed by the negotiation model and finishes without contract (contract := false) signed. Another scenario might be that the negotiation process has timing constraints. In that case the negotiation might iterate as long as the timing constraint allows, and then in the case that no agreement is reached, a repair shop can agree on a price lower than its set minimum price with maximum the amount of the marginal cost.

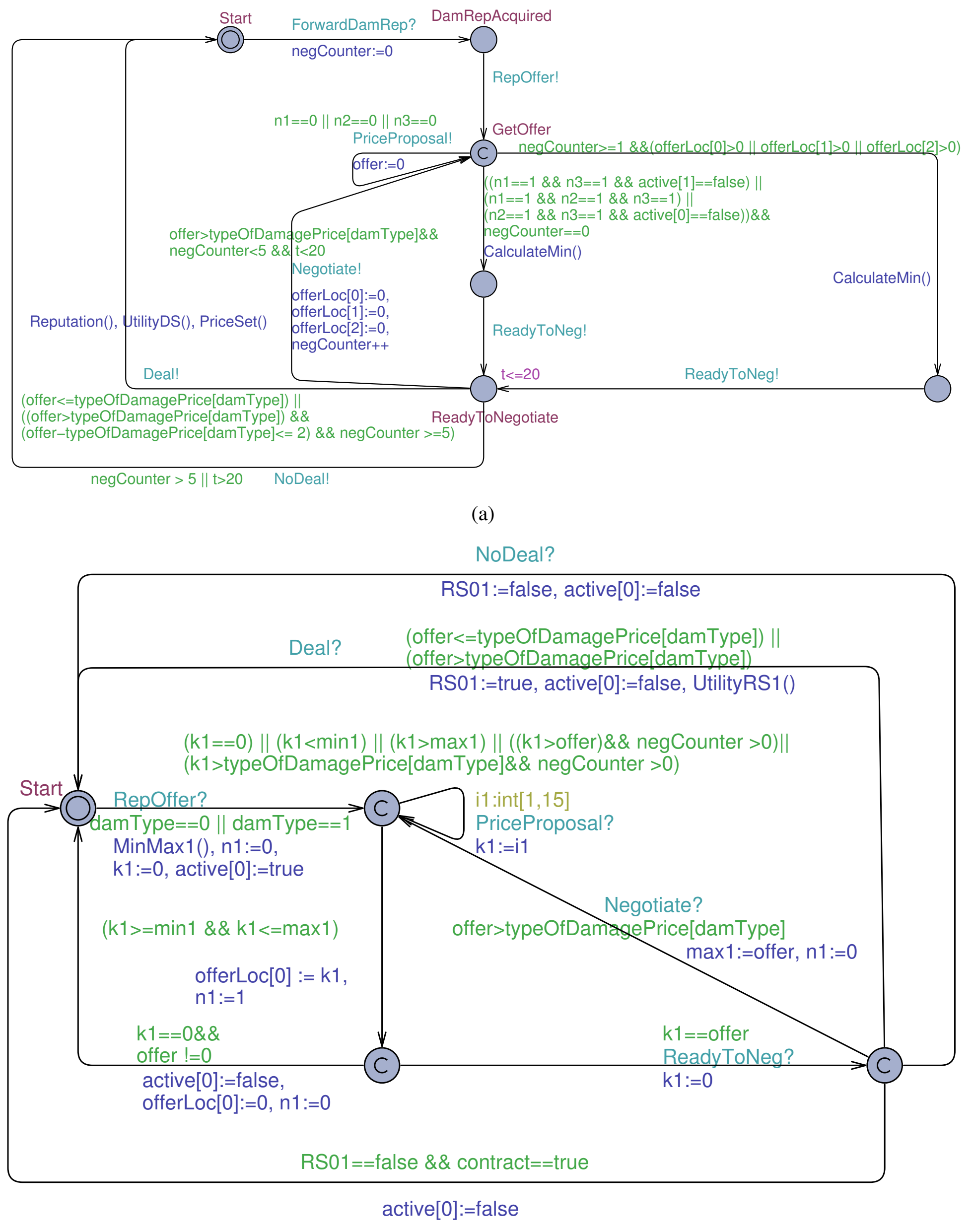
Start
ForwardDamRep?
negCounter:=0
DamRepAcquired
RepOffer!
n1==0 || n2==0 || n3==0
PriceProposal!
offer:=0
GetOffer
negCounter>=1 &&(offerLoc[0]>0 || offerLoc[1]>0 || offerLoc[2]>0)
((n1==1 && n3==1 && active[1]==false) ||
(n1==1 && n2==1 && n3==1) ||
(n2==1 && n3==1 && active[0]==false))&&
negCounter==0
CalculateMin()
offer>typeOfDamagePrice[damType]&&
negCounter<5 && t<20
Negotiate!
offerLoc[0]:=0,
offerLoc[1]:=0,
offerLoc[2]:=0,
negCounter++
CalculateMin()
Reputation(), UtilityDS(), PriceSet()
ReadyToNeg!
Deal!
t<=20
ReadyToNeg!
(offer<=typeOfDamagePrice[damType]) ||
((offer>typeOfDamagePrice[damType]) &&
(offer-typeOfDamagePrice[damType]<= 2) && negCounter >=5)
ReadyToNegotiate
negCounter > 5 || t>20
NoDeal!
(a)
NoDeal?
RS01:=false, active[0]:=false
Deal?
(offer<=typeOfDamagePrice[damType]) ||
(offer>typeOfDamagePrice[damType])
RS01:=true, active[0]:=false, UtilityRS1()
(k1==0) || (k1<min1) || (k1>max1) || ((k1>offer)&& negCounter >0)||
(k1>typeOfDamagePrice[damType]&& negCounter >0)
Start
RepOffer?
damType==0 || damType==1
MinMax1(), n1:=0,
k1:=0, active[0]:=true
i1:int[1,15]
PriceProposal?
k1:=i1
Negotiate?
offer>typeOfDamagePrice[damType]
max1:=offer, n1:=0
(k1>=min1 && k1<=max1)
offerLoc[0] := k1,
n1:=1
k1==0&&
offer !=0
active[0]:=false,
offerLoc[0]:=0, n1:=0
k1==offer
ReadyToNeg?
k1:=0
RS01==false && contract==true
active[0]:=false
(b)

Fig. 2. The timed automata model of DSC and RS01

C. Analyzing the TA Model of the Insurance Scenario

We have analyzed the service-based car insurance negotiation example, as a network of six TA models, in the UPPAAL model checker [5]. The TA model of Client is shown in Fig. 1 and described in Section II-B. The TA of Damage Secure Center (DSC) and the repair shop (RS01) are depicted in Fig. 2 (a) and Fig. 2 (b), respectively.

Both automata consist of one initial location Start. The DSC automaton gets invoked by the insurance company (IC) automaton via synchronization channel ForwardDamRep for the mediation of finding a suitable repair shop for the given car damage type. This synchronization corresponds to the DCL_req in HDCL in which an insurance company defines its own constraints on the requested car damage repair.

To serve this request, DSC automaton uses synchronization channel RepOffer to propagate this request to repair shops RS01, RS02, and RS03. All repair shops calculate their minimum and maximum price for a given damage type (damType) using MinMax() function. To offer the price of the repair, repair shops use synchronization channel PriceProposal. The offers are stored into the array offerLoc[3] that is used by the DSC automaton function CalculateMin() to determine the minimum

[5]See the web page www.uppaal.org for more information about the UPPAAL tool.

offered price. In case that this price is smaller or equal to the price that the insurance company is willing to pay, the contract is signed (contract == true) and the agreement is propagated via synchronization channel Deal. Otherwise, the DSC automaton and the repair shop with the lowest offered price continue negotiating via synchronization channels ReadyToNeg and Negotiate. In this example we have encoded maximum of 5 negotiation iterations (negCounter $\leq$ 5). Bounded integer variables n1, n2, and n3 are used to keep a track of the proposals by repair shops RS01, RS02, RS03, respectively. The integer array active[3] indicates which repair shops are active in the negotiation process. In case that no agreement is reached, DSC closes the negotiation via synchronization channel NoDeal. The TA description above is a transformation of HDCL "negotiation service" described in Section IV-B.

In order to analyze the negotiation we encode the DSC utility function (uds, see below Eq. 9) as a weighted sum of negotiation preferences (i.e., price, location, time-to-repair). The function is calculated for all repair shops and DSC given that they have different priorities for different preferences. From the point of view of the repair shop the goal would be to maximize the utility, since it is heavily influenced by the final price. On the other hand, the DSC would like to minimize the utility function.

$$\text{uds} = \text{price\_offer} * \text{wds1} + \text{location[i]} * \text{wds2} + \text{RS}_n\text{timeToRepair[j]} * \text{wds3} \quad (9)$$

In the utility function described in Eq. 9 weights wds1, wds2, and wds3 express the preferences of the DSC. The highest value is assigned to weight wds1 meaning that the price has the highest importance for DSC, while the lowest value is given to weight wds2 showing that the location of the client does not play a big role when choosing the repair shop.

Next, we define the repair shop utility function as follows:

$$\text{urs}_m = \text{price\_offer} * \text{wrs1} + \text{location[i]} * \text{wrs2} + \text{RS}_n\text{timeToRepair[j]} * \text{wrs3} \quad (10)$$

where m $\in$ [1,3] represents the number of repair shops involved in the negotiation process. On the other hand, for a repair shop the most important preference is the price with the highest given weight wrs1. In comparison with DSC the actual location of the client is more important than the time-to-repair the car damage, since to tow the damaged car to the repair shops location might be time consuming and might bring the additional and unexpected cost. The lowest weight is assigned to the time required to perform the repair.

In order to minimize the utility function for DSC, we have analyzed the cases when repair shops offer minimum possible prices to perform the car damage repair. The results are showed in Table I.

To be able to compare the results from Table I we have analyzed the utility function for each repair shop, and for each damage type with the same prices as used to minimize DSC utility function. One can notice that in the case of car damage types damType1 and damType2, from the perspective of both DSC and repair shops there is no big difference in prices of the car repair at different locations, while in case of the damage type damType0 it is in the best interest of DSC to repair the car in the repair shop RS03 even if that would mean to tow the car from location1 to location0 to get the cheaper repair. On the other hand, as presented in Table II in case of the same damage type damType0, the utility function is maximum if the car damage repair is performed at the same location, that is, location0.

	damType0	damType1	damType2
RS01	20	30	-
RS02	-	28	30
RS03	16	30	32

TABLE I
VALUES OF THE MINIMIZED UTILITY FUNCTION OF THE DSC

	damType0	damType1	damType2
RS01	16	24	-
RS02	-	23	24
RS03	12	22	26

TABLE II
VALUES OF THE UTILITY FUNCTION OF THE RESPECTIVE REPAIR SHOPS FOR THE SAME PRICE VALUES AS IN TABLE I

It is also interesting to analyze the maximized utility function values for the respective repair shop. When maximizing the function, we take into consideration the negotiation prices that are maximum from the perspective of repair shops, yet acceptable by DSC hence leading to an agreement.

	damType0	damType1	damType2
RS01	19	45	-
RS02	-	44	42
RS03	18	40	55

TABLE III
VALUES OF THE MAXIMIZED UTILITY FUNCTION OF THE RESPECTIVE REPAIR SHOPS

If we compare results in Table III with those in Table II we can notice that it is not a significant change in values for the car damage type damType0. The reason for this is probably the fact that the maximum acceptable price by DSC that leads to an agreement is closer to the repair shops' minimum price than to the maximum one. On the other hand, other two repair shops would benefit a lot in terms of profit in case that agreements are based on these values.

Values of the utility function of DSC in Table IV for the car damage types damType1 and damType2 increase significantly compared to the minimized values of the same function presented in Table I, since the maximum acceptable prices by DSC are much higher from the minimum allowed prices by the repair shops.

We also check the modeled example against safety properties given in Eq. 11 and Eq. 12:

	damType0	damType1	damType2
RS01	23	51	-
RS02	-	49	48
RS03	22	48	59

TABLE IV
VALUES OF THE UTILITY FUNCTION OF DSC FOR THE SAME PRICE VALUES AS IN TABLE III

$$AG\ not\ deadlock \tag{11}$$

where the first property (Eq. 11) checks whether the modeled negotiation process is deadlock free in all existing traces, while the property defined in Eq. 12 ensures that after 5 negotiation iterations a new negotiation round is never started.

$$AG\ not\ (DS.ReadyToNegotiate\ \wedge\ negCounter\ >\ 5\ \wedge\ DS.GetOffer) \tag{12}$$

If we assume a model with an encoded time-driven strategy, and a marginal cost, then we can verify whether the contract can be signed within given timing constraints, that is, $t \leq 20$ time units. Here we check the property for the car damage type damType1, but in general the property can be checked for any car damage type. The property to be verified has the following form:

$$AG\ not\ (t \geq\ 20\ \wedge\ negCounter\ >\ 5 \wedge\ contract == true\ \wedge damType == 1) \tag{13}$$

The property given in Eq. 13 is satisfied, and the final offered price is equal to 4 price units, that is, the price an insurance company is willing to pay. In this case, the negotiation finishes in favor of the insurance company. Also, it is interesting to do some reachability analysis. We could check whether there exists a trace in which, assuming a car damage type damType2, a contract can be signed within a prescribed time, that is, $t \leq 20$ time units, while maximizing the utility function urs_2.

$$EF\ (t \leq\ 20\ \wedge\ contract == true \wedge\ damType == 2 \wedge\ urs_2\ \geq\ 40) \tag{14}$$

We have verified, in UPPAAL, properties 11- 14, and the model satisfies them all.

V. DISCUSSION AND RELATED WORK

Lapadula et al. provide a description of modeling publication, discovery, negotiation, deployment, and execution of service-oriented applications in COWS [3]. COWS is a WS-BPEL-inspired process calculus, which can be seen as a lower level modeling language suitable for specifying, combining, analyzing services, while modeling their dynamic behavior. For the analysis purposes, the language can be translated to CMC model checker. In comparison to this approach, our approach offers an intuitive and expressive service model, with given semantics in UPPAAL's TA, which allows for both functional as well as extra-functional negotiation properties. Sierra et al. describe a formal model for negotiation between autonomous agents in service-oriented environments [4]. The service-oriented model is a modified version of the general negotiation model proposed by Faratin et al. [19]. The paper presents several negotiation schemes, including generation of the initial offer, evaluation of the incoming proposals, and generation of counter proposals. However, no analysis support has been mentioned. Comuzzi et al. present an automated approach to web service QoS negotiation [20]. The negotiation is performed via Negotiation Broker to which both a consumer and a service provider notify their preferences on QoS attributes and negotiation strategies by specifying the value of a relatively small set of parameters. In some later work Comuzzi et al. propose a semantic-based framework to support negotiation process in SOA [21]. The benefit of this approach is that framework allows to service client and a service provider to express their capabilities in terms of the negotiation protocols they are able to support and the actions they are able to perform, and based on that proposes a negotiation protocol to be used. Again, both papers offers a rich theoretical foundation for automated negotiation process, but compared to our approach lack the formal analysis of the negotiated QoS. Resinas et al. provide a description of automated agreement negotiation system based on a bargaining protocol called NegoFAST-Bargaining [22]. The architecture includes a rich environment to first identify key element in the negotiation process, then to model it together with its corresponding processes, and finally to create scenarios to be validated. However, the approach does not provides means for formal analysis as one described in our paper. Paurobally et al. describe a a way to deploy multi-agent negotiation techniques to facilitate dynamic negotiation for Grid resources in order to provide an adaptive and autonomous Grid [18]. Moreover, they describe the deployment of CNP and its corresponding strategies for negotiation between web services. The approach offers a rich environment to model the negotiation process using CNP, but in terms of the analysis it is limited to monitoring the modeled system while it lacks support for formal analysis.

VI. CONCLUSIONS

In this paper, we have presented an approach that accommodates modeling and analysis of service negotiation between two or more parties based on an iterative form of the CNP for web services. Although we have implemented only the CNP negotiation protocol and two strategies, price-, and time-driven, the negotiation model is general and not limited to these. Our model is an analyzable high-level description of the negotiation between service clients and providers, which so often characterizes SOS. The model has an implicit notion of time, and supports annotations in terms of price, quality, or other parameters, all modeled by the REMES textual service composition language HDCL. The crux of the model is that it has a formal timed automata semantics, which lets one verify various model properties, for all possible executions, which is not achievable in principle by any simulation or testing technique.

We have applied a narrower version of it in the car insurance example, for which we have shown how to analyze the negotiation model against safety properties, but also against specified timing and utility constraints. We accomplish these, by transforming the model into a network of Timed Automata, which we analyze by using the UPPAAL model checker. Real-world negotiation processes are indeed more complex than our high-level proposed model. Consequently, as future work, we plan to validate the proposed approach on a more complex case study, and possibly improve our models to ensure better scaling and to cover richer behavior. In addition, we would like to explore the ways in which we could connect our language to WS-BPEL language, such that the large analysis spectrum covered by our approach reaches and becomes accessible to a broader community.

VII. Acknowledgments

This work was funded by the VR Contesse project and Mälardalen University. The authors would like to thank the Formal Modeling and Analysis of Embedded Systems group at Mälardalen University for fruitful discussions and valuable comments.

References

[1] C. Seceleanu, A. Vulgarakis, and P. Pettersson, "Remes: A resource model for embedded systems," in *In Proc. of the 14th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS 2009)*. IEEE Computer Society, June 2009.

[2] A. Causevic, C. Seceleanu, and P. Pettersson, "Modeling and reasoning about service behaviors and their compositions," in *Proceedings of 4th International Symposium On Leveraging Applications of Formal Methods, Verification and Validation (ISOLA 2010), Formal Methods in Model-Driven Development for Service-Oriented and Cloud Computing track*. Springer LNCS, October 2010.

[3] A. Lapadula, R. Pugliese, and F. Tiezzi, "Service discovery and negotiation with cows," *Electron. Notes Theor. Comput. Sci.*, vol. 200, pp. 133–154, May 2008.

[4] C. Sierra, P. Faratin, and N. R. Jennings, "A service-oriented negotiation model between autonomous agents," in *Proceedings of the 8th European Workshop on Modelling Autonomous Agents in a Multi-Agent World: Multi-Agent Rationality*. London, UK: Springer-Verlag, 1997, pp. 17–35. [Online]. Available: http://dl.acm.org/citation.cfm?id=646909.710674

[5] R. Alur and D. L. Dill, "A theory of timed automata," *Theoretical Computer Science*, vol. 126, no. 2, pp. 183–235, 1994. [Online]. Available: citeseer.nj.nec.com/alur94theory.html

[6] D. Ivanov, M. Orlic, C. Seceleanu, and A. Vulgarakis, "Remes toolchain - a set of integrated tools for behavioral modeling and analysis of embedded systems," in *Proceedings of the 25th IEEE/ACM International Conference on Automated Software Engineering (ASE 2010)*, September 2010.

[7] D. Ivanov, "Integrating formal analysis methods in progress ide," Master of Science Thesis, Malardalen Research and Technology Centre, Vasteras, Sweden, June 2011.

[8] E. P. Enoiu, R. Marinescu, A. Causevic, and C. Seceleanu, "A design tool for service-oriented systems," in *9th International Workshop on Formal Engineering approaches to Software Components and Architectures (FESCA 2012)*. ENTCS, March 2012.

[9] E. W. Dijkstra and C. S. Scholten, *Predicate calculus and program semantics*. New York, NY, USA: Springer-Verlag New York, Inc., 1990.

[10] M. Orlić, "Resource usage prediction in component-based software systems," PhD thesis, Faculty of electrical engineering and computing, University of Zagreb, November 2010.

[11] J. Bengtsson and W. Yi, "Timed automata: Semantics, algorithms and tools." Springer-Verlag, 2004, pp. 87–124.

[12] R. Alur, C. Courcoubetis, and D. Dill, "Model-checking for real-time systems," in *Logic in Computer Science, 1990. LICS '90, Proceedings., Fifth Annual IEEE Symposium on e*, jun 1990, pp. 414 –425.

[13] R. Alur, C. Courcoubetis, and D. L. Dill, "Model-checking in dense real-time," *Inf. Comput.*, vol. 104, no. 1, pp. 2–34, 1993.

[14] T. Brihaye, V. Bruyère, and J.-F. Raskin, "Model-checking for weighted timed automata," in *Proc. of FORMATS-FTRTFTâĂŹ04*, ser. Lecture Notes in Computer Science, no. 3253. Springer–Verlag, 2004, pp. 277–292.

[15] D. G. Pruitt, *Negotiation Behavior*. Academic Press Inc., 1981.

[16] A. Causevic, C. Seceleanu, and P. Pettersson, "Checking correctness of services modeled as priced timed automata," in *Proceedings of 5th International Symposium On Leveraging Applications of Formal Methods, Verification and Validation*. LNCS Proceedings (Springer Verlag), October 2012.

[17] A. Kümpel, I. Braun, J. Spillner, and A. Schill, "(Semi-) automatic negotiation of service level agreements," in *IADIS International Conference WWW/INTERNET 2010*, Timisoara, Romania, 2010, pp. 282–286.

[18] S. Paurobally, V. A. M. Tamma, and M. Wooldridge, "A framework for web service negotiation," *TAAS*, vol. 2, no. 4, 2007.

[19] P. Faratin, C. Sierra, and N. R. Jennings, "Negotiation decision functions for autonomous agents," *INTERNATIONAL JOURNAL OF ROBOTICS AND AUTONOMOUS SYSTEMS*, vol. 24, pp. 3–4, 1998.

[20] M. Comuzzi and B. Pernici, "An architecture for flexible web service qos negotiation," in *Proceedings of the Ninth IEEE International EDOC Enterprise Computing Conference*, ser. EDOC '05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 70–82. [Online]. Available: http://dx.doi.org/10.1109/EDOC.2005.4

[21] M. Comuzzi, K. Kritikos, and P. Plebani, "A semantic based framework for supporting negotiation in service oriented architectures," in *CEC*, 2009, pp. 137–145.

[22] M. Resinas, P. Fernández, and R. Corchuelo, "A bargaining-specific architecture for supporting automated service agreement negotiation systems," *Science of Computer Programming*, vol. 77, no. 1, pp. 4 – 28, 2012, <ce:title>System and Software Solution Oriented Architectures</ce:title>. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167642310001851