Distributed Energy Management Case Study: A Formal Approach to Analyzing Utility Functions

Aida Čaušević, Cristina Seceleanu, and Paul Pettersson

Mälardalen Real-Time Research Centre (MRTC),
Mälardalen University, Västerås, Sweden
{aida.causevic,cristina.seceleanu,paul.pettersson}@mdh.se

Abstract. The service-oriented paradigm has been established to enable quicker development of new applications from already existing services. Service negotiation is a key technique to provide a way of deciding and choosing the most suitable service, out of possibly many services delivering similar functionality but having different response times, resource usages, prices, etc. In this paper, we present a formal approach to the clients-providers negotiation of distributed energy management. The models are described in our recently introduced REMES HDCL language, with timed automata semantics that allows us to apply UPPAAL-based tools for model-checking various scenarios of service negotiation. Our target is to compute ways of reaching the price- and reliability-optimal values of the utility function, at the end of the service negotiation.

1 Introduction

Service-oriented systems (SOS) represent a promising approach that accommodates the necessary conceptual foundation to provide quicker application development out of loosely coupled software entities, called services. The SOS paradigm also provides a way to connect new systems and services with legacy systems. Service negotiation is a key technique towards deciding and choosing the most suitable service, out of possibly many services delivering similar functionality but having different response times, resource usages, prices, etc.

The literature describes several rather theoretical results that tackle this topic [1–4] but lack constructs for formal analysis. The benefit of attaching such support to a service negotiation protocol is the capability of verifying if the negotiation design meets its specified requirements. Also, formal verification allows one to compute various quality-of-service (QoS)- optimal paths corresponding to different negotiation scenarios.

Motivated by the above, in this paper we describe the modeling and formal analysis of a distributed energy management in an open energy market, similar to one described by Mobach [5]. In an open energy market the traditional energy management does not suffice anymore, since it is required to facilitate interactions between market participants; this means that the management should be supported by a model that allows energy providers to establish agreements with energy consumers w.r.t. the supply of energy. The model of the energy market is described in Section 3.

Such a model involving customer-provider negotiation needs to be analyzed for various strategies that aim at reaching an agreement beneficial for both sides, against specified requirements. The goal of the analysis presented in this paper is also to validate

T. Margaria and B. Steffen (Eds.): ISoLA 2014, Part II, LNCS 8803, pp. 74–87, 2014.

our service-oriented modeling and analysis framework, that is briefly recalled in Section 2. The framework consists of the resource-aware timed behavioral modeling language REMES [6], reviewed in Section 2.1, and its underlying formal model described in terms of timed automata (TA) networks [7,8] (see Section 2.2). The negotiation model is obtained by composing REMES services, within a corresponding textual service composition language called Hierarchical Dynamic Composition Language (HDCL) (see Section 4), via operators that have been defined formally in our previous work [9]. The salient point of the approach is the fact that the obtained negotiation model can be analyzed against safety, timing, and utility constraints, for all possible behaviors of the parties. This can be achieved by transforming the negotiation model into a TA formal framework, which has a precise underlying semantics that allows its analysis with UPPAAL tools, for functional and extra-functional behaviors (timing and resource-wise behaviors) [10]. We show how to compute the price- and reliability-optimal values of the utility function, at the end of the service negotiation. The analysis of the energy negotiation process and its results are described in Section 5. Last but not least, we present some relevant related work in Section 6, before concluding the paper in Section 7.

2 Background

In this section we briefly overview the preliminaries on the REMES modeling language and the timed automata formalism, needed to comprehend the rest of the paper.

2.1 REMES - a Language for Behavioral Modeling of SOS

To describe service behavior in SOS, we use the dense-time hierarchical modeling language called REMES [6, 9]. The language is well-suited for abstract modeling, it is hierarchical, has an input/ouput distinction, a well-defined formal semantics, and tool support for SOS modeling and formal analysis [1] [10,11]. The formal analysis is accomplished by semantic transformation of REMES models into timed automata (TA) [7] or priced timed automata (PTA) [12], depending on the analysis type [10].

A service in REMES can be described graphically (as a mode), or textually, by a list of attributes (i.e., service type, capacity, time-to-serve, status, service precondition, and postcondition) exposed at the interface of the REMES service. A REMES service can be atomic, composite, but also employed in various types of compositions, resulting in new, more complex services. In order to model the synchronized behavior of parallel services we have previously introduced a special kind of REMES 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 are not constrained by any boolean enabling condition, called guard; in comparison, 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 at the end of their execution communicate via $\|_{\mathit{SYNC\text{-}and}}$ (all services take their respective exit edges at the same time

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

and mode finishes its execution), or $\|_{SYNC\text{-}or}$ (the mode finishes its execution as soon as one service has taken an exit edge) operators, respectively (see our previous work [9]).

In order to manipulate services, REMES supports service creation, deletion, composition, and replacement via REMES interface operations. An example of a create service operator is given in Eq. 1. Alongside the above operations, REMES is accompanied by a hierarchical dynamic composition language (HDCL) that facilitates modeling of nested sequential, parallel or synchronized services and their compositions.

$$\begin{aligned}
\textbf{[pre]} &: service\_name == \mathsf{NULL} \\
\mathsf{create} &: Type \times\ N \times N \times\ ''passive'' \times (\Sigma \to bool) \times (\Sigma \to bool) \to service\_name \qquad (1)\\
\textbf{\{post\}} &: service\_name \neq \mathsf{NULL} \wedge Type\ \in \{\text{web service, network service, embedded}\ \wedge \\
&\wedge\, capacity \geq 0 \wedge time - to - serve \geq 0 \wedge status\ =\ ''passive''
\end{aligned}$$

Our system is composed of REMES services that can be analyzed by transforming them into a formal network of TA that have precise semantics and can be model-checked against relevant properties (see the following section). In our recent work, we have introduced an analyzable negotiation model into the REMES language [13], that is, an analyzable high-level description of the negotiation between service clients and service providers. The model has an implicit notion of time and supports annotations in terms of price, quality, etc., all modeled by the REMES textual service composition language HDCL. The crux of the model is that it has a formal TA semantics, which lets one verify various model properties, for all possible executions. For a more thorough description of the REMES language, we refer the reader to our previous work [6,9,13,14].

2.2 Timed Automata

A timed automaton (TAn) [7,8] is a finite-state machine enriched with a set of clocks. All clocks are synchronized and assumed to be real-valued functions of time elapsed between events. In this work we use TA, as defined in the UPPAAL model-checker, which allows the use of data variables [15–17].

Let us assume a finite set of real-valued variables C ranging over x, y, etc., standing for clocks, V a finite set of all data (i.e., array, boolean, or integer), 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 $n \in N$. The elements of $\mathcal{B}(C)$ are called *clock constraints* over C. Similarly, we use $\mathcal{B}(V)$ to stand for the set of *non-clock constraints* that are conjunctive formulas of $i \sim j$ or $i \sim k$, where $i, j \in V$, $k \in \mathbb{Z}$ and $\sim\ \in \{<, \leq, =, \neq, \geq, >\}$. We use $\mathcal{B}(C, V)$ to denote the set of formulas that are conjunctions of clock constraints and non-clock constraints.

Definition 1. *A timed automaton A is a tuple $(L, l_0, C, V, I, Act, E)$ where: L is a finite set of locations, l_0 is the initial location, C is a finite set of clocks, V is a finite set of data variables, $I : L \to \mathcal{B}(C)$ assigns (clock) invariants to locations, $Act = \Sigma \cup \{\tau\}$ is a finite set of actions, where $\tau \notin \Sigma$ denotes internal or empty actions without synchronization, $E \subseteq L \times \mathcal{B}(C, V) \times Act \times R \times L$ is the set of edges, where R denotes the (clock) reset set. In the case of $(l, g, a, r, l') \in E$, we write $l \overset{g,a,r}{\to} l'$, where l is the source location, l' is the target location, 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.* ■

The semantics of TA is defined in terms of a labeled transition system. A state of a TAn is a pair (l, u), where l is a location, and $u : C \to R_+$ 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 $(l, u) \overset{d}{\to} (l, u \oplus d)$ 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 $(l, u) \overset{a}{\to} (l', u')$ corresponds to taking an edge $l \overset{g,a,r}{\to} l'$ 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 analysis to perform on a given TAn. 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. A network of TA, $A_1\|...\|A_n$, over C and Act, is defined as the parallel composition $A_1\|...\|A_n$ over C and Act. Semantically, a network describes a timed transition system obtained from the components, by requiring synchrony on delay transitions, and 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-state systems used to reason about sequence of events. Let AP be a set of atomic propositions, $p \in AP$. In this paper, a CTL formula ϕ is defined as follows:

$$\phi ::= \top \mid p \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \to \phi_2 \mid EF\phi \mid AF\phi \mid AG\phi$$

Each CTL well-defined formula is a pair of symbols. The first operator is a path operator, either A ("for All paths"), or E ("there Exists a path"). The latter operator, a temporal 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. [18, 19]. In the next section we present the details of the distributed energy management case study.

3 Energy Negotiation Model in REMES HDCL

The energy management system includes an energy consumer (i.e., client) that creates a request and communicates with energy providers via a mediator. A request contains information about requested amount of energy, required price per unit of energy, and expected reliability for energy to be provided. The supply of energy is based on a negotiation carried out between consumers and providers in possibly more than one round, assuming a certain strategy. The negotiation relies on advertisements, where energy providers specify the type of energy to be sold (i.e. depending on the energy source, diesel generators, wind turbine, etc.), available amount of energy, its reliability, and price per unit of energy. In this paper's negotiation model we assume an iterative form of a Contract Net Protocol (CNP). In the CNP there exist the following roles: the client (an energy consumer), the manager (a negotiation mediator), and the contractor (an energy provider). The manager gets a request from a client and aims at finding an appropriate contractor to fulfill the request via call for proposals (CFP). Based on the response from contractors the manager decides which offers to present to a client. Depending on the implemented strategy in each round both the contractors and clients aim to improve on their previous proposals and request, in order to come closer to the consensus.

The energy consumer is assumed to have a varying energy demand that has to be satisfied over a period of time (i.e., certain periods of the day have higher energy demand than the others), while at the same time energy providers have varying energy capacity. In this model, a single day is considered (see Fig. 1), with consumer requests coming every two hours. Every two hours a new negotiation starts and should provide energy for two subsequent hours. A consumer initiates the negotiation just before the moment the energy is to be claimed and used. After a request is created, the mediator negotiates with the available energy providers, on behalf of the consumer, creating competition between energy providers. As a result of each request an agreement should be signed covering the desired energy over a defined period of time. It might be the case that involved parties do not reach a consensus and in that case no agreement is established, meaning that the client might be out of energy for that period of time.

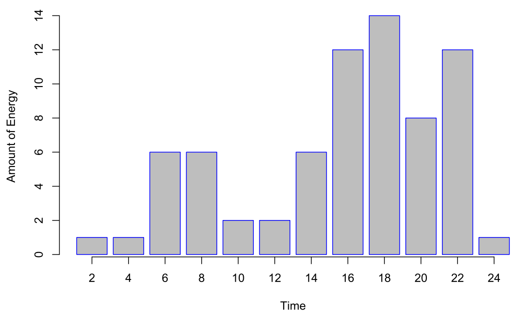


Fig. 1. An energy demand over a day

In our model, we have implemented three scenarios in which customers have encoded different behavior:

– *Scenario 1*: A customer has maximum bound on the price and the final acceptable price cannot be more than 20 price units higher than the initial requested price;
– *Scenario 2*: A customer has no maximum price value, the negotiation can continue until an agreement is conceived;
– *Scenario 3*: A customer adapts maximum price trying to get as close as possible to the offered price, but at the same time not to pay more than double initial price. The

idea behind this scenario is to get an agreement in the smallest possible number of price negotiations.

During the negotiation process the provider is not aware which strategy the client uses. In this paper, we provide a REMES - based description of the distributed energy management that is furthermore translated to the TA formal framework and analyzed against safety, timing, and utility constraints (described as a weighted sum of negotiation preferences). In the following section we will provide a REMES HDCL-based model of the energy negotiation described above.

4 REMES HDCL - Based Energy Negotiation Model

To enable a systematic and analyzable way to model the energy negotiation process, as described in Section 3, we provide the REMES HDCL description of the model. The model is based on the set of REMES interface operations and the hierarchical textual language HDCL [9].

Table 1. Service declaration

```
00 declare Service ::= <
01         service type : {web service},
02         capacity : N,
03         time_to_serve : N,
04         status : { passive, idle, active},
05         precondition : predicate,
06         postcondition : predicate >
07 create Mediator (web service, 2, 10, idle,
08 (req_client == false, contract == false),
09 (req_client == true, contract == true)) : Service
10 create Client (web service, 5, 20, idle,
11     (energy_amount == 0 ∧ t == 0s
12     ∧ min_c ≤ ppue ≤ max_c),
13     (energy_amount == k ∧ t ≤ 20s ∧)
14     min_c ≤ ppue ≤ max_c) : Service
15 create EP1 (web service, 5, 15, idle,
16      (energy_amount == ea1 ∧
17      min_ep1 ≤ ppue ≤ max_ep1 ∧
18      energy_reliability == r_ep1),
19      (energy_amount == ea1-k ∧
20      min_ep1 ≤ ppue ≤ max_ep1∧
21      energy_reliability == r_ep1)) : Service
22 create EP2 (web service, 2, 10, idle,
23      (energy_amount == ea2 ∧
24      min_ep2 ≤ ppue ≤ max_ep2 ∧
25      energy_reliability == r_ep2),
26      (energy_amount == ea2-k ∧
27      min_ep2 ≤ ppue ≤ max_ep2 ∧
28      energy_reliability == r_ep2)) : Service
29 declare List ::= <[service_name_0 : Service, . . .,
30                    service_name_n : Service]>
31 create list_request : List
32 create list_offer : List
33 add Client list_request
34 add Manager list_request
35 add EP1 list_offer
36 add EP2 list_offer
37 add Manager list_offer
```

The model assumes that we first have to declare and instantiate all participating services using REMES interface operations. We model one energy consumer, two energy providers and one mediator that represents the interests of all negotiation participants as shown in Table 1 (lines 00-28). However, we have to point out that in case it would be needed to model more than one energy consumer, and more than two energy providers, the described model would be able to support it. For each negotiation participant we have provided a list of service attributes, including their pre-, and postcondtions.

Next, to model the composition of services we need to create the lists (lines 29-32 in Table 1) and add the services to the appropriate lists (see Table 1 lines 33-37). In our approach we model service negotiation as a service composition via the parallel with synchronization protocol modeled by the operator $\|_{SYNC\text{-}and}$. Services that communicate via $\|_{SYNC\text{-}and}$ operator belong to the special type of REMES mode, called AND mode. By the semantics of the AND mode, the services connected by this operator start and finish their execution simultaneously.

Finally, our model of service negotiation is defined by the following:

```
bool contract := false;
clock h := 0;
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($param_c$)
OD (c_request < p_offer) $\wedge$ h $\leq$ 24)
  contract := true;
```

Requirements req_{client} and $req_{provider}$ are predicates that include both functional and extra-functional properties of services. In our case, req_{client} defines the client's request on amount of energy, price per unit of energy (ppue), and expected energy reliability (eng_rel). On the other hand, $req_{provider}$ encompasses properties of the service that is offered by a provider. Let us assume scenario 1, as described in Section 3 and a negotiation that takes place at 18 o'clock of the day (h == 18). At this specific time req_{client} is described as: h == 18 $\wedge$ energy_amount == 14 $\wedge$ req_ppue == 15,7 $\wedge$ eng_rel == 0,8. The provider's offer for a given request is: h == 18 $\wedge$ energy_amount == 14 $\wedge$ offered_ppue == 20 $\wedge$ eng_rel == 0,8. Generally, 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, or time at which a service should be available. As soon as the requirements are known, the negotiation can start. 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. [20]. In case that the provided offer has not met the client's expectation, the request (c_request) can be updated using the same function negotiation ($param_c$) but with a different parameter (here, we abstract from the function details). The negotiation process may continue as long as the participants are interested into reaching an agreement, or in case that the negotiation model is time constrained, as long as time allows. In our case, the model is time constrained, and the negotiation will continue as long as an energy supply over a day is not satisfied. The outcome of the negotiation can either be a contract (c_ request $\geq$ p_offer) or no contract (c_ request < p_offer). In our model, the contract will be signed only if the client agrees with the offered energy amount, the price per unit of energy, and the provided energy reliability. In the example presented above (scenario 1 and energy negotiation at 18 o'clock), one can notice that the requested and offered price differ and that the negotiation is needed. If we assume the same example, then the negotiation successfully finishes with a contract signed and the final agreement of the form: $\mathsf{h} == 18 \wedge \mathsf{energy\_amount} == 14 \wedge \mathsf{final\_ppue} == 16 \wedge \mathsf{eng\_rel} == 0{,}8$.

The REMES language is accompanied with a tool support for constructing models as one described above in a graphical form [21]. In the following section, we show a formal analysis of the described REMES negotiation model in order to check whether the available amount of energy suffices for the client's needs and at what prices the negotiation converges. Furthermore we analyze the utility-optimal functions w.r.t. the price and the energy reliability (a weighted sum of the price and the energy reliability as the negotiation preferences).

5 Formal Analysis of the Negotiation Model

5.1 The Analysis Goals

In this paper we consider a model that supports a competition between two energy providers, available for negotiation via a mediator that acts as representative of all parties involved in the negotiation process. We model the described negotiation model using out textual composition language HDCL, and then analyze the model against several requirements, such as price, time, and reliability, in order to check whether the available energy and given prices can satisfy the client's needs. Also, it is interesting to see how much time is needed for agreement to converge.

Additionally, we calculate the value of the optimal utility function as a weighted sum of negotiation preferences w.r.t. the price and the energy reliability (modeled here by a number), and model-check the trace (a sequence of actions (delays and transitions)) that leads to such state. We calculate the value of the optimal utility function in order to find points in time when the utility function is maximized. We assume the utility function to be maximized for all participants, when the difference between their initial and their final utility values either do not exist or is insignificant.

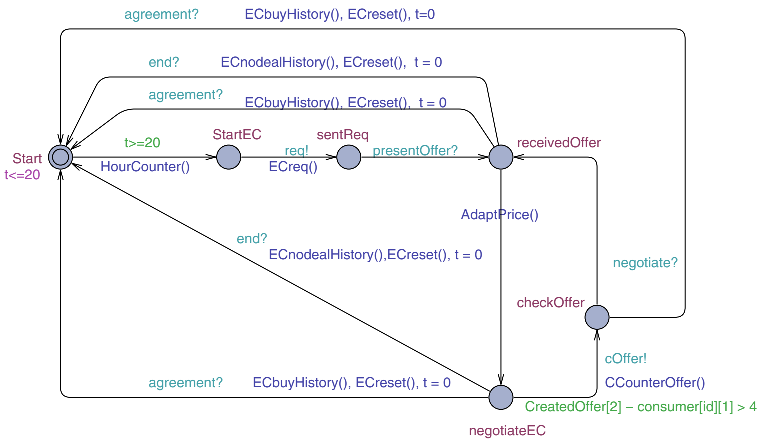
In order to ensure that our model has no deadlocks, we specify a safety UPPAAL property as follows: AG not deadlock. The given property has been verified in UPPAAL and our model satisfies it. All findings presented in the following are results of model-checking the described model in UPPAAL.

5.2 A TA Semantic Translation of the REMES Model and Analysis Results

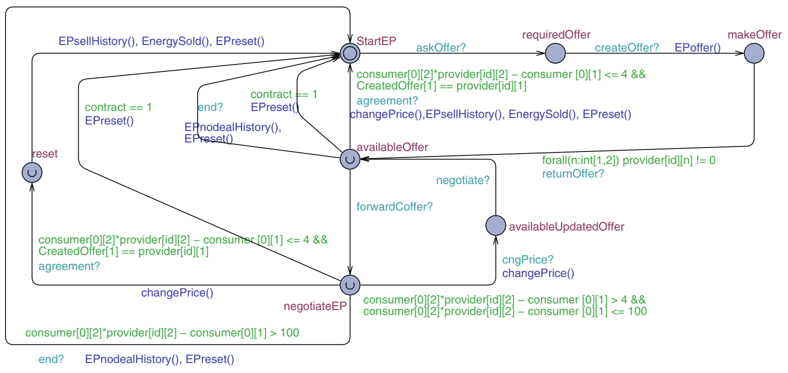
We have analyzed the REMES-based energy negotiation model, by semantically translating it into a network of TA models, in the UPPAAL [2] model-checker. The model contains five TA connected in parallel: EnergyConsumer, EnergyProvider (used to create two providers as instances of this TA), Mediator, EnergyProduction1, and EnergyProduction2. Due to space limitation, we present here the TA models of EnergyConsumer, EnergyProvider, and Mediator, shown in Fig. 2.

The TA of EnergyConsumer has six locations: Start, StartEC, sentReq, receivedOffer, negotiateEC, and checkOffer. A negotiation request is sent every 20 time units

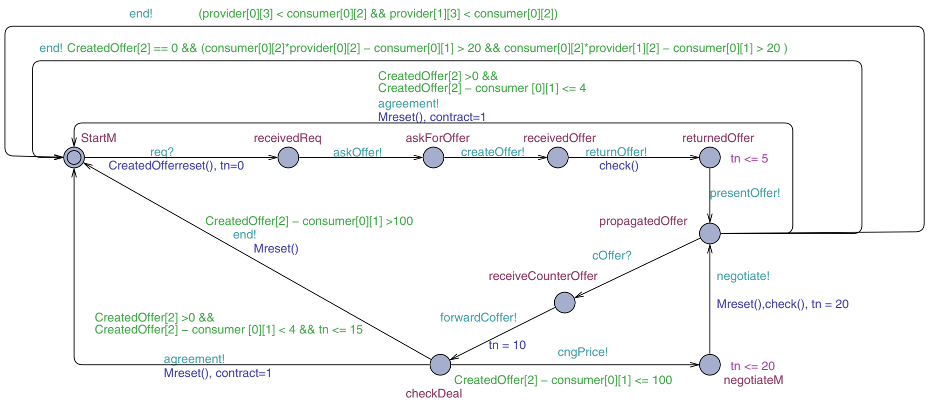
[2] See the web page www.uppaal.org for more information about the UPPAAL tool.


agreement?
ECbuyHistory(), ECreset(), t=0
end?
ECnodealHistory(), ECreset(), t = 0
agreement?
ECbuyHistory(), ECreset(), t = 0
Start
t<=20
t>=20
HourCounter()
StartEC
req!
ECreq()
sentReq
presentOffer?
receivedOffer
AdaptPrice()
end?
ECnodealHistory(),ECreset(), t = 0
negotiate?
checkOffer
cOffer!
CCounterOffer()
agreement?
ECbuyHistory(), ECreset(), t = 0
CreatedOffer[2] – consumer[id][1] > 4
negotiateEC

(a) A TAn of the client


EPsellHistory(), EnergySold(), EPreset()
StartEP
askOffer?
requiredOffer
createOffer?
EPoffer()
makeOffer
consumer[0][2]*provider[id][2] – consumer [0][1] <= 4 &&
CreatedOffer[1] == provider[id][1]
agreement?
changePrice(),EPsellHistory(), EnergySold(), EPreset()
contract == 1
EPreset()
end?
EPnodealHistory(),
EPreset()
contract == 1
EPreset()
reset
availableOffer
forall(n:int[1,2]) provider[id][n] != 0
returnOffer?
negotiate?
forwardCoffer?
availableUpdatedOffer
consumer[0][2]*provider[id][2] – consumer [0][1] <= 4 &&
CreatedOffer[1] == provider[id][1]
agreement?
cngPrice?
changePrice()
changePrice()
negotiateEP
consumer[0][2]*provider[id][2] – consumer [0][1] > 4 &&
consumer[0][2]*provider[id][2] – consumer [0][1] <= 100
consumer[0][2]*provider[id][2] – consumer[0][1] > 100
end?
EPnodealHistory(), EPreset()

(b) A TAn of the provider


end!
(provider[0][3] < consumer[0][2] && provider[1][3] < consumer[0][2])
end! CreatedOffer[2] == 0 && (consumer[0][2]*provider[0][2] – consumer[0][1] > 20 && consumer[0][2]*provider[1][2] – consumer[0][1] > 20)
CreatedOffer[2] >0 &&
CreatedOffer[2] – consumer [0][1] <= 4
agreement!
Mreset(), contract=1
StartM
req?
CreatedOfferreset(), tn=0
receivedReq
askOffer!
askForOffer
createOffer!
receivedOffer
returnOffer!
check()
returnedOffer
tn <= 5
presentOffer!
CreatedOffer[2] – consumer[0][1] >100
end!
Mreset()
propagatedOffer
cOffer?
receiveCounterOffer
negotiate!
Mreset(),check(), tn = 20
CreatedOffer[2] >0 &&
CreatedOffer[2] – consumer [0][1] < 4 && tn <= 15
forwardCoffer!
tn = 10
cngPrice!
agreement!
Mreset(), contract=1
tn <= 20
negotiateM
CreatedOffer[2] – consumer[0][1] <= 100
checkDeal

(c) A TAn of the negotiation mediator

Fig. 2. TA models of the negotiation participants

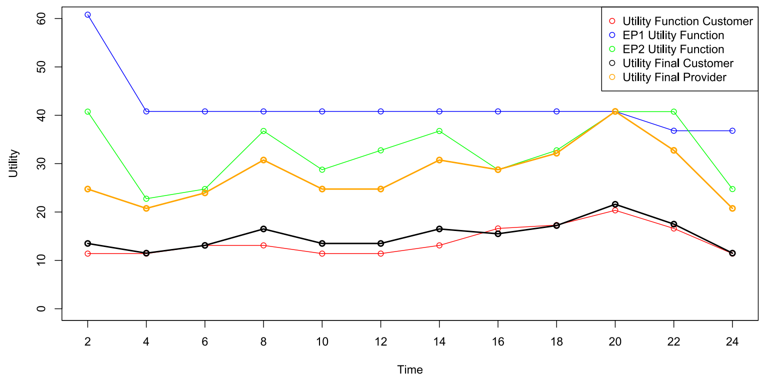


Fig. 3. Utility function change over a day for scenario 2

(t == 20), corresponding to every two hours as described in the model. Sending a request for an offer to Mediator, and receiving an offer from Mediator is modeled with channels req, and presentOffer, respectively. In case that participants need to negotiate on the current request and offer, they communicate via the broadcast channel negotiate. When an agreement is made a boolean variable contract is set to 1 and it is propagated via channel agreement, while on the other hand, in cases when no agreement has been reached (contract == 0) the channel end is used.

The TA of EnergyProvider consists of seven locations: StartEP, requiredOffer, makeOffer, availableOffer, availableUpdatedOffer, negotiateEP, and reset. A request for an offer is received from Mediator via channel askOffer. To create an offer and to further on propagate it to Mediator channels, createOffer, and returnOffer are used, respectively. In case that a request has been updated a counter offer is propagated via forwardCoffer and cngPrice broadcast channels.

The TA of Mediator has nine locations: StartM, receivedReq, askForOffer, receivedOffer, returnedOffer, propagatedOffer, receivedCounterOffer, checkDeal, and negotiateM. The automaton contains a clock variable tn, used to keep track of the time elapsed from the moment a request is received to the moment an agreement or no agreement has been signed.

In our analysis model, we encode the utility function for the consumer, and the providers, respectively, as a weighted sum of negotiation preferences (i.e., price per unit of energy and reliability given as a number and not probability), as follows:

$$\begin{aligned} utility_c &= wc_1 \times req\_ppue + wc_2 \times eng\_rel \\ utility_p &= wp_1 \times offered\_ppue + wp_2 \times eng\_rel \end{aligned} \tag{2}$$

The function is calculated for the energy consumer, both energy providers, taking into consideration the starting request/offer and the final agreement given that they have different priorities for different preferences. In case of the energy consumer reliability gets higher priority (wc_2), while in the case of the energy providers the energy price is more important (wp_1). In our case study, we consider the utility functions as described

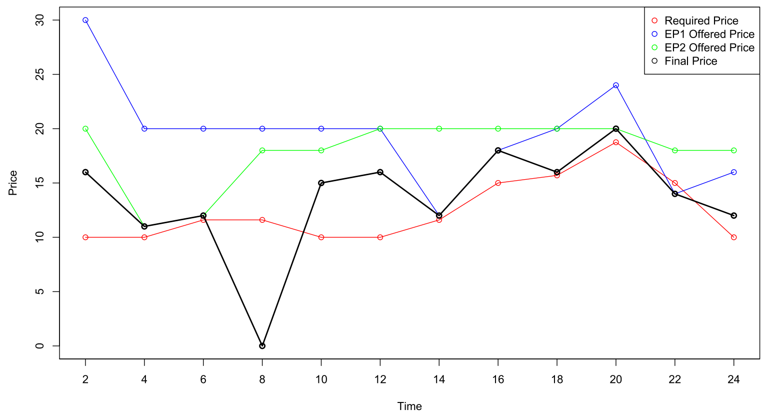


Fig. 4. Price per unit of energy for scenario 1

in Eq. 2, where wc_1, wc_2, and wp_1, wp_2 are client and provider's preferences on price and energy reliability, respectively. In the following, we present and discuss the results of the model verification in UPPAAL model-checker.

Verification shows that in scenario 1, there exists a case in which no agreement has been reached (8 o'clock), since the initially requested and offered prices were too far from each other, and since the customer had an upper bound on the price. In the same scenario, as shown in Fig. 4, in order to provide sufficient energy supply, the client is forced to spend slightly more money that initially planed, but still within the maximum price bound.

Fig. 3 depicts the utility function change over a day assuming scenario 2. Based on the history of previous request, 18 o'clock is considered as the peak hour in consumer's energy consumption. At this point in time, the utility function is maximized for each negotiation participant, respectively (the difference between initial and the final utility value either does not exist or is insignificant), meaning that the energy market favors them equally. Consequently, the consumer is prepared to request a reasonable price to make sure that he gets a required amount of energy. On the other side, the provider's offered price depends on the amount of the available energy, that is, the more energy is available the price is lower and vice versa. This means that the providers are ready for the peak hour, and have stored greater amount of energy such that they are competitive enough at the energy market. At 16 o'clock, the provider's initial and the final utility is similar, while the customer's final utility value is slightly lower than the initial value since the final price is lower than the one requested by the customer. At 20 o'clock the same situation appears, but in favor of the energy provider.

In scenario 3 we have expected that in total the client would spend more money on energy due to the fact that he was adapting his requests based on the offered prices. However, the total price is relatively close to the expected one, probably due to the mediator selecting offers on behalf of the client, which leads to the client only receiving the cheapest offers in the market, in each round. Also, the time spent to negotiate the energy supply was expected to be lower than in the other two scenarios, but it shows

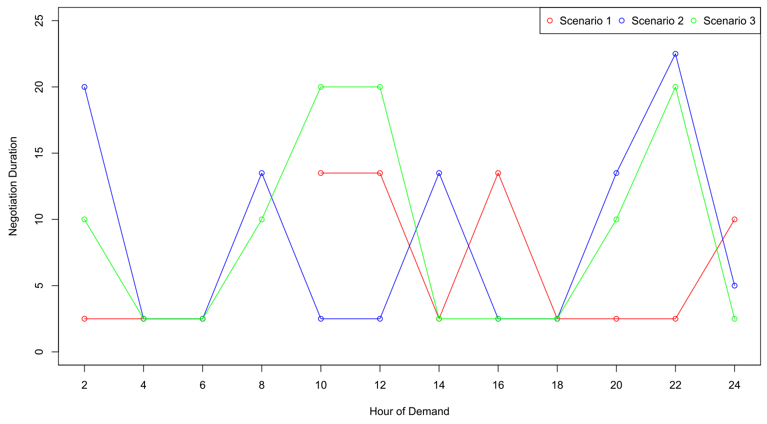


Fig. 5. Time required for negotiation

that this scenario was the most time consuming, probably due to the fact that the consumer has to adapt his acceptance threshold all over again, but still to keep within the maximum available budget (see Fig. 5). At the same time, in each negotiation round, the mediator has to check all available offers, in order to provide the client with the cheapest and most fitting one.

It was very interesting to see who owns the market in which scenario. Based on Fig. 4, in scenario 1 it is obvious that the market is own by the provider, and that even with the introduced maximum price bound, the providers were able to force the prices in their favor. Similarly, in the other two scenarios we have noticed that in scenario 3, the agreed prices are in favor of the consumer, while the final prices in scenario 2 are in favor of the energy provider. Overall, the total amount of money spent on energy in all three scenarios is very close to the initial request, with an average increase of less than 10% of the initially requested price. Before verifying the time needed in negotiation, we expected that the participants would converge toward the agreement the fastest in scenario 3. However, the results have shown the opposite, the slowest negotiation process was recorded in this scenario, possibly due to the fact that the client needed to recalculate new prices compared to the previous offers, and based on this the mediator had to ask for the new offers, always from all providers. One can notice that the least time is needed in scenario 1, while scenario 2 requires slightly more time.

6 Related Work

Mobach describes a negotiation framework based on the WS-Agreement specification [5], deployed in domains of distributed agent middleware and distributed energy management. The latter case has been simulated and evaluated through the different strategies in which energy has been distributed to the clients, including negotiation and bidding for a suitable energy source. The simulation has provided better insight in different negotiation policies, however the model lacks constructs for the formal analysis and means to provide performance analysis results. Lapadula et al. provide a description of modeling publication, discovery, negotiation, deployment, and execution of service-oriented applications in COWS [22], a language that can be translated to CMC model-checker for analysis purposes. In comparison to this approach, our framework includes analysis that caters for more than one QoS attribute (performance and reliability), while assuming time in the process too. Capodieci et al. propose an agent-based approach to model and analyze deregulated energy market [23]. In their work they adapt minority game approach that enables better distribution of the available resources. The simulation of the time flow and risk variations is done using stochastic game design. The resulting model has been simulated using JADE platform. Compared to our work the presented approach is equally fit for the modeling issues, but it lacks a possibility to exhaustively analyze the given model that could uncover more information on the issues that they describe.

7 Conclusions

In this paper, we present a case study where our recently introduced approach for automated service negotiation in REMES has been applied to model and analyze distributed energy management. The given study has been analyzed by semantically translating the REMES-based models into a network of TA to enable model-checking in the UPPAAL tool. We have focused on three scenarios as described in Section 3 by calculating the value of the optimal utility function w.r.t. the price and the energy reliability and model-checked the model to compute the traces that lead to such states. The negotiation model is time constrained, which lets one get an insight into the analysis of the time needed to reach an agreement. As future work we plan to model an auction-based energy management which would show the full potential of our negotiation model.

References

[1] Tamma, V., Wooldridge, M., Blacoe, I., Dickinson, I.: An ontology based approach to automated negotiation. In: Padget, J., Shehory, O., Parkes, D.C., Sadeh, N.M., Walsh, W.E. (eds.) AMEC 2002. LNCS (LNAI), vol. 2531, pp. 219–237. Springer, Heidelberg (2002)

[2] Resinas, M., Fernandez, P., Corchuelo, R.: A conceptual framework for automated negotiation systems. In: Corchado, E., Yin, H., Botti, V., Fyfe, C. (eds.) IDEAL 2006. LNCS, vol. 4224, pp. 1250–1258. Springer, Heidelberg (2006)

[3] Paurobally, S., Tamma, V.A.M., Wooldridge, M.: A framework for web service negotiation. TAAS 2(4) (2007)

[4] Mu-kun, C., Chi, R., Liu, Y.: Service oriented automated negotiation system architecture. In: 6th International Conference on Service Systems and Service Management, ICSSSM 2009, pp. 216–221 (2009)

[5] Mobach, D.: Agent-Based Mediated Service Negotiation. PhD thesis, Vrije University (2007)

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

[7] Alur, R., Dill, D.L.: A theory of timed automata. Theoretical Computer Science 126(2), 183–235 (1994)

[8] Bengtsson, J., Yi, W.: Timed automata: Semantics, algorithms and tools. In: Desel, J., Reisig, W., Rozenberg, G. (eds.) ACPN 2003. LNCS, vol. 3098, pp. 87–124. Springer, Heidelberg (2004)

[9] Čaušević, A., Seceleanu, C., Pettersson, P.: Modeling and reasoning about service behaviors and their compositions. In: Margaria, T., Steffen, B. (eds.) ISoLA 2010, Part II. LNCS, vol. 6416, pp. 82–96. Springer, Heidelberg (2010)

[10] Ivanov, D., Orlic, M., Seceleanu, C., Vulgarakis, A.: Remes tool-chain - 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)

[11] Enoiu, E.P., Marinescu, R., Causevic, A., Seceleanu, C.: 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)

[12] Alur, R., La Torre, S., Pappas, G.J.: Optimal paths in weighted timed automata. In: Di Benedetto, M.D., Sangiovanni-Vincentelli, A.L. (eds.) HSCC 2001. LNCS, vol. 2034, pp. 49–62. Springer, Heidelberg (2001)

[13] Causevic, A., Seceleanu, C., Pettersson, P.: An analyzable model of automated service negotiation. In: IEEE SOSE 2013: 7th International Symposium on Service Oriented System Engineering. IEEE (March 2013)

[14] Čaušević, A., Seceleanu, C., Pettersson, P.: Checking correctness of services modeled as priced timed automata. In: Margaria, T., Steffen, B. (eds.) ISoLA 2012, Part II. LNCS, vol. 7610, pp. 308–322. Springer, Heidelberg (2012)

[15] Bengtsson, J., Griffioen, W.D., Kristoffersen, K.J., Larsen, K.G., Larsson, F., Pettersson, P., Yi, W.: Automated verification of an audio-control protocol using uppaal. The Journal of Logic and Algebraic Programming, 163–181 (2002)

[16] Bengtsson, J., Griffioen, W., Kristoffersen, K., Larsen, K., Larsson, F., Pettersson, P., Yi, W.: Verification of an audio protocol with bus collision using UPPAAL. In: Alur, R., Henzinger, T.A. (eds.) CAV 1996. LNCS, vol. 1102, pp. 244–256. Springer, Heidelberg (1996)

[17] Bengtsson, J., Larsen, K., Larsson, F., Pettersson, P., Yi, W.: UPPAAL - a tool suite for automatic verification of real-time systems. In: Alur, R., Sontag, E.D., Henzinger, T.A. (eds.) HS 1995. LNCS, vol. 1066, pp. 232–243. Springer, Heidelberg (1996)

[18] Alur, R., Courcoubetis, C., Dill, D.: Model-checking for real-time systems. In: Proceedings of the Fifth Annual IEEE Symposium on Logic in Computer Science, LICS 1990, pp. 414–425 (June 1990)

[19] Alur, R., Courcoubetis, C., Dill, D.L.: Model-checking in dense real-time. Inf. Comput. 104(1), 2–34 (1993)

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

[21] Enoiu, E.P., Marinescu, R., Causevic, A., Seceleanu, C.: A design tool for service-oriented systems. In: Proceedings the 9th International Workshop on Formal Engineering approaches to Software Components and Architectures (FESCA). Electronic Notes in Theoretical Computer Science (ENCTS), vol. 295, p. 95. Elsevier (May 2013)

[22] Lapadula, A., Pugliese, R., Tiezzi, F.: Service discovery and negotiation with cows. Electron. Notes Theor. Comput. Sci. 200, 133–154 (2008)

[23] Capodieci, N., Cabri, G., Pagani, G.A., Aiello, M.: An agent-based application to enable deregulated energy markets. In: 2012 IEEE 36th Annual Computer Software and Applications Conference, pp. 638–647 (2012)