The Energy Management Adviser at EDF

Pierre Chaussecourte[3], Birte Glimm[2], Ian Horrocks[1], Boris Motik[1], and Laurent Pierre[3]

[1] University of Oxford, Oxford, UK
[2] University of Ulm, Ulm, Germany
[3] Electricité De France R&D, Clamart, France

Abstract. The EMA (Energy Management Adviser) aims to produce personalised energy saving advice for EDF's customers. The advice takes the form of one or more 'tips', and personalisation is achieved using semantic technologies: customers are described using RDF, an OWL ontology provides a conceptual model of the relevant domain (housing, environment, and so on) and the different kinds of tips, and SPARQL query answering is used to identify relevant tips. The current prototype provides tips to more than 300,000 EDF customers in France at least twice a year. The main challenges for our future work include providing a timely service for all of the 35 million EDF customers in France, simplifying the system's maintenance, and providing new ways for interacting with customers such as via a Web site.

1 Introduction

The EMA (Energy Management Adviser) has been under development at Electricité De France (EDF) R&D for several years. It aims to produce personalised energy saving advice for individual customers. This advice is in the form of one or more 'tips' that depend on customer specific factors, such as housing and electricity consumption, as well as environmental factors, such as weather conditions during the relevant period. For example, a customer whose energy consumption during the summer is higher than was forecast, whose home is air conditioned, and who lives in a region where, during the relevant period, the weather was warmer than expected, might be advised that their increased consumption was probably caused by increased use of air conditioning due to the warmer weather; furthermore, the customer might be offered advice for reducing their reliance on air conditioning.

EMA is implemented as a web service that is passed a set of parameters describing customer circumstances, and that returns suitable advice text. Thus, other systems at EDF can use EMA to obtain energy saving advice for customers as needed. Currently, EMA is primarily used by the EDF billing system in order to provide energy saving advice on customers' bills, but it is envisaged that the service will be used more extensively in the future, such as for improving customer experiences when interacting with EDF's web site.

The production of such personalised customer tips is achieved using semantic technologies: facts about customers are represented using RDF triples (equivalently, OWL assertions), OWL ontologies are used to provide a conceptual model of the relevant domain (housing, environment, and so on) and different kinds of tips, and SPARQL queries are used to identify relevant tips. For example, given OWL assertions describing the circumstances of the above mentioned customer (whose energy consumption during the summer is higher than was forecast), an OWL reasoner is used to answer a SPARQL query that retrieves classes that represent relevant tips, and one of these classes is associated with the appropriate advice about air conditioning. In contrast, if the customer's circumstances were described using a set of assertions stating that the weather was cooler than expected, then the query would retrieve different classes, and the system would produce different advice. In general, advice may identify causes of increased consumption such as changes in living habits, increased use of electrical devices, or keeping devices on standby.

Basing the system on semantic technologies has many advantages: OWL provides a rich, flexible, and fully declarative language for modelling customers and customer environments; infrastructure such as ontology editing and reasoning tools is readily available for OWL, and it can be used to support ontology development as well as for identifying relevant tips; and the behaviour of EMA can be adapted, extended and maintained without coding. By using SPARQL queries to identify relevant tips, it is even possible to provide a useful form of nonmonotonic behaviour, where otherwise relevant tips can be 'cancelled' by the existence of some special client circumstances, such as age or disability (see Section 3.4). Finally, future enhancements of the system could involve storing customer data directly in highly scalable RDF stores.

The latest version of the ontology and the reasoning system is the result of an ongoing collaboration between EDF and the University of Oxford that started in 2011. The current version of EMA provides tips to more than 300,000 EDF customers in France at least twice a year. The main challenges for our future work include providing a timely service for all of the 35 million EDF customers in France, simplifying the system's maintenance, and providing new ways for interacting with customers such as via a Web site.

In the rest of the paper we assume basic familiarity with RDF, OWL, and SPARQL; readers are referred to [8, 5, 1] for suitable primers.

2 EMA Architecture

Customer information is stored in various databases that are independent of the EMA web service. Therefore, to use the EMA, a client must retrieve the relevant information, invoke the EMA web service—passing the information as parameters—and then receive the relevant tips from the service (see Fig. 1).

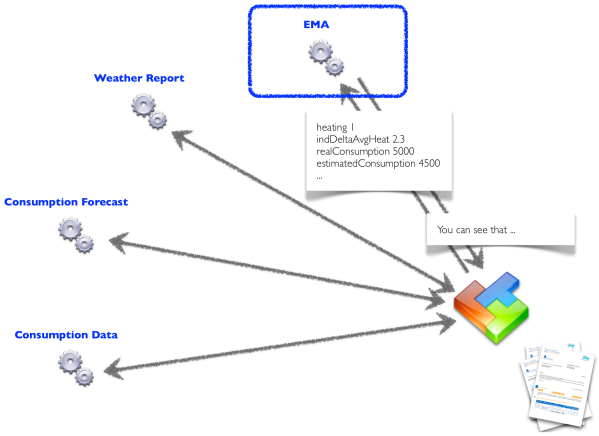

Fig. 1. EMA Web Service Architecture

2.1 Describing Customer Circumstances

Within EMA, the job of determining which tips are appropriate to a given customer is delegated to an ontology reasoner; the current implementation uses the Hermit[4] reasoner developed at the University of Oxford. In an earlier prototype of EMA, a new reasoning process was launched for each web service call. This, however, turned out to be very inefficient as it required reloading all the relevant ontologies in addition to reasoning about the given customer's circumstances. In order to improve the performance, we have extended HermiT to support incremental reasoning and batch processing of customers (see Section 4).

In order to use ontology reasoning, EMA describes the relevant circumstances of each customer using OWL assertions. For brevity, in this paper we use the Manchester syntax [7], which is also supported in the ontology editor Protégé; however, one could equivalently write down all of our examples using RDF triples. A given customer and the home that they live in are modelled using assertions of the following form:

```
Individual: a_client Types: ClientSituation
Individual: a_house Types: Home
Individual: a_client Facts: livesIn a_house
```

These assertions state that a_client is customer situation, a_house is a home, and a_client lives in a_house. Please note that we use 'client' and 'customer' interchangeably; moreover, please note that we actually model customer situations, rather than customers. In order to identify relevant tips, the individual that represents a customer situation (a_client in this case) is also connected via

[4] http://www.hermit-reasoner.com/

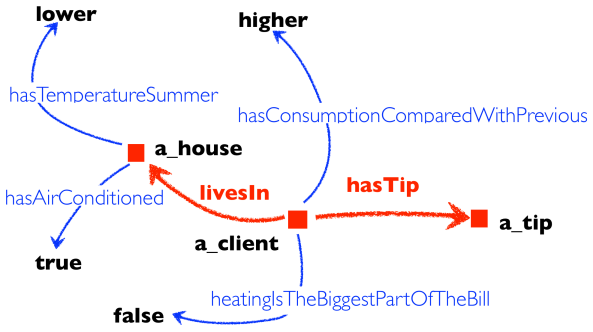

Fig. 2. Assertions describing a customer's situation

the hasTip property to an individual that is an instance of the Tip class; this individual will be used to determine which tips are applicable to a_client. The resulting assertions are shown graphically in Fig. 2.

Customers and their homes are described using various parameters passed to the EMA service; for example,

- the `indAC` parameter is set to one if the customer's home is equipped with an air conditioning system, and zero otherwise;
- the `consElecAC` parameter captures the estimated quantity of energy used for air conditioning; and
- the `consElecHeating` parameter captures the estimated quantity of energy used for heating.

Some of these parameters can be transformed directly into assertions and added to the model. For example, the presence of air conditioning is represented using the object property hasAirConditioning, and so the `indAC` parameter with value one can be simply transformed into the following assertion:

`Individual:` a_house `Facts:` hasAirConditioning true

Note that, in the above assertion, true is an individual and not a data value; we will discuss this in more detail in Section 3.2.

Other parameters may require more complex transformations. For example, the object property consACvHeating is used to represent the relative amounts of energy consumed for air conditioning and heating; for example, assertion

`Individual:` a_client `Facts:` consACvHeating Higher

represents the fact that the customer consumed more energy for air conditioning than for heating. This assertion, however, does not correspond to a single parameter passed to the system; rather, it is produced through a numerical comparison of the values of the `consElecAC` and `consElecHeating` parameters.

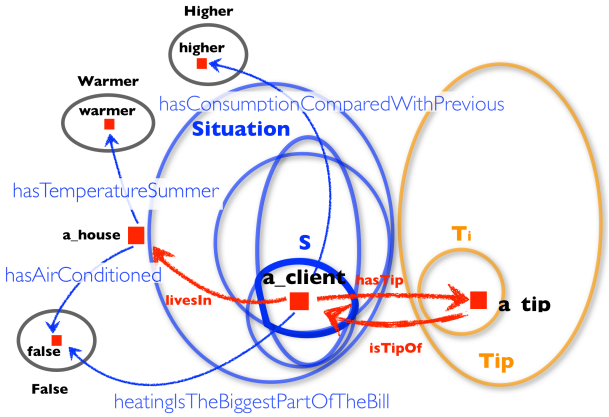

Fig. 3. Finding suitable tips

2.2 Identifying Appropriate Tips

Once a customer's situation has been fully described using OWL assertions, an OWL reasoner can be used to classify the a_client individual as an instance of one or more of the customer situation classes described in the ontology. For example, a_client might be recognised as an instance of the situation class S, a class of customers whose OWL definition includes the existence of a home with air conditioning and so on. In addition, the a_tip individual will be recognised as an instance of a class T_i that represents tips relevant to situation S (see Fig. 3). Relevant tips can then be identified by simply retrieving the classes that the individual a_tip is an instance of; this can be achieved using the following SPARQL query:

```
SELECT ?t WHERE { a_tip rdf:type ?t  . ?t rdfs:subClassOf ActualTip . }
```

2.3 Retrieving Textual Tips

The result of the SPARQL query provides tips in the form of class names. To complete its task, the EMA service transforms these names into text that can be presented to the customer. Suitable text is associated with each of the tip classes, and it may include special strings that are replaced with values from the input parameters. The following is an example of a tip text:

> Electricity consumption between %1 and %2 is higher than during the same period last year. This increase may be caused by a higher outdoor average temperature (%3 degrees higher), which could have led to increased use of your air conditioning system.

Strings %1, %2, and %3 are replaced by parameter values to provide the final text. Thus, the above tip text would be converted into the following finished tip:

> Electricity consumption between **June 1st** and **September 30th** is higher than during the same period last year. This increase may be caused by a higher outdoor average temperature (**2** degrees higher), which could have led to increased use of your air conditioning system.

3 Ontology Modelling

The EMA service uses OWL ontologies to model customer situations and their relationships to relevant energy saving tips. An example of a customer situation is 'living in the North of France in a house with electric heating, and using more electricity than in the same period in the preceding year', and this situation might be associated with the tip 'check the temperature on the thermostat, and consider reducing it by a couple of degrees'.

The system uses several ontologies in order to improve modularity and facilitate maintenance by a team of developers with different areas of expertise. In particular, the *core* and *bases* ontologies define generic classes and properties used to describe more complex situations; the *ACS* ontology describes situations relevant to customers for which EDF have a record of previous energy consumption; the *conseils* (advice) ontology captures generic situations that require relatively little information about the customer; and the *TPN* ontology describes 'special situations' in which the standard tips may not be appropriate.[5] Examples of special situations include those arising from customer disabilities or other special needs. The *ACS* ontology imports the *core*, *bases*, *conseils*, and *TPN* ontologies. The ontologies use features such as disjunction (`or`), all values from restrictions (`only`), and inverse properties (`InverseOf`), and hence do not conform to any of the OWL 2 profiles.

3.1 Customer Situation Ontologies

Home management specialists are responsible for providing advice relevant to different customer contexts. These specialists use decision trees to analyse different situations that a customer might be in, with each situation being distinguished by a range of features such as whether the customer owns their own home, what kind of home they live in (house, apartment, and so on), and whether they are a new customer. Home management specialists then associate tips with some or all of the nodes in the decision trees. The original design goal for EMA was to ensure that the system's ontologies closely mimic the decision tree structure, with the idea that this would allow for a rapid 'bootstrapping' of the ontologies and reduce the chance of introducing errors during the conversion of decision trees into ontologies. The resulting OWL modelling is sometimes rather stilted, but the ontology is under constant revision to address these issues and to extend the capabilities of the system.

[5] The *TPN* ontology is currently only a prototype and is being used in experiments whose goal is to extend EMA so that it can deal with special situations; we discuss the surrounding issues in more detail in Section 3.4.

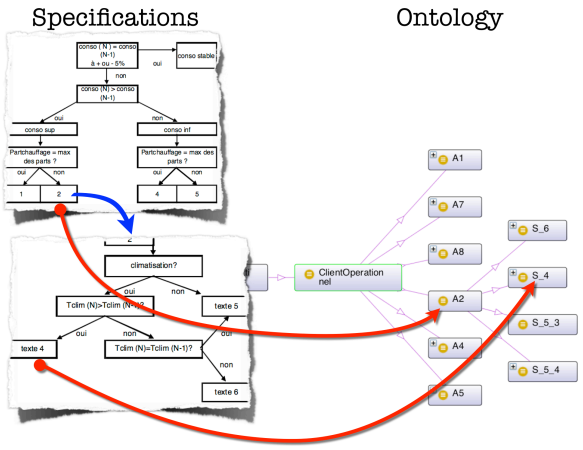

Fig. 4. From Paper Specifications to Ontology

Fig. 4 shows fragments of such decision trees and the mapping from decision tree nodes into ontology classes. Although the structures are similar, decision tree nodes are not in one-to-one correspondence with ontology classes. This is because it was sometimes convenient to collapse multiple nodes in the tree into one class expression, to introduce 'intermediate' classes so as to avoid repetition in class expressions, and to introduce 'structural' classes that group together related classes and improve the structure of the ontology hierarchy.

For example, S_4 is a 'real' class (i.e., a class that corresponds to a node in a decision tree), while class A2 is an intermediate class (it captures a part of the class expression used in the description of S_4). These classes are described using the following axioms:

```
Class: A2 EquivalentTo:
        heatingIsTheBiggestPartOfTheBill some False and
        hasConsumptionCompareWithPrevious some Higher

Class: S_4 EquivalentTo:
        A2 and
        hasAirConditioning some True and
        livesIn some (hasTemperatureSummer some Warmer)
```

Class A2 describes an intermediate situation in which electricity consumption is higher than in the corresponding period of the previous year, and with heating not making up the largest part of the bill. Class S_4 then refines A2 by additionally requiring that the client owns an air conditioning system and lives in a house where the summer temperature was warmer than in the previous year.

The result of such a design is that customer situations are described in a tree-like hierarchy of classes, the root of which is the CustomerSituation class.

Directly under CustomerSituation there are two classes, StandardSituation and SpecialSituation, that split the situation hierarchy into two parts. The former part (under StandardSituation) describes standard customer situations (such as their home environment), while the latter part (under SpecialSituation) describes customers with special needs (such as disabilities). Note that these two classification hierarchies are not disjoint: a given customer situation could be classified both as a standard situation (e.g., living in a home with air conditioning) and a special situation (e.g., being disabled). We will say more about special situations in Section 3.4.

3.2 Avoiding Value Class Expressions

In the StandardSituation hierarchy, classes are typically defined as a conjunction of a more general situation class and a number of existential restrictions. In some cases, these restrictions are simple attribute-value pairs, with the restriction class being True, False, Equal, Higher, or Lower. In other cases, the restriction class is itself a class description, often describing features of a customer's home.

Earlier versions of the EMA ontologies used `value` restrictions (also known as *nominals*) to encode such class descriptions. For example, the axiom

`Class:` S_2 `EquivalentTo:` S_1 `and` livesIn `some` (electricHeating `value` true)

was used to describe a situation S_2 as a refinement of S_1 with the additional feature that the client's home is equipped with electric heating. Please note that true in the above axiom is an individual used in a `value` restriction. Such a modelling style, however, was found to be problematical: whenever the assertions describing a customer were updated, the ontology hierarchy had to be recomputed as well due to the usage of nominals in class descriptions, which eventually proved to be a severe performance bottleneck.

In order to avoid these problems, we adopted a different modelling strategy that simulates nominals using fresh classes [3]. In particular, the above axiom is actually written in the EMA ontologies as

`Class:` S_2 `EquivalentTo:` S_1 `and` livesIn `some` (electricHeating `some` True)

where True is a 'simulated nominal class'. We also introduce an individual true and state it to be an instance of the True class. Then, given assertions

`Individual:` h1 `Facts:` electricHeating true

one can now recognise individual h1 as as an instance of the class

electricHeating `some` True

which can then be used to appropriately identify instances of class S_2. By eliminating nominals from the EMA ontologies, we ensure that the ontologies can be classified once upon the system startup, and not each time the customer situation is updated, which has considerably improved the system's performance.

3.3 Tips and GCIs

Textual tips are not stored directly in the ontology; rather, each tip is associated with a class in the ontology. Tip classes are arranged in a simple hierarchy the root of which is the Tip class; the hierarchy groups tips into various types (e.g., heating, air conditioning, summer), but without any further description.

Tip classes are associated with classes describing customer situations via the isTipOf property, which is the inverse of hasTip. For example, axiom

```
Class: isTipOf some S_1 SubClassOf: T1
```

states that each individual related to an instance of the S_1 customer situation class via the isTipOf property will be inferred to be an instance of the T1 tip class. Unfortunately, Protégé—the ontology editor used to develop and maintain the EMA ontologies—cannot represent such axioms since it does not allow for class expressions on the left hand side of axioms (a kind of axiom known in description logics as *general concept inclusions*, or *GCIs*).[6] To overcome this deficiency, EMA uses 'auxiliary tip recognition classes': each class expression used to recognise a tip is defined to be equivalent to a corresponding auxiliary tip recognition class, and this auxiliary class is then used on the left hand side of the relevant axiom. For example, the above axiom is transformed as follows:

```
Class: AT1 EquivalentTo: isTipOf some S_1
Class: AT1 SubClassOf: T1
```

Apart from working around the limitations of Protégé, this has the added advantage that it provides a separation between the situation ontology and the tip ontology, allowing different people to work on different parts of the ontology.

The auxiliary classes are grouped under AuxiliaryTipRecognitionClass and subdivided under its two subclasses AuxTip and AuxNotTip. Under the former (AuxTip) are classes whose definitions recognise client situations for which particular tips are appropriate; and under the latter (AuxNotTip) are classes whose definitions recognise client situations for which particular tips are inappropriate; we discuss the distinction between these two in more detail in Section 3.4.

3.4 Special Situations

The special situation hierarchy is used differently from the standard situation hierarchy in order to simulate a kind of nonmonotonic reasoning. This is achieved by using classification of customer situations in the special situations hierarchy to cancel tips that would otherwise be given as a result of classification in the standard situations hierarchy. For example, the tip 'check the temperature on the thermostat, and consider reducing it by a couple of degrees' may be considered inappropriate for elderly customers, for whom keeping warm is a priority. In this

[6] Note that the Manchester syntax also requires a class name as a subclass, so the mentioned statement is actually not syntactically correct.

case, when a customer is recognised as being elderly, for example by satisfying a data property restriction of the form

hasAge some integer[>= 75],

the system will 'cancel' the temperature reduction tip, even if the latter is applicable according to the standard situation hierarchy.

In practice, such a situation is modelled in the EDF's ontologies using axioms of the form

Class: NT1 EquivalentTo: notIsTipOf some S_2
Class: NT1 SubClassOf: T2

which state that each individual related to an instance of the S_2 customer situation class via the notIsTipOf property will be inferred to be an instance of the T2 tip class. In our example, S_2 would be the special situations class that represents elderly customers, and T2 would be the tip class that the 'check the temperature ...' tip is associated with. The modelling of individual customers is correspondingly extended by connecting the individual that represents the customer's situation (a_client in our running example) via the notHasTip property to an individual that is an instance of the Tip class (we will call this individual a_non_tip for the purposes of our example), where notHasTip is the inverse of notIsTipOf. The individual a_non_tip can then be used to retrieve those tips that are *not* applicable to a_client, using the SPARQL query

```
SELECT ?t WHERE { a_non_tip rdf:type ?t . ?t rdfs:subClassOf ActualTip . }
```

Although OWL does not support negation as failure in class descriptions, the use of negation as failure is possible in queries, as it amounts to subtracting the answer to one query (the above inapplicable tips query in our case) from the answer to another subquery (the applicable tips query from Section 2.2 in our case); this subtraction process has been formalised in the EQL-Lite query language [2]. This can be achieved directly in SPARQL using the following query:[7]

```
SELECT ?t WHERE {
    a_tip rdf:type ?t . ?t rdfs:subClassOf ActualTip .
    MINUS { a_non_tip rdf:type ?t . ?t rdfs:subClassOf ActualTip . }
}
```

Note that this procedure cancels tips only for customers *known* to be at least 75 years old, but not for customers whose age is unknown. Furthermore, one might expect that the problem could be solved by defining a standard situation class containing the `not hasAge some integer[>= 75]` restriction—that is, by using OWL's negation operator; however, this would not have the desired effect as the tip would be applicable only to customers *known* to be less than 75 years old.

[7] The `MINUS` keyword is new in SPARQL 1.1; in SPARQL 1.0 one can simulate negation as failure using a combination of `OPTIONAL` and `FILTER` keywords.

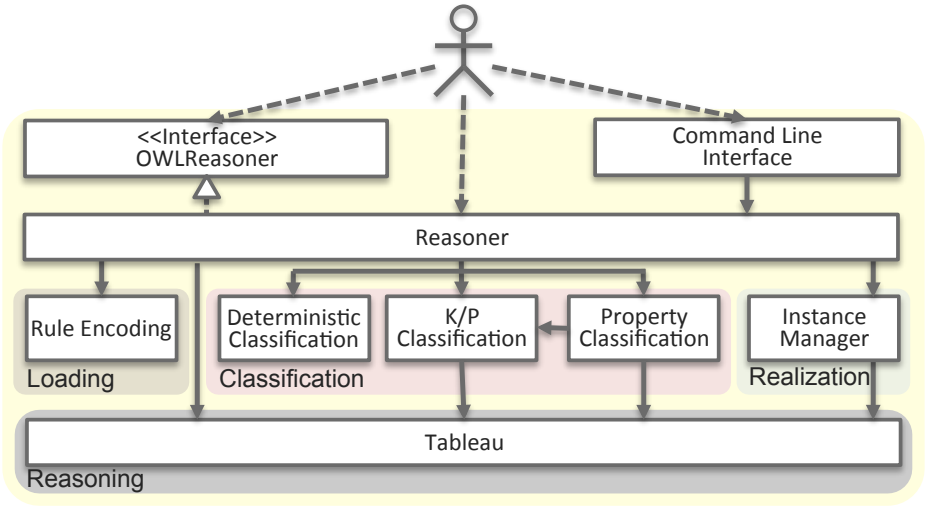

Fig. 5. A schematic system architecture of the HermiT reasoner

In other words, the `not` operator does not have the semantics needed to model exceptions and perform nonmonotonic reasoning, thus necessitating a solution such as the one outlined above. An in-depth discussion of these issues is given in [9].

4 The HermiT Reasoner

In this section, we present a brief overview over the open-source HermiT OWL reasoner, its reasoning algorithms, and relevant optimisations. HermiT is implemented in Java, so it can be used on a wide range of platforms. It comprises several modules that together realise a sound and complete OWL reasoning system. Figure 5 gives a high-level overview of the main system components. HermiT can be used via a command line interface, via its native Java interface (the `Reasoner` component), or via the `OWLReasoner` interface of the OWL API [6]. The EMA uses the `OWLReasoner` interface, thus allowing for easy substitution of different OWL reasoners. The main reasoning service in HermiT is checking whether an ontology is satisfiable; this functionality is realised by the `Tableau` component and its submodules. HermiT also supports many other reasoning tasks, all of which can be reduced to ontology satisfiability more or less straightforwardly, including ontology classification (i.e., the computation of the subClass and subProperty hierarchies) [4], and ontology realisation (i.e., the computation of all instances of all classes and properties). We next describe in more detail HermiT's reasoning algorithm and the realisation module since these are particularly relevant for EMA.

In order to check whether an ontology O is satisfiable, HermiT uses the hypertableau calculus [10], which first translates the TBox/schema axioms into

(derivation) rules (the `Rule Encoding` component in Figure 5). Then, starting with the explicitly stated assertions (e.g., the facts about a_client and a_house), by applying several derivation rules the reasoner tries to construct an abstraction of a model of O. Derivation rules usually add new class and property assertions and may introduce new individuals. For example, given axiom

`Individual:` h1 `Facts:` electricHeating `some` True

the reasoner introduces a new instance of the class True and connects h1 with this new individual via the electricHeating property in order to satisfy the `some` existential restrictions. The derivation rules used in HermiT can be nondeterministic (e.g., due to the use of disjunctive classes), and so the reasoner can often choose between several derivations. The rules can also lead to a *clash*, which occurs when the reasoner detects a logical contradiction. If the reasoner can construct a clash-free set of assertions to which no more derivation rules are applicable, the reasoner concludes that the ontology O is satisfiable; otherwise, if an attempt to construct such a clash-free set of assertions fails, the reasoner concludes that O is unsatisfiable. Each derived assertion is derived either deterministically or nondeterministically. An assertion is *derived deterministically* if it is derived by the application of a deterministic derivation rule from assertions that were all derived deterministically; all other assertions are *derived nondeterministically.*

We illustrate the reasoning process employed in HermiT using the following example ontology.

`Class:` D `SubClassOf:` E `or` B	$D(x) \rightarrow E(x) \vee B(x)$	(1)
`Class:` A `SubClassOf:` r `some` B	$A(x) \rightarrow \exists r.B(x)$	(2)
`ObjectProperty:` r `Domain:` C	$r(x, y) \rightarrow C(x)$	(3)
`Individual:` i `Types:` A		(4)
`Individual:` j `Types:` D `and not` B		(5)

Axioms (1)–(3) are TBox axioms, and their translation into rules is shown on the right-hand side. Axioms (4) and (5) are ABox assertions. Figure 6 shows a graphical representation of the initial ABox for assertions (4) and (5) on the left-hand side and an extended ABox obtained by applying the rules (1) to (3) on the right-hand side. The derivation that individual j is an instance of class E is nondeterministic since rule (1) also allows for choosing B a possible type for j. Individual n_1 is added due to rule (2); this makes rule (3) applicable, so C is added to the types of i. At this point no more rules are applicable and the constructed ABox is clash-free, so we conclude that the ontology is satisfiable.

In order to check whether O entails an axiom α, one typically checks whether O extended with the negation of α is satisfiable. If that is not the case, then every model of O satisfies α, and so O entails α. For example, to check whether an individual i is an instance of a class C in O, one extends O with an assertion stating that i is an instance of the negation of C and checks whether the extended ontology is unsatisfiable. In our example, HermiT will fail to construct a clash-free set of assertions, and so it will conclude that individual i is indeed an instance of class C.

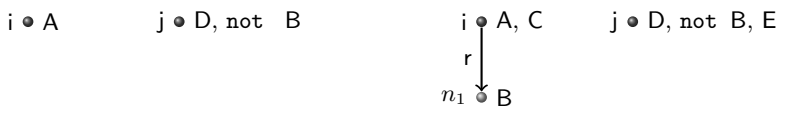

Fig. 6. A graphical illustration of the initial ABox and an extended ABox obtained by applying the rules (1) to (3)

Please note, however, that in the above example we have no choice but to derive that i is an instance of C. This observation can be generalised as follows: whenever we derive deterministically that an individual i is an instance of a class C, then this holds in each model of O; in other words, we conclude that O implies that i is an instance of C. Similarly, whenever we construct a clash-free set of assertions in which i is not an instance of C, we conclude that O implies that i is not an instance of C. Only if we derive nondeterministically that i is an instance of C, we cannot be sure whether O really entails this fact, so we have to actually perform a separate test. For example, in the model abstraction shown on the right-hand side of Figure 6, the assertion that individual j is an instance of class E was derived nondeterministically, so we do not know for sure whether O implies that j is an instance of E, and so we must perform an actual test; in this particular case, if we extend O with an assertion that j is an instance of not E, we obtain a contradiction since have to choose the second alternative B in rule (1), contradicting the fact that j has type not B due to assertion (5). In order to realise an ontology efficiently, HermiT determines the certain and possible instances of classes and properties from model abstractions as described above, and it performs the remaining tests lazily (i.e., at query time or when explicitly requested by the user). Furthermore, if the ontology does not contain disjunctive information, then all assertions in a model abstraction are derived deterministically, and so HermiT can determine all class and property instances by performing a single ontology satisfiability test, thus considerably improving the system's performance.

4.1 Incremental Ontology Changes

Even with all the optimisations outlined so far, HermiT cannot process the information about all customers at once. Thus, the ontology has been designed as follows: the TBox contains general statements about the domain (e.g., the taxonomy of different kinds of heating systems), while the ABox contains simple assertions that describe the situation of one or several customers. The customers are independent of each other, so we can process them in batches, possibly even on different reasoner instances in a cluster of machines. After loading, classification, and realisation, we retrieve the types of each tip individual in the batch (recall that each customer is connected via the hasTip property to an instance of the Tip class) using a suitable SPARQL query (see Section 2.2), and then we generate the appropriate tips based on the retrieved types.

Initially, HermiT did not support any form of incremental reasoning, so for each batch of customers it was necessary to reload TBox and the ABox, reclassify the ontology, and then realise the relevant instances. In order to improve this process, we extended HermiT (v1.3.4 onwards) with a limited support for incremental addition and retraction of axioms. In particular, HermiT supports incremental changes only for class assertions with named or negated named classes, and for property assertions; moreover, the TBox cannot contain nominals so that TBox classification becomes independent of the ABox. These criteria are satisfied in the ontology used in EMA, which allows us to first load and classifying the TBox, and then iteratively load and realise an ABox for each batch of customers. These changes to HermiT considerably improved the performance of the EMA service.

5 Evaluation

To determine a good batch size, we tested how long it takes to compute tips for 10,000 customers by processing them in batches of 1, 2, 4, and 8 customers with and without incremental reasoning support (i.e., when loading and classifying the TBox for each customer). The results in Table 1 show that a surprisingly small batch of 4 customers works best. The time is given in the format min:sec and, apart from the total time, we also show the time for loading (including the updates in the incremental mode), classification, and realisation. In the incremental mode, the ontology is classified only once, which takes between 40 and 50 ms, which we round to 0 s. With smaller batch sizes, there is too much overhead for loading, whereas with bigger batch sizes the time to realise the ABox no longer outweighs the reduction in loading time.

The table also shows the results for 50,000 and 100,000 customers, where we only compared the initial approach (processing each customer separately without incremental reasoning) and the approach based on incremental reasoning with 4 customers processed at a time. Note that the overall processing time increases by a factor of 5 and 10, respectively, which shows a linear increase in the number of customers. Thus, by designing the ontology that allows independent processing of customers, and by combining it with incremental loading of relatively simple ABox assertions, we developed a system that can process a large number of customers in a reasonable time. The overall time can further be reduced by running several reasoner instances in parallel.

The tests were performed on a MacBook Air with an 1.8 GHz Intel Core i7 processor, and 4GB of main memory. We used Java 1.6 and allowed for 1GB of Java heap space. The times shown are the average over two runs and have been rounded to seconds.

6 Discussion and Future Work

The EMA is today used to produce tips only for about 300,000 of EDF's customers, each of whom receives energy saving advice twice per year. EDF's goal,

Table 1. Results for computing tips for 10,000, 50,000, and 100,000 customers with different batch sizes and with or without incremental reasoning

Customers	10,000								50,000		100,000	
Batch Size	1	1	2	2	**4**	**4**	8	8	1	4	1	4
Incremental	✓		✓		✓		✓			✓		✓
Loading	19	18	9	9	5	5	3	3	1:27	24	2:47	53
Classification	0	28	0	15	0	8	0	4	2:17	0	4:30	0
Realisation	10	9	9	12	11	17	18	29	42	54	1:34	1:50
Total time	30	56	18	35	**16**	30	21	36	4:27	1:18	8:50	2:43

however, is to provide such advice to all of its 35 million customers in France. This much larger customer base might require greatly extended modelling of both situations and tips, which has prompted us to consider features for the next generation of the Energy Management Adviser. We have identified several directions for future research and enhancement.

6.1 Using Modular Tip Fragments

In the existing EMA service, tips are represented as single classes, and each tip class is associated with an appropriate customer situation class. This leads to a combinatorial explosion of tip and customer situation classes, and increases development and maintenance cost for both the ontology and the tips.

For example, the ontology currently represents customer energy consumption using classes highCons (high consumption), normCons (normal consumption) and lowCons (low consumption); furthermore, it represents relevant environmental conditions using classes warmSummer, normSummer and coolSummer. This gives rise to nine distinct situations (highCons and warmSummer, ..., lowCons and coolSummer), each of which may be associated with a different tip.

An alternative design might associate tip fragments with each elementary situation, use reasoning to determine the relevant elementary situations, and then assemble the relevant fragments into a coherent tip. This, however, is nontrivial, as different combinations of circumstances may require fragments to be assembled in a different way. For example, given a customer who has air conditioning and whose circumstances also include highCons and warmSummer, we might need to combine the relevant tip fragments 'high consumption' and 'warm summer' into 'high consumption explained by the warm summer'; however, after changing the circumstances to coolSummer, we might need to combine the relevant tip fragments 'high consumption' and 'cool summer' into 'high consumption *despite* the cool summer'. Thus, the assembly of coherent tips will require sophisticated natural language generation, which may itself depend on background knowledge of the domain (e.g., that in an air-conditioned house a warm summer can explain high consumption), possibly captured in an ontology.

6.2 Triple store

The EMA service currently provides tips by means of a web service, and information about customers is stored elsewhere (typically in databases) and passed to the service as needed. The reasoning process then analyses the data and computes the relevant tips by classifying the individual customers.

A new version of EMA might store all relevant customer information in a triple store and thus use reasoning also for other purposes than tip computation. For example, the system could be used to compare different customers, analyse historical energy consumption, and even integrate new data sources containing, for example, spatial data. Furthermore, the information stored in such a system could be used to drive customer interfaces that solicit (possibly over multiple interactions) additional information about customer circumstances that allow for more precision when identifying relevant tips.

Finally, SWRL reasoning capacities might allow us to capture in a declarative way all formulae for the transformation of various parameters—e.g., comparing the energy used for air conditioning and for heating (see Section 2.1). At present, such transformations are managed programmatically, which makes maintenance of the system more difficult.

References

1. *SPARQL 1.1 Overview*. 21 March 2013.
2. D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. EQL-Lite: Effective First-Order Query Processing in Description Logics. In M. M. Veloso, editor, *Proc. IJACI 2007*, pages 274–279, Hyderabad, India, January 6-12 2007.
3. G. De Giacomo. *Decidability of Class-Based Knowledge Representation Formalisms*. PhD thesis, Dipartimento di Informatica e Sistemistica, Universita' di Roma "La Sapienza", 1995.
4. B. Glimm, I. Horrocks, B. Motik, R. Shearer, and G. Stoilos. A Novel Approach to Ontology Classification. *Journal of Web Semantics*, 14:84–101, 2012.
5. P. Hitzler, M. Krötzsch, B. Parsia, P. F. Patel-Schneider, and S. Rudolph, editors. *OWL 2 Web Ontology Language: Primer*. 27 October 2009.
6. M. Horridge and S. Bechhofer. The OWL API: A Java API for OWL ontologies. *Semantic Web Journal*, 2(1):11–21, 2011.
7. M. Horridge and P. F. Patel-Schneider, editors. *OWL 2 Web Ontology Language: Manchester Syntax*. 18 October 2012.
8. F. Manola and E. Miller, editors. *Resource Description Framework (RDF): Primer*. 10 February 2004.
9. B. Motik, I. Horrocks, R. Rosati, and U. Sattler. Can OWL and Logic Programming Live Together Happily Ever After? In *Proc. ISWC 2006*, volume 4273 of *LNCS*, pages 501–514, Athens, GA, USA, November 5–9 2006. Springer.
10. B. Motik, R. Shearer, and I. Horrocks. Hypertableau reasoning for description logics. *Journal of Artificial Intelligence Research*, 36:165–228, 2009.