SAT based Enforcement of Domotic Effects in Smart Environments

Journal of Ambient Intelligence and Humanized Computing manuscript No.
(will be inserted by the editor)

SAT based Enforcement of Domotic Effects in Smart Environments

Fulvio Corno · Faisal Razzak

the date of receipt and acceptance should be inserted later

Abstract The emergence of economically viable and efficient sensor technology provided impetus to the development of smart devices (or appliances). Modern smart environments are equipped with a multitude of smart devices and sensors, aimed at delivering intelligent services to the users of smart environments. The presence of these diverse smart devices has raised a major problem of managing environments. A rising solution to the problem is the modeling of user goals and intentions, and then interacting with the environments using user defined goals. 'Domotic Effects' is a user goal modeling framework, which provides Ambient Intelligence (AmI) designers and integrators with an abstract layer that enables the definition of generic goals in a smart environment, in a declarative way, which can be used to design and develop intelligent applications. The high-level nature of domotic effects also allows the residents to program their personal space as they see fit: they can define different achievement criteria for a particular generic goal, e.g., by defining a combination of devices having some particular states, by using domain-specific custom operators. This paper describes an approach for the automatic enforcement of domotic effects in case of the Boolean application domain, suitable for intelligent monitoring and control in domotic environments. Effect enforcement is the ability to determine device configurations that can achieve a set of generic goals (domotic effects). The paper also presents an architecture to implement the enforcement of Boolean domotic effects, and results obtained from carried out experiments prove the feasibility of the proposed approach and highlight the responsiveness of the implemented effect enforcement architecture.

Keywords Ambient Intelligence · Domotic Effects · Domotic Effect Enforcement · SAT based Enforcement · High level modeling · DogEffects ontology

Fulvio Corno
Politecnico di Torino, Dipartimento di Automatica ed Informatica, Corso Duca degli Abruzzi 24, 10129 - Torino, Italy
E-mail: fulvio.corno@polito.it

Faisal Razzak
Politecnico di Torino, Dipartimento di Automatica ed Informatica, Corso Duca degli Abruzzi 24, 10129 - Torino, Italy
E-mail: raja.faisal@gmail.com

1 Introduction

The last decade saw the emergence of economically viable and efficient sensor technology which can be integrated with appliances, enabling them to sense different parameters of their respective environments, i.e., temperature, luminosity, pressure, etc. It helped realizing the vision of smart environments (Weiser, 1995) by developing heterogeneous dynamic ensembles: groups of co-located devices of different types which evolve over time (Bader and Dyrba, 2011). Such environments promise to offer additional intelligent capabilities that go beyond the integrated and remote control of appliances present in the environment. But the presence of diverse devices and the associated complexity has given rise to a major problem in the past years, i.e., the problem of providing users with the ability to control and manage their respective environments.

State of the art revolves around the issues related to communication protocols and technologies (Dey et al, 1999; Rashidi and Cook, 2009; Kawsar et al, 2008). Many approaches are furthermore based on abstract modeling of smart devices, resorting to some knowledge representation tool (e.g., ontologies (Heider and Kirste, 2002; Encarnaçao and Kirste, 2005; Bonino and Corno, 2008)), but the research trend is moving from a

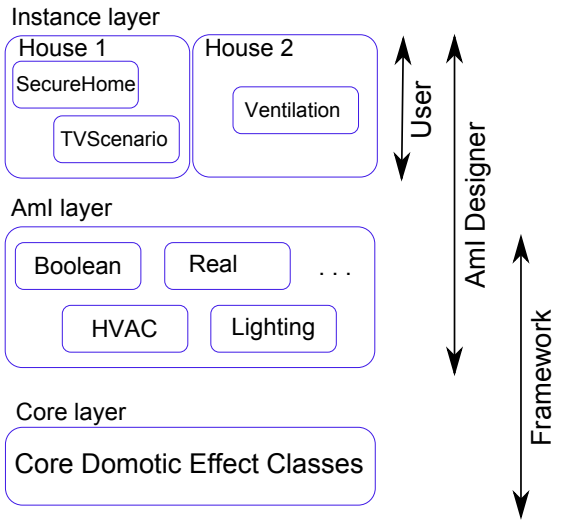

Fig. 1 Logical architecture of the Domotic Effects Framework (Corno and Razzak, 2012)

traditional device-centric vision (bottom-up) to a vision of providing higher level design for user interaction and control (Ducatel et al, 2003; Amigoni et al, 2005; Cheng et al, 2009; Kaldeli et al, 2010), i.e., user goal modeling. However, this research trend has received little attention as acknowledged in (Chen et al, 2012). One such, user goal modeling approach is 'Domotic Effects' (DE) framework (Razzak, 2013).

DE framework models user intentions or goals (called *domotic effects*) (Razzak, 2013; Corno and Razzak, 2012). The context is that every device in a smart environment is capable of providing certain visible (perceivable) effects for a user. These effects are fulfilled by possible states of the device. For example, an effect of *illumination* can be provided by a lamp in 'ON' state. However, modern devices are complicated in nature and a single device can have a composite state, which may be modeled as concurrent sub-states. These sub-states are orthogonal regions combining multiple descriptions of a device. For example, a TV set may have an on-off state (with possible values ON or OFF), a volume state (with possible values 0 through 100), a channel state (with possible values depending on the set of programmed channels). A device state is therefore composite in nature and therefore it is modeled as the parallel composition of different sub-states.

There might be cases in which an effect can only be fulfilled by a combination of devices having particular states. For example, the effect of *securing* a building may require all the exit doors and windows to be closed. In the context of DE framework, an effect that depends upon a single device (having a state or sub-states) is called a simple effect (SE) and an effect dependent on a combination of devices (having particular states and sub-states) is called a complex effect (CE). A CE is described by combining SEs and other CEs.

The DE framework is logically organized in a 3-tiered architecture as shown in Fig. 1. The *core layer* contains the basic class definitions for expressing domotic effects. Each Domotic Effect (DE) is expressed as a function of existing device states or sensor values. Such function is expressed using a set of operators that can be extended by a AmI designer. The *AmI layer* encodes the set of operators defined or customized by the AmI designer, depending on the application domain. Finally, the *Instance layer* represents the specific DEs being defined in a specific environment.

The DE framework addresses the concerns from perspectives of the AmI designer and the residents. It provides AmI designers with an abstraction layer that enables the definition of generic goals inside the environment, in a declarative way, and that can be used to design and develop intelligent applications. It provides a general framework for expressing functional properties, in a domain-dependent way: for each application domain, the AmI designer may choose the most suitable representation, and define suitable functional operators. Using these operators, various user goals are then defined in a specific environment. The high-level nature of the DEs, on the other hand, also allows the residents to program their personal, office or work spaces as they see fit: they can define different achievement criteria for a particular generic goal, by using the domain-specific operators defined in the previous phase.

This paper discusses the control aspect of the DE framework. The control aspect involves the ability of users to manage and control their environments with the help of user-defined intentions or goals. This amounts to correctly mapping user goals in terms of a combination of devices having particular states. In this paper, the control aspect of the *DE* framework restricted to Boolean application domains is discussed. Boolean application domains are such domains in which the values of user goals (effects) can either be true (active) or false (inactive). They cover most control applications and many monitoring use cases in smart homes, offices and industrial plants.

The paper is divided into seven sections. Section 2 provides the formal and conceptual modeling for domotic effects specialized to the case of Boolean application domains, and then defines the problem tackled in the paper. The general approach adopted for enforcement is described in Section 3, and later Section 4 defines its architecture and implementation. Section 5 shows results of the experiments carried out on effects enforcement. Section 6 compares our approach to some

related works and Section 7 concludes the paper and highlights future work.

2 Modeling

2.1 Formalism

Given an intelligent environment, we define as $\mathcal{D}$ the set of installed *controllable devices* $d \in \mathcal{D}$. Each device is characterized by a *device category* that, among the other things, defines the allowed *sub-states* for a device. Depending on the device category, for each device d we define the set of allowed sub-states $S(d)$; this set may be discrete (e.g., {ON, OFF} for a lamp) or continuous (e.g., [0, 100] for a volume knob). During system evolution, the actual state of each device is a time-dependent function $s(d,t) \in S(d)$. The whole environment therefore possesses a *global state space* $\mathcal{G}$, represented by the Cartesian product of all device state spaces: $\mathcal{G} = \prod_{d \in \mathcal{D}} S(d)$, thus defining a global environment state $g \in \mathcal{G}$.

Formally, a Domotic Effect *DE* is defined as a function of the global state space: $DE : \mathcal{G} \rightarrow \mathcal{V}$, where $\mathcal{V}$ is an application-dependent value space. For example, for control applications, $\mathcal{V} = \{0, 1\}$ since each Domotic Effect represents the activation of a given state configuration; conversely, when dealing with energy savings, $\mathcal{V} = \Re^+$ since Domotic Effects may be used to represent consumed power.

AmI designers and end users may define custom Domotic Effects by working with a domain-specific set of operators. Such operators work on the value space $\mathcal{V}$ relevant to the specific application domain[1]. The specification of a *DE* function requires three levels of formalization:

1. defining Simple Effects (SE), to extract a $\mathcal{V}$-valued quantity from a single device state. Formally, SE is a function that considers the state on only one device, $SE : S(d) \rightarrow \mathcal{V}$; such function is also time-dependent since it depends on $s(d,t)$.
2. defining *effect operators* working within $\mathcal{V}$-space algebraic semantics, suitable for composing new functions in the application domain. Formally, an operator `op` is a function $\texttt{op} : \mathcal{V}^N \rightarrow \mathcal{V}$, where N represents the number of operands of the specific `op`.
3. defining Complex Effects (CE), by applying effect operators to the values computed by other SE or CE. Formally, a CE is represented by a couple ($\texttt{op}$, $(DE_1 \dots DE_N)$) composed of an operator name `op` and a list of Domotic Effects DE_i whose values are used as operands.

For each application domain, there would be a set of pre-defined SE and a set of operators that the users may combine to compute the values of interest. For example, if we consider control applications ($\mathcal{V} = \{0, 1\}$), then the SE functions that may be adopted are:

- for discrete-valued states, a SE detects whether a device currently is in any given state. E.g., $SE_{\text{ON}}(d,t) = (s(d,t) == \text{ON})$.
- for real-valued sensors, a SE usually compares the current sensor data with a threshold. E.g., $SE_{\text{HOT}}(d,t) = (s(d,t) > 30^oC)$.

The instance layer defines a set $\mathcal{I}$ of all defined domotic effects (instances), i.e., $\mathcal{I} = \{DE_1, DE_2 \dots DE_N\}$.

2.2 Conceptual Modeling

In order to provide a formal knowledge-base for the DE framework, a modular 'DogEffects' ontology is developed. The DogEffects ontology is organized in a three-tier structure, corresponding to the logical architecture of the DE framework (Fig. 1). It models the user defined goals and their mapping to devices and their corresponding states. The DogEffects ontology requires the concepts of devices and their states, the modularity pattern was adopted for designing the ontology. Modularity allows the ontology to easily integrate with other ambient ontologies that model environments. In our case, DogOnt (Bonino and Corno, 2008) is adopted. Three modeling layers of the DogEffects ontology are explained below:

2.2.1 Core Layer

The core layer defines concepts lying at the foundation of the DE framework. The main concepts are illustrated in Fig. 2. The core layer consists of three main classes, i.e., Effect, EffectOperator and Operand. Every *Domotic Effect* is formally organized into a concept hierarchy inheriting from the `dogEffects:Effect` class. Effects can either be simple (`dogEffects:SimpleEffect`) or complex (`dogEffects:ComplexDeviceEffect`). For both kinds of effects, domain-dependent subclasses are defined at the AmI layer.

Simple Effects (SEs) are the terminal nodes of the representation and compute a value depending on a device state or sensor value. SEs act as interface points between the DogEffects ontology and some device description ontology (e.g., DogOnt). The `dogEffects:effectOf` and `dogEffects:functionOf` open relations

[1] for cross-domain applications, $\mathcal{V}$ would be the union of all relevant value spaces

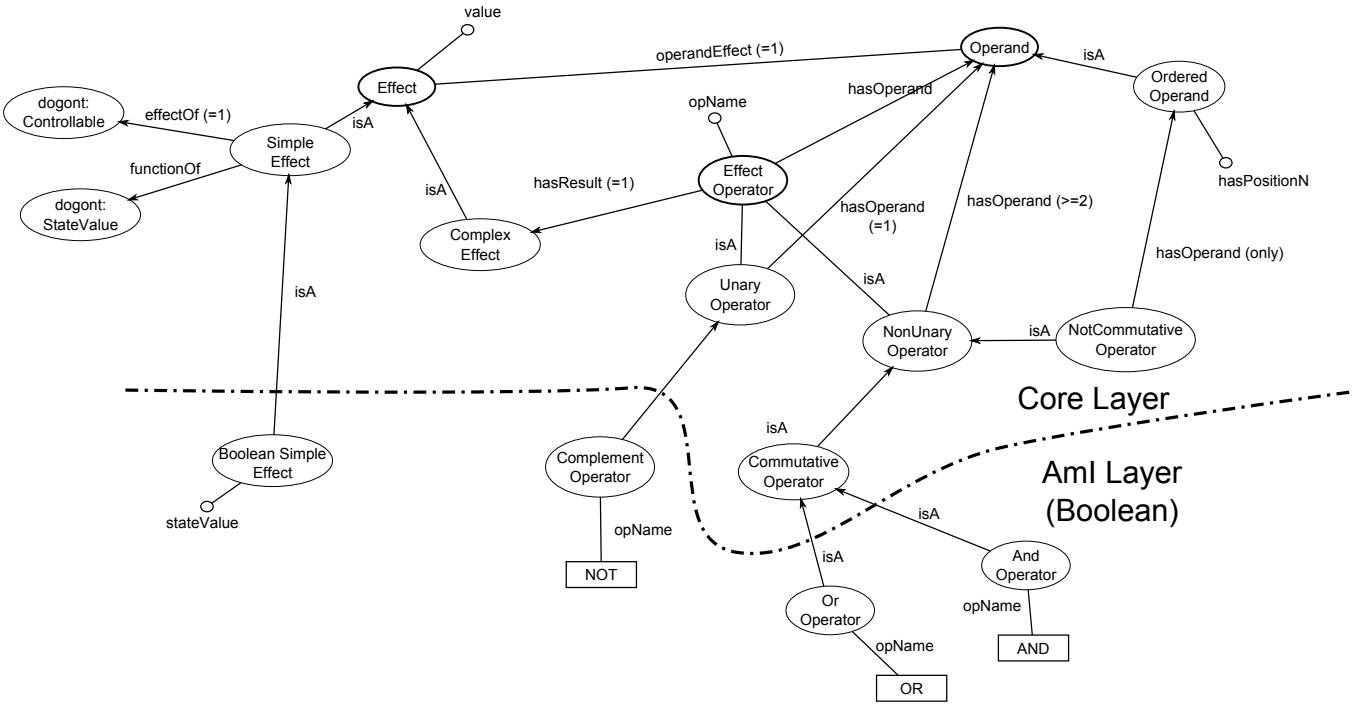

Fig. 2 The DogEffects ontology (Core and AmI layer) - Boolean Application Domain

(i.e., relations without range restrictions) permit to identify the device and the device state for which a given SE is computed, respectively.

Every Complex Effect (CE) represents a functional expression of SEs and other CEs declared by using domain-dependent operators defined at the middle-layer of the DomoticEffects framework. Effect operators take either simple or complex effects as operands (through the `dogEffects:hasOperand` relation) and generate new CEs as result, identified by means of `dogEffects:hasResult` relation.

Two main disjoint families of operators are modeled: unary operators (`dogEffects:UnaryOperator`) and non-unary operators (`dogEffects:NonUnaryOperator`). Unary operators only involve one `dogEffects:Operand` (cardinality restriction on the `dogEffects:hasOperand` relation), which can either map to a SE or a CE (by the `dogEffects:operandEffect` relation). Typical examples of unary operators are the NOT operator (in the boolean control domain). Non-unary operators are further specialized (disjoint union) into commutative (`dogEffects:CommutativeOperator`) and not commutative (`dogEffects:NotCommutativeOperator`) operators. According to the mathematical definition of commutative (not-commutative) operators, the result produced by the former (`dogEffects:CommutativeOperator`) is independent on the order in which operands are evaluated while, the latter operator provides a results depending on the order of the effect operands. In such a case the `dogEffects:OrderedOperand` subclass of operands shall be used to account for the operand order, expressed as an ascending integer number by the `dogEffects:hasPositionN` property. Typical examples of non-unary effects are the AND, OR and EX-OR logic operators, in the boolean control domain.

2.2.2 AmI layer

The AmI layer allows AmI designers to define the domain-dependent effect operators that govern the combination of domotic effects. Every application domain will define different operator classes for this layer by sub-classing the general operator classes defined in the core layer. In Boolean application domain, simple effects correspond to devices (sensors) being in specific states (measuring specific range of values). SEs and CEs can only evaluate to true or false, and Boolean logic is sufficient to compute CEs and to implement rather advanced activation scenarios. From the modeling point of view, mapping operators to Boolean logic requires a minimum set of logic operators, e.g., AND ($\wedge$), OR ($\vee$) and NOT ($\neg$) (see Fig. 2, bottom); however, AmI designers may choose to define more complex, user-intelligible operators such as the ONE_OF operator (true iff exactly one of the operands is true), the NONE_OF operator, and so on. Some of them are explained below:

1. *ImpliedOperator:* This operator represents the "logical implication" relationship.

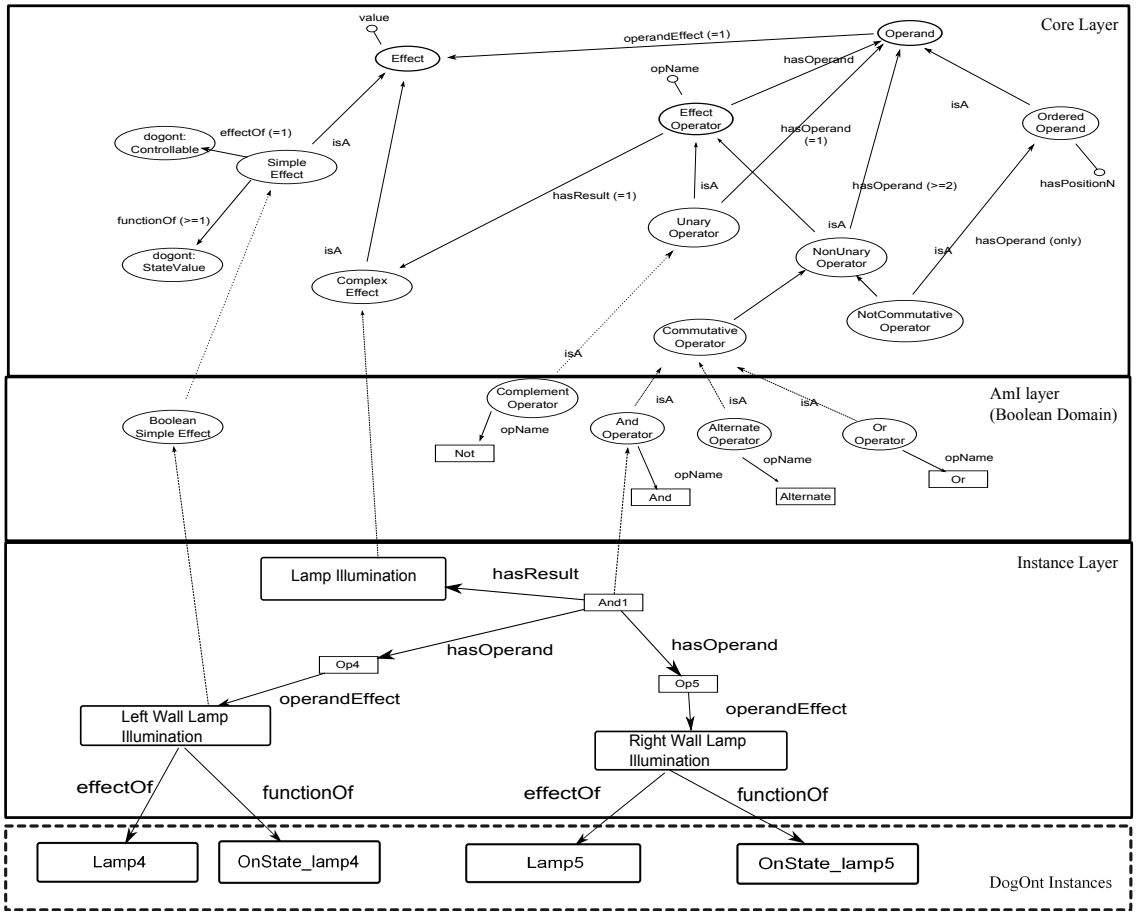

Fig. 3 An example of DogEffects ontology defining CE and SEs - Lamp Illumination use case

2. *AlternateOperator (**A**):* This operator represents a function whose value is active when *exactly one* of its operands is active. It is commutative and non-unary. Mathematically, the Alternate effect operator can be defined as Equation 1.
3. *ExactlyMOperator:* This non-unary operator represents a function whose value is active when *exactly* M number of its operands are active. Suppose there are n operands, i.e., $OP = \{1, 2, \ldots n\}$. Then the *ExactlyMOperator* effect operator can be defined as Equation 2.

$$Alt(x_1 \ldots x_n) = \sum_i \left(x_i \cdot \prod_{j \neq i} \overline{x_j} \right). \quad (1)$$

$$Exactly_M(x_1 \ldots x_n) = \sum_{O \subseteq OP, |P|=M} \left[\prod_{i \in O} x_i \cdot \prod_{j \notin O} \overline{x_j} \right] \quad (2)$$

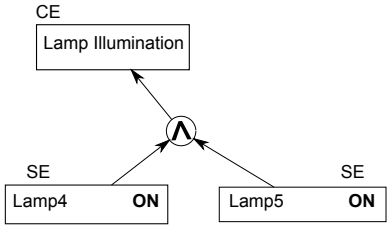

Fig. 4 A simplified representation of the Lamp Illumination CE

2.2.3 Instance layer

The instance layer of the DE framework represents specific DEs defined in a given smart environment. They are modeled as instances of the classes defined in the core and AmI layer. The DEs are defined according to user preferences, i.e., can be defined by the users using a GUI or can be designed by AmI designers.

Consider a trivial example in which a user wants to achieve illumination in the room using lamps, called 'Lamp Illumination' use case[2]. The 'Lamp Illumination'

[2] A use case represents a scenario or the overall effect that a user wants to achieve.

use case is illustrated in Fig. 3. The 'Lamp Illumination' use case will be represented as a 'Lamp Illumination' CE (an instance of `ComplexEffect`). Suppose the "Lamp Illumination" can be achieved using 'Left Wall Lamp Illumination' SE and 'Right Wall Lamp Illumination' SE. The combination is governed by the "And1" instance of `And Operator`) class. The 'Left Wall Lamp Illumination' SE represents the 'Lamp4' in 'OnState_lamp4' state, while the 'Right Wall Lamp Illumination' SE represents the 'Lamp5' in 'OnState_lamp5' state. The information about specific devices and their states comes from the DogOnt ontology. In order to provide easy comprehension Fig. 4 shows the simplified graphical representation of the 'Lamp Illumination' CE and Table 1 outlines the functional representation of the 'Lamp Illumination' CE. In Table 1, a SE is represented as **SE**(device, sub-state(s)). For instance, the 'Right Wall Lamp Illumination' SE is equivalent to **SE**(Lamp5, OnState_lamp5). On the other hand, a CE is represented as **Operator**$(DE_1, DE_2 \ldots)$. For instance, the 'Lamp Illumination' CE is equivalent to **And**(Left Wall Lamp Illumination, Right Wall Lamp Illumination).

The reader interested in more details about the DogEffects ontology is referred to (Razzak, 2013).

2.3 Problem Statement

Consider a smart environment with an AmI system managing it. A user can define several domotic effects (simple and complex) on top of the domotic structure, based on the effect operators defined for the environment. At any instant, each domotic effect has a value associated with it. The user has the ability to request R the AMI system to enforce a set of domotic effects on the environment. 'Effect Enforcement' addresses the problem of finding at least one configuration that satisfies the user's request R. The configuration refers to the combination of devices having particular states and sub-states.

The user request R is defined as a subset of the domotic effects present in the instance layer: $R \subseteq \mathcal{I}$. In simple terms, the user request R is the subset of DE_i that the user wants to be active (true) at a given instant.

Given a request R, effect enforcement consists of finding a global domotic state $g \in \mathcal{G}$ where *all* the domotic effects $DE_i \in R$ are true. This is equivalent to computing the *satisfiability* of the function shown in Equation 3.

$$F_R(g) : \prod_{DE_i \in R} DE_i \tag{3}$$

3 Approach

In order to enable the user to enforce particular values of domotic effects on the environment, at least a configuration needs to be found which fulfills the user request R, as defined in Section 2.3. To solve this problem the paper proposes to transform the user's request into a Boolean satisfiability problem (SAT). SAT is a decision problem to determine whether a given propositional formula is a tautology (Cook, 1971). In other words, it establishes if the variables of a given formula can be assigned in such a way as to make the formula evaluate to TRUE. Equally important is to determine whether no such assignments exist, which would imply that the function expressed by the formula is identically FALSE for all possible variable assignments.

To transform the user's request into a SAT problem, each domotic effect defined in the instance layer is mapped as a Boolean variable. The functionality of each effect operator defined in the AmI layer is mapped in terms of a Boolean sub-expression in the SAT problem. The value of the variable corresponding to the Simple Effect is true (active) if and only if the device is in a particular sub-state(s). Meanwhile, complex domotic effects can depend upon the values computed by multiple simple or complex domotic effects and, therefore, the value corresponding to their variables are dependent on the values of their operands. As a consequence, the Boolean expressions for a complex domotic effect are constructed over its dependent domotic effects using the effect operator defined for it. The process is recursive, as the Boolean expressions for all the operands are constructed and conjuncted.

For example, consider a trivial user request R to enforce the 'Illumination' use case on the environment. The 'Illumination' use case represents the user intention to achieve illumination in a room. The 'Illumination' use case will be represented as an 'Illumination' CE inside the 'DogEffects' ontology and it is graphically shown in Fig. 5. The functional representation is listed in Table 2. It can be seen that the illumination in the room can be achieved by means of either natural lightening (represented as 'Natural Illumination' SE) or artificial lightening (represented as 'Artificial Illumination' CE). The natural lightening is achieved by opening a shutter, whereas the artificial lightening is achieved by using different combinations of lamps in the room.

In order to build the Boolean expressions for the user request, all domotic effects are represented as Boolean variables. Then, the effect operator (and its type) attached with the 'Illumination' CE is extracted, i.e., *OR*, which is followed by the extraction of operands (do-

Table 1 Lamp Illumination functional form

$LampIllumination = \mathbf{And}(LeftWallLampIllumination, RightWallLampIllumination)$
$RightWallLampIllumination = \mathbf{SE}(Lamp5, OnState\_lamp5)$
$LeftWallLampIllumination = \mathbf{SE}(Lamp4, OnState\_lamp4)$

Table 2 Illumination functional form (CE_B)

$Illumination = \mathbf{Or}(ArtificialIllumination, NaturalIllumination)$
$ArtificialIllumination = \mathbf{Alternate}(CeilingLampIllumination, \quad MirrorLampIllumination)$
$MirrorLampIllumination = \mathbf{And}(LeftMirrorLampIllumination, RightMirrorLampIllumination)$
$RightMirrorLampIllumination = \mathbf{SE}(Lamp5, OnState\_lamp9)$
$LeftMirrorLampIllumination = \mathbf{SE}(Lamp4, OnState\_lamp8)$
$CeilingLampIllumination = \mathbf{SE}(Lamp2, OnState\_lamp2)$
$NaturalIllumination = \mathbf{SE}(ShutterActuator, UpStateValue\_ShutterBath)$

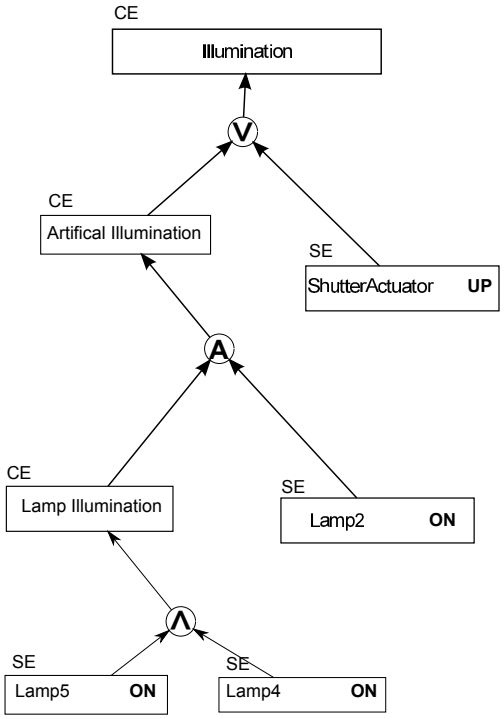

Fig. 5 A simplified graphical representation of the Illumination CE

motic effects) attached with the operator, i.e, *Natural illumination* and *Artificial Illumination*. After the extraction of operator and operands, the first Boolean expression (shown in Equation 4) is constructed. Then, the Boolean expressions for *Natural illumination* and *Artificial Illumination* CEs are constructed recursively, until SEs are reached. All the Boolean expressions are then conjuncted and the value of Boolean variable associated with *Illumination* CE is set to true.

$$\text{Illumination} = \mathbf{OR}(\text{Natural Illumination, Artificial Illumination}) \tag{4}$$

To put it concisely, at any instant the user can request R several domotic effects DE_i, to be enforced on the environment. The Boolean expressions for all domotic effects DE_i present in the user request R are constructed and conjuncted. The process is recursive, as the Boolean expressions for all the operands are constructed too. After getting all the Boolean expressions, the DE_i corresponding present in the user request R are enforced as SAT constraints, i.e., the values of variables corresponding to DE_i are set to true.

Once the Boolean expressions are constructed, conjuncted and the values of the variables corresponding to DEs in R are set, they are fed to a SAT solver to determine values of other variables (corresponding to other DEs) under which the values of the DEs in R will hold. Since SEs represent terminal nodes of the expressions, the values of the variables corresponding to SEs will give us a combination of devices and their particular states and sub-states fulfilling the user's request R. In short, bringing the combination of devices into particular states and sub-states would fulfill R. Whenever the user request R is not satisfiable, the enforcement procedure is canceled and the user is informed. Additionally, it is likely that several configurations satisfy R which gives system designers an option to find an optimal configuration based on some constraints. For example, a configuration that minimizes energy consumption (Corno and Razzak, 2012).

4 Architecture

This section describes a generic, modular and extensible architecture for the implementation of the effect enforcement approach (defined in Section 3) inside the smart environments and highlights the procedure to extend the architecture to define new effect operators. The architecture consists of a *Domotic Effect Enforcement* module and the *DogEffects* ontology containing all the domotic effects defined for the environment.

4.1 Domotic Effect Enforcement

Given a user's request R, the *Domotic Effect Enforcement* module finds a configuration to fulfill R. The *Domotic Effect Enforcement* module is responsible for extracting all domotic effects from the *DogEffects* ontology, receiving user's request for enforcing particular values for a set of domotic effects, transforming the user request into a SAT problem, and finding at least one configuration that fulfills the user's request (or otherwise finding conflicts).

The logical architecture of the *Domotic Effect Enforcement* module is depicted in Fig. 6. It consists of query, solver, library components and an effect operator store.

The query component queries the *DogEffects* ontology for all the domotic effects and then it organizes all the domotic effects in a hierarchical internal data structure which is similar to the organization of domotic effects in the *DogEffects* ontology. Whenever any addition or editing in the *DogEffects* ontology occurs, the querying component reconstructs the data structure.

As defined in Section 3, to transform the user's request into a SAT problem, the effect operators defined in the AmI layer should be defined in terms of Boolean sub-expressions. The Effect Operator store contains the Java code for each effect operator defined in the AmI layer by providing methods to create the corresponding sub-expressions in terms of basic Boolean operators.

The user's request is handled by the Solver component. The Solver component transforms the user request into a SAT problem, finds a configuration that satisfies the user's request and then it enforces the configuration on the environment. For each request R, the steps taken by the solution are detailed below:

- *Transformation and Feeding:* It comprises transforming the user's request for particular values of domotic effects in to a correct set of Boolean equations and applying constraints over them. Then, these Boolean equations and constraints are fed to the SAT solver. Currently, the Sat4j solver (Le Berre and Parrain, 2010) is used.
- *Solving:* Based on the set of Boolean equations, the Sat4j solver determines (if possible) the values of all the variables inside the Boolean equations. There may be cases in which the values of domotic effects requested in R can not be satisfied at all.
- *Interpretation:* It comprises finding the values of the variables corresponding to SEs and interpreting them in terms of devices and their states and sub-states.

The Sat4j library requires that the input is in the Conjunctive Normal Form and each variable in the SAT problem is represented by an integer positive number (negative numbers represent complemented variables). Therefore, the querying component assigns a unique integer to each domotic effect. The transformation begins by taking each user requested domotic effect and determining the effect operator that acts among its children. Once the effect operator type of a domotic effect is determined, the corresponding Java class in the Effect Operator Store creates its sub-expression in terms of basic Boolean operators and appropriate Boolean equations are constructed for the domotic effect and its children domotic effects. These Boolean equations are fed to the Sat4j solver to determine a configuration satisfying them.

4.2 Extensibility

In order to become a potential candidate for a wider adoption, an approach should demonstrate the characteristic of extensibility. The DE framework should also demonstrate such characteristic. In the DE framework the question of *extensibility* can be raised at two levels, i.e., Cross-Domain extensibility and In-Domain extensibility. Cross-Domain extensibility refers to the ability of the semantic modeling (DogEffects ontology) of the DE framework to be extended to application domains, other than Boolean application domain. In-Domain extensibility refers to the ability of the AmI designer to define new Boolean operators in Boolean application domain.

The DE framework provide Cross-Domain extensibility by enabling the semantic modeling of DogEffects ontology (AmI layer) for several application domains, i.e., Boolean domain, Energy Saving Domain. This paper discusses the specialized case of the semantic modeling in the Boolean application domain. An example of semantic modeling of the AmI layer in the Energy Saving domain is presented in (Razzak, 2013). In the Energy Saving domain the example of operators can be SUM, AVERAGE, THRESHOLD, INTEGRATION operators. The DE framework is based on an Ontology-Based approach and therefore it has an advantage of large-scale adoption, application development, system prototyping, solid technological infrastructure as acknowledged in (Chen et al, 2012).

In the Boolean application domain, the In-Domain extensibility of the proposed approach is achieved by providing the AmI designers control over defining and implementing their own Boolean operators (see Section 4). This paper defines the fundamental Boolean operators and their implementations, but the designers are free to define any operator that can be translated into a Boolean expression. The new Boolean operators

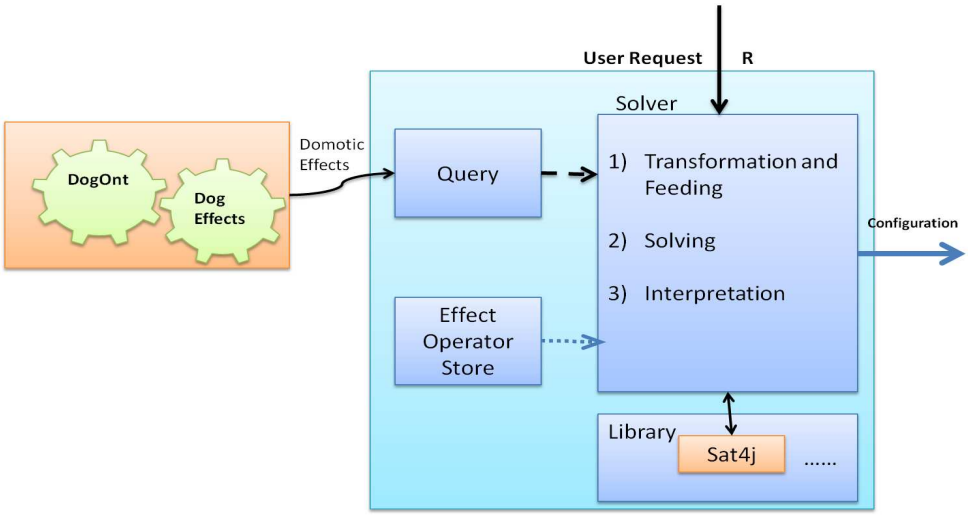

Fig. 6 Domotic Effect Enforcement Architecture

may be required by the AmI designers to define new rules to govern combination between domotic effects. Appropriate implementation of new operators should be included in the Effect Operator store for constructing Boolean equations. Suppose an AmI designer wants to declare a new Boolean effect operator called 'NewBooleanEffectOperator'. Then, in addition to declaring the new effect operator in the AmI layer of the DogEffects ontology, the AmI designer must also provide a proper implementation of the new Boolean effect operator in the effect operator store.

When a new Boolean effect operator is defined in the AmI layer of the 'DogEffects' ontology, the implementation of the new effect operator can be provide inside the 'Domotic Effect Enforcement' module by following two steps:

1. A new class extending the *EffectNode* class is defined. The*EffectNode* class represents the general properties of a domotic effect [3].
2. For the construction of Boolean equations, the mapping of the effect operator using basic Boolean operators is provided inside the *setEquation()* method. The *setEquation()* method receives a parameter (*gator* of type *GateTranslator*). The *gator* parameter accumulates all constructed Boolean expressions. The 'GateTranslator' is a Sat4j library class which provides functionalities of the SAT's basic Boolean operators like Not, And and Or. One can define any kind of effect operator in the AmI layer as long as it can be defined in terms of basic Boolean operators.

[3] The reader interested in more detail is referred to Razzak (2013)

Currently, the effect operator store includes the Complement, And, Or and Alternate operators. The Complement effect operator represents an invert relationship, and is mapped as a Not Boolean operator in SAT (Algorithm 1).

Algorithm 1 Complement effect operator

```
nodeNumber = node.getNodeNumber();
child=node.getFirstChild();
literals= new VecInt();
literals.push( child ) ;
gator.not(nodeNumber,literals);
return true;
```

Algorithm 2 shows the mapping of the And effect operator in terms of basic Boolean operators. The Or effect operator algorithm is similar, but it is mapped as an Or Boolean operator in SAT.

Algorithm 2 And effect operator

```
nodeNumber = node.getNodeNumber();
children=node.getChildren();
literals= new VecInt();
for all children as child do
   literals.push( child ) ;
end for
gator.and(nodeNumber,literals);
return true;
```

The Alternate effect operator (Algorithm 3) represents a function which is true when only one of its children is active.

5 Experimental evaluation

The 'Domotic Effect' modeling framework was developed to be integrated with the Dog2.0 (Bonino et al,

Algorithm 3 Alternate effect operator

```
nodeNumber = node.getNodeNumber();
children=node.getChildren();
literals= new VecInt();
globalNumber → Counter for temporary variables;
globalCounter → list of temporary variables;
literals.clear();
int_List = newList();
int_List.addAll( children ) ;
for count:=0 ; count < int_List.size(); count++ do
   literals.clear();
   for innercount:=0 ; innercount < int_List.size();
   innercount++ do
      if count =innercount then
         literals.push( int_List.get(innercount) ) ;
      else
         literals.push( - int_List.get(innercount) ) ;
      end if
   end for
   globalCounter.add(globalNumber++);
   gator.and(globalNumber, literals);
end for
literals.clear();
for all globalCounter as each do
   literals.push( each ) ;
end for
gator.or(nodeNumber,literals);
return true;
```

2008a) smart home gateway. Dog is an ontology-powered Domotic OSGi Gateway that is able to expose different domotic networks as a single, technology neutral, home automation system. It consists of 12 core bundles and it is built on top of the OSGi framework and the adoption of semantic modeling techniques allows Dog to support intelligent operations inside the home environment. Dog uses the DogOnt ontology to model an environment and it provides the ability to overcome issues like interoperability among different device/sensor protocols, validating device state or sensor value, etc. Two modules, i.e., Ontology Loader and Domotic Effect Enforcement, were built atop/deployed on Eclipse Equinox, which is an implementation of the OSGi framework (OSGI Alliance, 2003). The OSGi framework brings versatility and modularity by providing each module as a service called a bundle. Experiments were conducted to measure different performance parameters of the 'Domotic Effect Enforcement' module. These performance parameters include the time needed to transform a user's request R into a SAT problem, and if possible, to find at least a configuration that satisfies the user's request.

A complete house environment was simulated. The domotic structure was modeled as an instance of the DogOnt ontology. A new *TestDogEffectSolution* bundle was developed to perform the experiments and to measure performance parameters for each experiment. Based on the procedure defined in Section 2, we define six use cases $\{CE_A \dots CE_F\}$ (see Section 5.1). In order to define the use cases in the instance layer 190 intermediate domotic effects (CEs and SEs) were declared. A number of iterations were performed enforcing different user requests $R \subseteq \mathcal{I}$. In the experiments, a total of 63 iterations were performed, corresponding to each request R over 6 use cases (omitting the trivial $R = \emptyset$). The experiments were conducted on an Intel Core i5 CPU running at 2.6 GHz with 4GB of RAM.

5.1 Use cases

Based on the functional representation form presented in the Section 2, this section describes some use cases defined over a home. The house has a bed room, a living room, a lobby, a bath room, a store and a kitchen, and is equipped with with several automatic devices/appliances like lamps, oven, tv, door actuators, window actuators, shutter actuators, gas heaters etc. Based on simple domotic effects and using the set of Boolean operators encoded in the AmI layer, several complex domotic effects have been defined. Some of them are explained in the following sections.

Secure Home: The 'Secure Home' use case (CE_A) secures all the exit points of the house, i.e., by closing all the exit doors and shutting all the windows of the house. This use case comprises many DEs providing the ability to secure different rooms of the house. This can be used in case of emergency, theft, robbery or fire etc.

BathRoom Illumination: The 'BathRoom Illumination' (CE_B) combines small use cases that illuminate the bathroom. The illumination can be artificial by switching on the mirror lamps or ceiling lamp in different combinations, or illumination can be natural by opening the shutter of the window during morning and afternoon hours.

Home Illumination: The 'Home Illumination' (CE_C) requires that all the rooms of the house are illuminated. Illumination can be both natural or artificial in nature.

Afternoon Lunch: The 'Afternoon Lunch' (CE_D) deals with the daily routine of cooking lunch inside the kitchen. The resident desires the kitchen's oven to be heated, the television to be switched on and the kitchen to be closed so that the aroma of cooking does not spread to other rooms of the house.

Air Passage inside the house The 'Air Passage' use case (CE_E) manages the natural air flow inside the house or its different rooms. It combines different DEs that open windows of opposite sides for the flow of air between different rooms of the house.

Table 3 Secure Home use case (CE_A)

$SecureHome\_All = \mathbf{OR}(SecureHome\_Scenario\_1, SecureHome\_Scenario\_2);$
$SecureHome\_Scenario\_1 = \mathbf{AND}(LivingRoom\_l2\_WS\_CloseDown, BathRoom\_WS\_CloseDown, Kitchen\_WS\_CloseDown, BedRoom\_l1\_WS\_North\_CloseDown, BedRoom\_WS\_West\_CloseDown, DoorActuator\_d4\_lobby\_ext\_Close, LivingRoom\_L1\_WS\_CloseDown);$
$LivingRoom\_l2\_WS\_CloseDown = \mathbf{AND}(WindowActuator\_w6\_living\_Close, ShutterActuator\_sh2\_living\_Down);$
$LivingRoom\_L1\_WS\_CloseDown = \mathbf{AND}(WindowActuator\_w5\_living\_Close, ShutterActuator\_sh1\_living\_Down);$
$BathRoom\_WS\_CloseDown = \mathbf{AND}(WindowActuator\_w3\_bath\_Close, ShutterActuator\_bath\_Down);$
$BedRoom\_l1\_WS\_North\_CloseDown = \mathbf{AND}(WindowActuator\_w1\_living\_Close, ShutterActuator\_sh1\_Down);$
$BedRoom\_WS\_West\_CloseDown = \mathbf{AND}(WindowActuator\_w2\_Close, ShutterActuator\_sh2\_Down);$
$SecureHome\_Scenario\_2 = \mathbf{AND}(Secure\_LivingRoom, Secure\_BedRoom, Secure\_BathRoom, Secure\_Lobby, Secure\_Kitchen);$
$Secure\_LivingRoom = \mathbf{AND}(DoorActuator\_d7\_kitchen\_Close, DoorActuator\_d6\_living\_Close, Tv\_LivingRoom\_Off, LivingRoom\_L1\_WS\_CloseDown, LivingRoom\_l2\_WS\_CloseDown);$
$Secure\_BedRoom = \mathbf{AND}(DoorActuator\_d1\_bed\_Close, BedRoom\_l1\_WS\_North\_CloseDown, BedRoom\_WS\_West\_CloseDown);$
$Secure\_BathRoom = \mathbf{AND}(BathRoom\_Ws\_CloseDown, DoorActuator\_d2\_bath\_Close);$
$Secure\_Lobby = \mathbf{AND}(DoorActuator\_d6\_living\_Close, DoorActuator\_d5\_kitchen\_Close, DoorActuator\_d4\_lobby\_ext\_Close, DoorActuator\_d3\_lobby\_stor\_Close, DoorActuator\_d1\_bed\_Close, DoorActuator\_d2\_bath\_Close);$
$Secure\_Kitchen = \mathbf{AND}(DoorActuator\_d5\_kitchen\_Close, DoorActuator\_d7\_kitchen\_Close, Kitchen\_WS\_CloseDown);$
$Kitchen\_WS\_CloseDown = \mathbf{AND}(WindowActuator\_w4\_kitchen\_Close, ShutterActuator\_kitchen\_Down);$

Morning WakeUp: The 'Morning WakeUp' use case (CE_F) maps a typical scenario when a resident wants to perform a sequence of activities after waking up in morning, like illuminating the bedroom, the kitchen and the bathroom, switching off the gas heater inside the bedroom, switching on the television in the kitchen and the radio inside the bathroom.

The functional representation of the 'Secure Home' use case is given in Table 3. The function representations of other use cases can be found at (Razzak, 2013)

5.2 Results and Discussion

In the first experiment, two performance parameters were measured:

- the time taken by the 'Domotic Effect Enforcement' module to construct the set of Boolean equations and to feed them to the Sat4j solver (construction time, t_c);
- the time to find at least one configuration that satisfies the set of Boolean equations (solution time, t_s).

Both time measurements were taken at the milliseconds level, and Fig. 7 shows the performance measures for all the 63 iterations. Each cell contains a combination expressed as $t_c + t_s$. The results are represented as a Karnaugh map for easier reading and identification of the simultaneously enforced domotic effects.

It can be seen that the 'Domotic Effect Enforcement' module is quite responsive and in all cases the time for construction of Boolean equations and determining configuration is less than 100 ms. The module was developed to be used in real world applications and therefore completing the user's requests in few milliseconds shows that the proposed approach is promising. On the other hand, finding at least a configuration to satisfy user's request depends upon the number of variables involved in the Boolean expression. Though the measured time is in few milliseconds for this experiment, the time may vary according to the number of variables involved in the Boolean expression.

The second experiment is highlighted in Fig. 8. For each iteration, Fig. 8 shows the total number of configurations that can satisfy a set of Boolean equations (total configurations, TC) and the number of devices involved in the construction of Boolean equations (Dev). Each cell contains a combination like $TC_{\{Dev\}}$. The clusters of unsolvable problems ($TC = 0$) depicted in Fig. 8 correspond to incompatible user requests, such as air flow and security.

The applicability of the proposed approach to Boolean application domain depends upon the robustness of the Sat4j solver. Sat4j is a mature, open-source library providing access to SAT-related technologies to Java programmers. While the core SAT engine may not be competitive against commercial SAT solvers, the results of the library on pseudo-boolean problems are reasonable (Le Berre and Parrain, 2010). From the experiments it can be observed that the solver can handle hundreds of domotic effects in few milliseconds. In fact, the number of domotic effects needed for homes and small buildings will be in hundreds and the Sat4j solver will be robust enough to solve Boolean expressions in near real-time. However, for large facilities the DE framework may require commercial SAT solvers (since domotic effects may be in thousands) or other approaches such as problem partitioning. This aspect needs to be investigated further.

6 Related Work

Garcia-Herranz et al. (García et al, 2010) proposes an application-independent indirect control programming

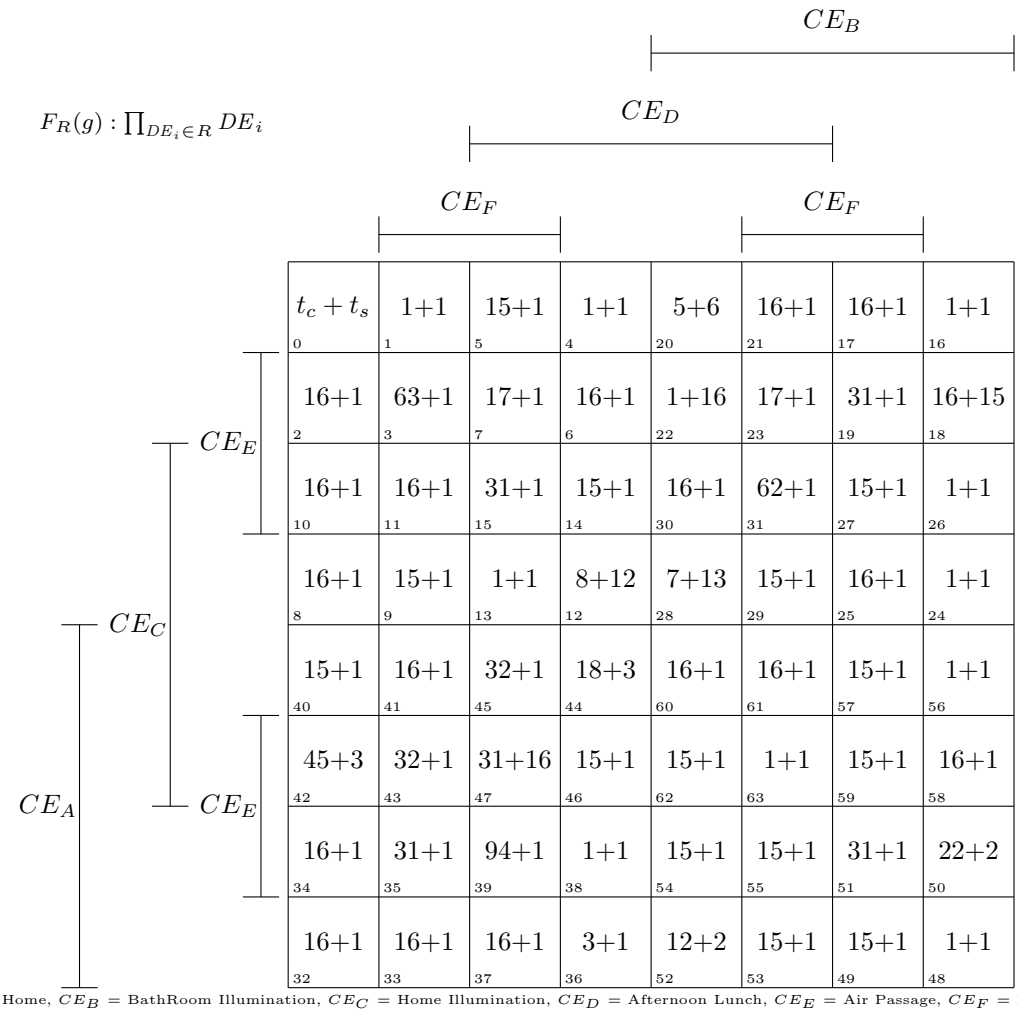

Fig. 7 CPU time measurements (in ms, $t_c + t_s$)

system to program complex behaviors yet simple enough to enable novice users to program their smart environments. The objective is to allow users to create powerful and personal behavior without expert assistance. They developed a rule-based language for a modular agent system (García et al, 2008). The rules allow expressing behaviors of type "When *triggers*, if *conditions*, then *action*". The rule language lacks the flexibility of providing different courses of action to achieve a solution. Moreover, it does not provide abstraction to allow AmI designers to develop techniques independent of devices.

Katasonov (Katasonov, 2010) motivates to build 'Digital fluency' in smart environments by enabling the non programmers to design, create and modify their smart environments. The paper proposes a higher level of abstraction in application design, on-the-fly development, flexibility with respect to adding new devices and software components. To build higher level of abstraction, an ontology that contains the hierarchy of tasks at the higher level is needed. The paper mentions defining tasks and their corresponding subtasks, without providing the organization of tasks in ontologies and the mechanism to achieve tasks. Our proposed solution in this paper not only provides details of organizing abstract goals but also provides mechanism to achieving those goals.

D-HTN (Amigoni et al, 2005) is a planning system for AmI applications, based on the hierarchical task network (HTN) approach, that is able to find courses of actions to address given goals. It combines concepts of both centralized planning and distributed planning in agent theory but the language ('Task network' (Erol et al, 1996)) that is used to store goals and their courses of actions is static. Our proposed solution also provides a hierarchical structure to store goals and their courses of actions but allows AmI designers to define their own language of translation.

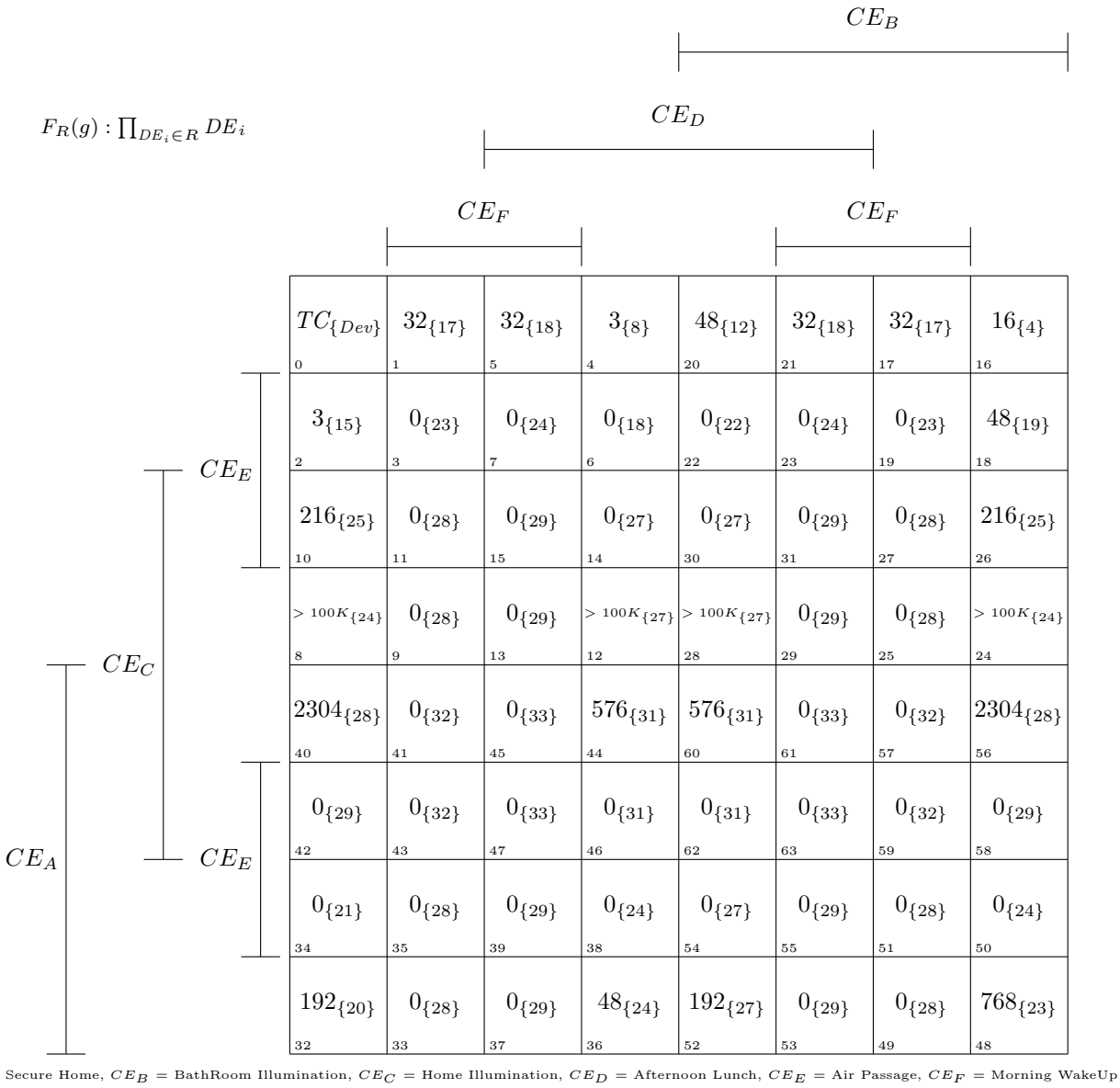

Fig. 8 Number of solutions TC and involved devices Dev

In (Kawsar et al, 2008) an Artefact framework is proposed which allows end users to deploy ubicomp systems easily in a Do-it-yourself fashion. Secondly, it allows developers to write applications and to build augmented artefacts in a generic manner. The Artefact framework provides a layered architecture where basic artefact functionalities are combined in a core component. Additional augmented features can be added as plugins into the core. Each augmented feature is called a profile. Each profile defines a specific functionality and implements the underlying logic of the functions, e.g., room temperature, lamp brightness. Though the profiles provide abstraction to hide the heterogeneity of the underlying devices, their functionality corresponds to the functionalities of devices and lacks the focus of providing a more generic goal that the user might wish to achieve.

Rashidi et al. (Rashidi and Cook, 2009) proposes a software architecture which incorporates learning techniques to discover patterns in resident's daily activities. The activity pattern are observed by monitoring the changes of states in different devices around the house. After discovering an activity pattern, it stores the activity pattern and its related temporal knowledge in a Hierarchical activity model (HAM). HAM captures the temporal relationships between events in an activity by explicitly representing sequence orders in a tree structure containing Markov chains at the bottom level. The

activities are stored based on individual devices in the house which does not allow observers to see the bigger picture at higher level of user goal. Though currently our solution does not employ learning patterns to automatically extract repetitive tasks but it can easily be employed in our proposed organization of Domotic Effects. Moreover, observing patterns at an abstract level can give a more clear picture of user's intentions instead of focusing on individual device or chain of devices.

Cheng et al. (Cheng et al, 2009) proposes a smart homes reasoning system called ASBR system. The system learns user's preferences by adaptive history scenarios and put forwards a way to rebuild reasoned knowledge in other smart homes. They proposed that contextual information can be extracted and reasoned as a set of scenarios. In addition, the system can derive personalized habits and store them in OWL files. They do not provide an organization mechanism and the concept of scenario is different from our proposed effect. Effects are different from scenarios as it is not a storage of historical events or repetitive tasks. Though repetitive tasks can be mapped onto effects, our approach provides complete control to the residents to define their own abstract level control which are ultimately resolvable to a set of devices in certain states. The effect based approach is designed to be extensible to smart environments in general.

Dey et al. (Dey et al, 1999) proposes a software infrastructure solution to detect the current states of the environment (called Context) and take action based on it. The infrastructure is focused on developing context aware applications. Though the concept of Domotic Effect can be used to monitor the current state of the environment using the process of Effect Evaluation (Corno and Razzak, 2012), the focus of this paper in particular is to enforce generic goals on the environment. Our modeling allows to handle both tasks in a more simple manner. Moreover, currently the enforcement implementation focuses on Boolean application domain states, but the DogEffects ontology can be used to monitor devices with continuous states, which was described as the limitation of the infrastructure in (Dey et al, 1999). The Effect Enforcement implementation is designed with the extensibility in mind, which is missing in (Dey et al, 1999). Generally, context represents the knowledge of external conditions and their complexities in the environment. This knowledge is used in some way to make particular action(s) choice. The question of how the action(s) are actually performed is not part of the context but rather is a characteristic of the system that handles the environment. Our paper focuses on this internal characteristic of the system rather than collecting conditions that triggered those actions.

Reference (Kaldeli et al, 2010) proposes a middleware architecture for smart home systems. The architecture has pervasive, composition and user layers. The composition layer contains a CSP (Constraint Satisfaction Problem) based planner (Kaldeli et al, 2009) which computes a plan, that is, a sequence of actions that need to be applied in order to satisfy a user's goal. The goals are pre-defined in a declarative manner. Our approach does employ declarative manner to describe goals called domotic effects but it is more flexible as it allows the AmI designers to define operators to manage combinations according to environments and the user's can define their own domotic effects based on those operator. Moreover, the planner in (Kaldeli et al, 2010) takes time in seconds to construct a problem and determine results, whereas the effect enforcement module takes time in milliseconds for both construction of set of Boolean equations and finding a solution.

In (Heider and Kirste, 2002; Encarnaçao and Kirste, 2005) a goal based interaction has been proposed, and extended in (Hellenschmidt, 2005), that takes a user's goal and finds a path achieving the goal. The use of propositional calculus is advocated, however, unlike our paper, (Hellenschmidt, 2005) lacks implementation details, i.e., a proper mapping from user's goal to propositional calculus and its interpretation. To support goal based interaction (Heider and Kirste, 2002; Encarnaçao and Kirste, 2005; Hellenschmidt, 2005) advocate that each device in the environment implements an event processing pipeline consisting of user interface, control application and actuators. Moreover, they make an assumption that each device shares data inside event processing pipelines across all the devices present in the environment, creating a SodaPop (Self-Organizing Dataflow Architecture suPporting Ontology-based problem DecomPosition) model. The aim of SodaPop model is to supports self-organization of its devices by using adhoc cooperation of distributed device ensembles. In real world, an environment comprises devices from several different and competing vendors which may not be willing to expose to other vendors internally stored information of their devices, or may not have enough computing capabilities. Domotic Effects are a more centralized approach, where all relevant information about devices is available in the automation gateway and no requirements are imposed onto the devices, thus providing easy and immediate interoperability with existing devices from different vendors.

Domotic Effects provides the end user, the ability to personalize a smart environment, as well as allows AMI application designers to design, develop and manage based on a higher level of abstraction. While (Rashidi and Cook, 2009; Cheng et al, 2009) proposes a com-

plete independent control solution based on the learning pattern of a user's activity, the (García et al, 2010; Katasonov, 2010; Amigoni et al, 2005) advocates enabling non programmers to create and manage their smart environments according to their wishes. Providing a complete independent control extracted from a user's activity might not be a very good idea as it does not allow people to program their personal spaces. It also raises a new set of problems like privacy and security issues. Instead of focusing on user's goals or intentions, it focuses on a set of devices and their activity. It can be said that learning algorithms discover patterns of device activity instead of user's intention and activity. The latter, on the other hand provide end user programming environment but underlying structure of organizing goals and their different courses of actions is missing. Though (Amigoni et al, 2005; Kawsar et al, 2008; Heider and Kirste, 2002) provide goal based interaction mechanism, they lacks the flexibility, separate views of development for system designers and users and applicability to different application domains.

7 Conclusion

This paper presented a high level approach, based on the concept of Domotic Effects for modeling and satisfying user requests in complex smart environments. The Domotic Effects framework, based on the DogEffects ontology, is general, and extensible, and is easy to customize. In particular, this paper focuses on control and monitoring applications, where high level effects may be described resorting to Boolean expressions defined on device states.

The paper presented experimental examples of Simple and Complex Effects over a sample home environment setup, and shows experimental results that prove that high-level user requests are satisfied in less than 100 ms, thanks to the mapping of the request into a SAT problem that may be efficiently solved.

The DE framework has a wider scope of objectives. In the future, the role of monitoring high-level state of environment (in Boolean or Real domain) using domotic effect will be investigated. Moreover, the enforcement approach for other application domains like Energy Saving will be analyzed.

References

Amigoni F, Gatti N, Pinciroli C, Roveri M (2005) What planner for ambient intelligence applications? IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans 35(1):7–21

Bader S, Dyrba M (2011) Goalaviour-based control of heterogeneous and distributed smart environments. In: 7th International Conference on Intelligent Environments (IE), 2011, pp 142–148

Bonino D, Castellina E, Corno F (2008) The DOG gateway: enabling ontology-based intelligent domotic environments. IEEE Transactions on Consumer Electronics 54(4):1656–1664

Bonino D, Corno F (2008b) Dogont - ontology modeling for intelligent domotic environments. In: Sheth A, Staab S, Dean M, Paolucci M, Maynard D, Finin T, Thirunarayan K (eds) The Semantic Web - ISWC 2008, Lecture Notes in Computer Science, vol 5318, Springer Berlin / Heidelberg, pp 790–803

Chen L, Hoey J, Nugent CD, Cook DJ, Yu Z (2012) Sensor-based activity recognition. IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews 42(6):790–808

Cheng ST, Wang CH, Chen CC (2009) An adaptive scenario based reasoning system cross smart houses. In: 9th International Symposium on Communications and Information Technology, 2009, IEEE, pp 549–554

Corno F, Razzak F (2012) Intelligent energy optimization for user intelligible goals in smart home environments. IEEE Transactions on Smart Grid 3(4):2128–2135

Cook S.A (1971) The complexity of theorem-proving procedures. In: ACM Proceedings of the third annual ACM symposium on Theory of computing, 1971, ACM, pp 151–158

Dey A, Abowd G, Salber D (1999) A context-based infrastructure for smart environments. In: 1st International Workshop on Managing Interaction in Smart Environments, Springer, pp 114–128

Ducatel K, Bogdanowicz M, Scapolo F, Leijten J, Burgelman JC (2003) Ambient intelligence: From vision to reality. IST Advisory Group Draft Rep., European Commission

Encarnaçao J, Kirste T (2005) Ambient intelligence: Towards smart appliance ensembles. From Integrated Publication and Information Systems to Information and Knowledge Environments pp 261–270

Erol K, Hendler J,Nau DS (1996) Complexity results for HTN planning. Annals of Mathematics and Artificial Intelligence 18(1):69–93

Heider T, Kirste T (2002) Supporting goal-based interaction with dynamic intelligent environments. In: ECAI, Fraunhofer Publica (Germany), pp 596–602

Hellenschmidt M (2005) Distributed implementation of a self-organizing decentralized multimedia appliance middleware. In: Davies N, Kirste T, Schumann H (eds) Mobile Computing and Ambient Intelligence: The Challenge of Multimedia, IBFI, Germany,

Dagstuhl, Germany, no. 05181 in Dagstuhl Seminar Proceedings

Kaldeli E, Lazovik A, Aiello M(2010) Extended goals for composing services. In: Proceedings of the 19th International Conference on Automated Planning and Scheduling, pp 19–23

Kaldeli E, Warriach E, Bresser J, Lazovik A, Aiello M (2010) Interoperation, composition and simulation of services at home. In: Maglio P, Weske M, Yang J Fantinato M, (eds) Service-Oriented Computing, Springer LNCS, vol 6470, pp 167–181

Katasonov A (2010) Enabling non-programmers to develop smart environment applications. In: IEEE Symposium on Computers and Communications (ISCC), IEEE, pp 1059–1064

Kawsar F, Nakajima T, Fujinami K (2008) Deploy spontaneously: supporting end-users in building and enhancing a smart home. In: Proceedings of the 10th international conference on Ubiquitous computing, ACM, pp 282–291

Le Berre D, Parrain A (2010) The Sat4j library, release 2.2 system description. Journal on Satisfiability, Boolean Modeling and Computation 7:59–64

García-Herranz M,Haya PA, Esquivel A,Montoro G, Alamán X (2008) Easing the smart home: Semi-automatic adaptation in perceptive environments. Journal of Universal Computer Science 14(9):1529–1544

García-Herranz M, Haya P,Alamán X(2010) Towards a ubiquitous end-user programming system for smart spaces. Journal of Universal Computer Science 16(12):1633–1649

OSGI Alliance (2003) Osgi service platform, release 3. IOS Press, Inc.

Rashidi P, Cook D (2009) Keeping the resident in the loop: Adapting the smart home to the user. IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans 39(5):949–959

Razzak F (2013) The Role of Semantic Web Technologies in Smart Environments. PhD Dissertation, Politecnico di Torino, Italy http://porto.polito.it/id/eprint/2506366

Weiser M (1995) The computer for the 21st century. Scientific American 272(3):78–89