Agent-based Negotiation Approach for Feature Interactions in Smart Home Systems using Calculus of the Context-aware Ambient

Ahmed S. Alfakeeh[1], Ali H. Al-Bayatti[2], Francois Siewe[2], and Thar Baker[3]

[1]Faculty of Computing and Information Technology, King Abdulaziz University,
Email:asalfakeeh@kau.edu.sa
[2]Faculty of Computing, Engineering ad Media, De Montfort University University,
Email:alihmohd@dmu.ac.uk
[2]Faculty of Computing, Engineering ad Media, De Montfort University University,
Email:fsiewe@dmu.ac.uk
[3]Department of Computer Science, Liverpool John Moores University
Email:t.baker@ljmu.ac.uk

September 17, 2019

Abstract

Smart Home Systems (SHSs) provide several services which are tailored to different residents' preferences. As a result, SHSs are highly exposed to undesirable interactions, known as feature interactions (FIs). FIs might occur as a result of a conflict in services' goals or a conflict with residents' preferences. Previous studies have proposed solutions based on applying priorities, in which some services or preferable features are disabled in favour of other services. Alternatively, the agent-based negotiation approach (ABNA) utilises agents and applies negotiation, enabling services with contrary features to work simultaneously. ABNA avoids applying priority between services or house residents' preferences whenever a space for a compromise exists. The mechanism of ABNA is based on the use of a hierarchy of features based on their contribution to the function of the service or on the importance of these features to house residents. To achieve a compromise between conflicting services, ABNA models services and residents by using agents, and implements a negotiation algorithm that allows services with conflicting features to work simultaneously. This paper presents a description of ABNA with a formal specification of ABNA in the Calculus of Context-aware Ambient (*CCA*). This enables the formal analysis of ABNA by using the execution environment of *CCA*.

Keywords: feature interactions, smart home systems, negotiation, agents, *CCA*

1 INTRODUCTION

Software systems supporting several services are exposed to undesirable interactions which may affect the functionality of the overall system [1]. This problem is common in telecommunication systems and it is known as Features Interactions (FIs) [2] [23]. The root causes of FIs are conflicting goals, competition for shared resources and a lack of inference between services [3]. The problem on FIs is particularly challenging in Smart Homes Systems (SHSs) because of the dynamic environment implementation involved and the multitude of services provided to several residents according to individual preferences [4] [24]. In the SHS domain, there are two types of interactions: policy- and service-based interactions. The former type is a resident-defined instruction which modifies the behaviour of the system [5]. Unlike previous approaches for detecting and resolving FIs in SHSs, the agent-based negotiation approach (ABNA) is based on a reaching a compromise, whenever possible. In order for ABNA to operate successfully, the components of SHSs i.e. services and residents, will be modelled as agents. Each time an agent wants to control a device, he/she is involved in negotiation sessions to reach a compromise and avoid any possible undesirable interactions. Priority will be applied only if no possible compromise can be achieved. In this paper, a formal specification of ABNA is presented. The formalisation is carried out using the Calculus of Context-aware Ambient (*CCA*). The main features of *CCA* include concurrency, context-awareness and mobility. What is more, *CCA* specifications are executable and enable rapid prototyping [6]. The main contributions of this article are as follows:

- A formal specification of ABNA using *CCA* is given. This enables the formal analysis of ABNA by using the execution environment of *CCA*
- The pragmatics of the proposed ABNA is demonstrated using an SHS case study
- The correctness of the ABNA is analysed using ccaPL, the execution environment of *CCA*

2 RELATED WORK

Many researchers have investigated FI issues within SHSs, Internet of Things (IoT) and other related disciplines. [25] [26] [27] discussed the issue of services possibly in conflict with other services, as they might share the same actual device and/or the same field. These studies have suggested using a pre-defined framework which clarifies the roles and responsibilities of each agent in the parent–child structure. Some of the other proposed solutions for FI detection and resolution in SHSs are implemented during the design phase of the system [12].

One off-line detection approach is a service-centric framework for FIs in National Health Service (NHS) integrated services [14]. It proposes revealing

all potential interactions within home network systems. The approach use an off-line technique, which is performed before the deployment of the system, even though most FI problems can be discovered only at run-time. Additionally, they consider a wide scope of unlikely FI which would not normally appear in a real run.

What is more, an off-line technique does not support feature scalability. As mentioned above, the limitation of the off-line technique is that it is performed before the deployment of the system, therefore missing problems that occur at run-time. Using only the off-line technique to solve FI problems is insufficient. In response to this issue, some approaches use an on-line technique. One such example is the Policy Interactions Manager Module (PIMM), proposed to extend the traditional networking systems which use KNX communications standards for home and building automation [15]. PIMM is integrated with the KNX home network as a part of the engineering tool software suite to work as a run-time interactions manager for detecting and resolving any undesirable interactions.

Similarly, a proposed formal method approach, the semantic web-based policy interaction detection method (SPIDER) [16], and a semi-formal method known as identifying requirements interactions (IRIS) [17], have been proposed to detect and resolve FIs in SHSs. However, PIMM, SPIDER and IRIS can only solve feature interactions that take place as a result of policy-based conflicts and not service- or device-based conflicts. In the same sequence, the work in [18] uses the belief-desire-intention (BDI) agent model also to solve conflicts which are a result of the differences in inhabitants' desires, i.e. policy based. The approach keeps alternative proposals in a library, allowing the agent, with minimal risk to choose an alternative proposal in case of conflict. Although the negotiation between agents is used in this study [18], only the agent with a high priority, referred to as the maximum risk agent, will have the advantage. In other words, there is no compromise allowing conflicting agents to work simultaneously. Likewise, the entity-relationship (ER) model is presented in [19]. The model creates a corresponding domain which captures the relevant context information to determine the conflict between occupants'preferences. Besides not covering service-based conflicts, the ER model resolves only conflicts related to user comfort, such as light, sound and temperature.

Another study introduces the notion of resource locking [7]. In this approach, the access attribute of each device is set to not shared (NS) or shared (S). NS means a device can only be used by one service at any one time, and S means multiple services can control the device concurrently. The resource locking approach utilise a three-layer architecture service, device and environment to present a clear and helpful taxonomy of FIs. It works as a feature manager and is implemented as a service on the OSGi platform to detect and resolve any FIs. However, this approach is not flexible. The notion of resource locking requires the execution or non-execution of features; in other words, there is no way for features to reach a compromise. Moreover, it prevents interactions at the device level and not at the service level which is the main concern of SHSs. Additionally, looping interactions in which two or more services go on an in-

finite loop because the activation of one of them leads unintentionally to the activation of another and vice versa are not resolved [7].

A related work in [13] introduced a millimeter lightwave communication between multiple devices (D2D). In this work, the authors focused on so-called future generation mobile networks to reduce propagation and latency were devices communicating with each other through base stations can achieve the proposed goal. On the other hand, the work in [20] covers both off-line and on-line FI. It models each appliance as an object consisting of properties and methods in an object-oriented fashion. Although our study is inspired by the notion of negotiation and compromise, which are mentioned in such a work, the proposed solution does not target FIs, which are caused by residents' policies; the scheme to resolve on-line FIs is also not clarified. The negotiation agent approach is first used in the telecommunications domain to resolve FIs [21]. The proposed system was able to detect and resolve FIs at run-time (on-line) automatically without the need for users or service providers to know about other users' or service providers' features or settings. However, the proposed solution is not applicable to the smart home domain because SHSs are more complicated and have many features, and they are implemented in a dynamic environment in which they have to respond to a variety of contexts. From the discussion above, the works which have been done so far for FI detection and resolution in SHSs are grouped into three categories, namely off-line, on-line and both (off-line and on-line together). The mechanisms used vary, and most of the works target only one type of FI, i.e. service interaction [7] or policy interaction [15, 16, 18, 19].

Although, some works target both interactions [22], a negotiation approach to reach a compromise has not been adopted yet. The significance of the approach is that it enables services with conflicting features or contrary resident policies to work simultaneously.

3 OVERVIEW OF ABNA

Most of the services within SHSs could be broken down into simple features [7, 8]. If we take the home cinema as an example, one of its functions is to play films and adjust the room settings in order to create an ideal atmosphere for watching a film. When this service is activated, the following features or processes need to be carried out in order to achieve these goals [9]:

1. Switching on the TV
2. Choosing the appropriate TV channel or movie service (e.g. Netflix)
3. Setting the speaker volume to 70
4. Dimming the lights to 10
5. Pulling the curtains with smart curtain motor
6. Closing the door

It is important to note that from the above-listed features, there is a hierarchy of priorities, in which it is essential to switch on the TV, while closing the door is less important. Despite the key differences in service features, for an optimum experience, all features should be present.

Commonly, FIs occur when more than one service need to function or more than one resident are making requests. Hence, there are potential undesirable interactions (conflicts), such as multiple services using the same device, multiple users using the same service or multiple users using multiple services. In each of these cases, an undesirable interaction may occur between the features of the involved services.

Unlike other proposed solutions [12, 13, 14, 15, 16, 17], the aim of ABNA is not to prioritise one service over another or one resident over another, but rather to reach a compromise by allowing two conflicting services to run simultaneously with slightly less efficiency (without affecting the overall performance). This is more desirable than totally disabling one service or overriding a resident's in favour of another. The ABNA mechanism for SHS detection and resolution in SHSs is based on establishing a negotiation system. In order for the negotiation system to function, the ABNA utilises agents, enabling services with contrary features to work simultaneously. The components of SHSs i.e. services and residents, will be modelled as agents in addition to the negotiator (Figure 1).

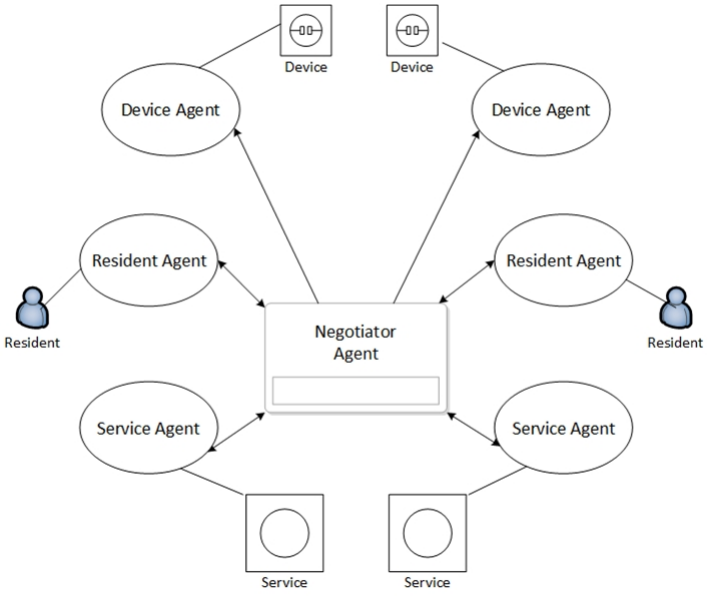

Figure 1: The structure of ABNA

Each agent registers features or preferences (known as tasks) in a hierarchical order, in which tasks with a high priority will be at the top, whereas the least important tasks are at the bottom (Figure 2). Furthermore, each task will be

assigned as allowing a compromise or not. Tasks which are necessary for an agent will be assigned as allowing a compromise, whereas tasks which are not necessary but preferable will be assigned as allowing a compromise.

This is important to enable two or more conflicting agents to reach compromise by enabling them to operate simultaneously. The compromise will enable agents, first to reach a middle point of performance, such as setting the light illumination to 40 when one agent wants to dim the light to 20 and another wants to dim it to 60; and second, to allow an agent to take advantage of performing a task when the task is a priority for itself whereas it is preferable but not necessary for the other agent. For example, it is necessary for ventilation service to switch on the ventilation fan in order to refresh the air, but it is only preferable and not necessary for the air conditioning service to switch off the ventilation fan in order to keep the room temperature at certain a degree.

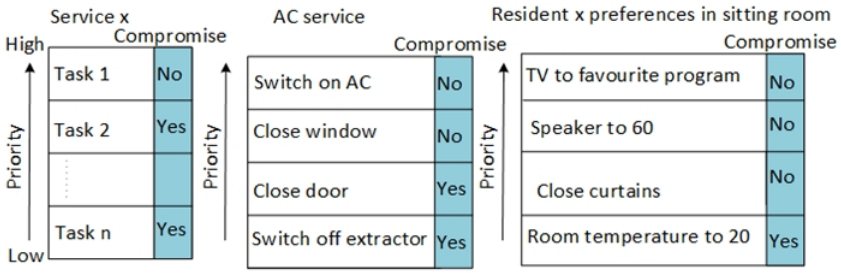

Figure 2: Hierarchy of features [9]

ABNA in SHSs sets a negotiation session between concerned agents every time a task needs to be carried out. The main goal of the negotiation process is to detect and resolve any conflicts, allowing agents to operate simultaneously. The negotiation session involves the initiating agent, i.e. the agent who needs to perform a specific task, the negotiator agent and the involved agents. The involved agents are any other agents who utilise the targeted device or could affect the performed task. For example, on a warm day, in order to save energy, the power management service agent sends a task in the form of a proposal to open a window, allowing breeze to lower the room temperature instead of activating the air conditioning. The negotiator, using a register of all agents who utilise the window controller device, such as security, safety, heating, ventilation, air conditioning and lighting, in addition to the power management service agent itself, will send the proposal to each of these services to request their approval.

Each involved agent will evaluate the received proposal and then respond to the negotiator, indicating whether the proposal could be accepted, could allow a compromise, could be counter-proposed or could be rejected.

Each proposal consists of three components, namely the device ID, the action to be performed and the value to describe the action degree. Hence, the proposal in our previous example is (window, set, 100), in which 100 is the value of the

action ‘set’ to make the window fully open (the value 0 is used to make it completely closed).

The counter-proposal is the option if there is no possibility to accept the original proposal or to allow a compromise on it. As explained earlier, tasks are organised in a hierarchy, so the counter-proposal will be the task with the lowest priority or for which a compromise can be made. For illustration, we consider the above example when a proposal is sent from the power management service to open a window. If the security service is active and one of its priority features is to keep the window closed, then it will not approve the received proposal; instead, it will send a counter-proposal.

The counter-proposal will be a feature that is not a priority for the security service, such as closing curtains because this feature will help reduce the room temperature. Before considering the counter-proposal, the initiating agent will check if it has priority over the involved agent by consulting the negotiator accordingly. If the initiating agent has priority, the counter-proposal will be rejected; otherwise, it will be accepted.

A proposal will be rejected only if there is no possibility for it to be accepted or for a compromise to be achieved and there is also no counter-proposal to be made. The sequence diagram (Figure 3) shows the negotiation process in ABNA.

Although ABNA aims to avoid utilising priority as much as possible, there are some situations in which the negotiator has to decide which agent has priority in controlling a device. Specifically, these situations occur when a counter-proposal or the original proposal is rejected. To determine priority, the negotiator has a register that shows the priority of each agent, whether it is the service agent or the resident agent.

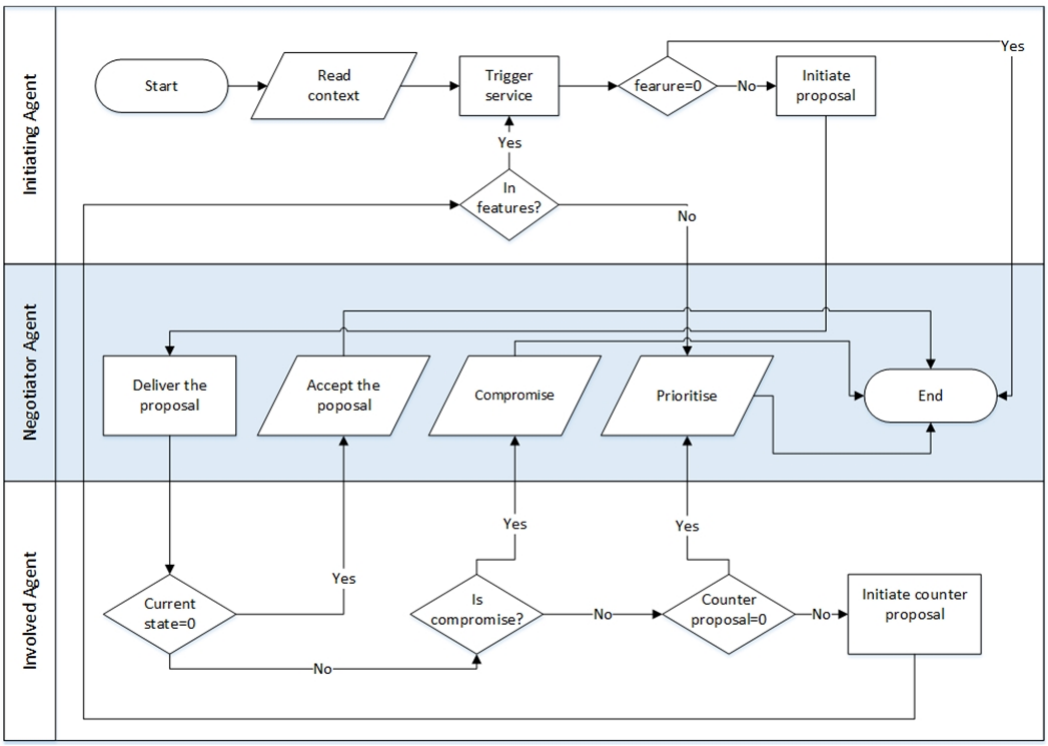

Figure 3: Negotiation algorithm

4 OVERVIEW OF *CCA*

CCA was proposed in [10] as a process calculus for modelling mobile systems that are context-aware. It builds upon a previous calculus known as Mobile Ambients [11] whilst introducing new ideas. It enables ambient (e.g., software, devices, locations) and processes to have an awareness of the conditions and context in which they are executed. The resulting process calculus is flexible and powerful, emphasising context awareness and mobility. An ambient is an abstraction of a bounded place where computation happens. An ambient can be mobile, and, can communicate with peers and be nested inside another ambient. In this section, we present the syntax (Table 1) and informal semantics of the calculus. Because of space constrains, we refer readers to [10] for the formal semantics of *CCA*. We shall define four syntactic categories with *CCA*. These are processes P, capabilities M, locations α and context expressions (CEs) κ.

Names are always written in lowercase letters, e.g. n, x and x. A list of names is denoted by $\tilde{y}$, and $|\tilde{y}|$ represents the size of the list.

Table 1: SYNTAX OF *CCA*

P, Q	$:=$	$0 \mid n\,[P] \mid (vn)P \mid !P \mid \kappa?M.P$
M	$:=$	$\text{in}\ n \mid \text{out}\mid \alpha\ x\,(\tilde{y}) \mid \alpha\ \text{recv}(\tilde{y}) \mid \alpha\ \text{send}(\tilde{y})$
α	$:=$	$\uparrow\mid n\uparrow\mid\downarrow\mid n\downarrow\mid::\mid n::\mid \epsilon$
κ	$:=$	$\text{true}\mid \bullet \mid n = m \mid \neg\kappa \mid \kappa_1\mid \kappa_2\mid \kappa_1\wedge\kappa_2\mid new\ n, \kappa \mid \bigoplus \kappa \mid \quad \kappa$

Processes: The process 0 terminates immediately. If two processes P and Q are running in parallel, this is denoted by $P \mid Q$. To limit the scope of a name, the following notation is used: $(vn)P$, indicating that the scope of n is limited to P. The replication $!P$ denotes a process that can recreate a copy of itself whenever needed, i.e. $!P \equiv P|!P$. The process $n[P]$ represents an ambient named n whose behaviour is described by the process P.

A CE κ denotes the situation that must be met by the environment of the executing process. The context-guarded prefix $\kappa?M.P$ is a process that waits until the environment satisfies the CE κ, and then executes the capability M and continues similar to the process P.

Locations: The location α can $\uparrow$ to indicate any parent, $n\uparrow$ for a definite parent ambient named n, $\downarrow$ for any child, $n\downarrow$ for a definite child ambient named n, $::$ refers to any sibling and $n::$ signifies a specific sibling ambient named n. The symbol ϵ (empty string) refers to the current ambient.

Capabilities: There are two mobility capabilities defined in *CCA [10]*, which make it possible for an ambient to move in its environment. These are 'in' and 'out'. An ambient can execute the capability 'in n' to move into a sibling ambient named n, and the capability out allows an ambient to move out of its parent ambient. Ambients can send and receive messages. Using the capability α send $(\tilde{y})$, an ambient can send a list of names $\tilde{y}$ to a location α. An ambient can execute the capability α recv $(\tilde{y})$ to receive in the variables $\tilde{y}$ a set of names sent from the location α.

Context-expressions: In *CCA*, a context is modelled as a process with a hole in it. The hole (denoted by $\odot$) in a context represents the position of the process that context is the context of. For example, suppose a system is modelled by the process $P|n[Q|m[R|S]]$. Therefore, the context of the process R in that system is $P|n[Q|m[\odot|S]]$, and that of the ambient named m is $P|n[Q|\odot]$. The properties of contexts are called CEs.

CE holds true for all contexts. A CE $n = m$ holds if the names n and m are lexically identical. The CE $\bullet$ holds solely for the whole context, i.e. the position of the process evaluating that CE. Propositional operators, such as negation $(\neg)$ and conjunction $(\wedge)$, expand their usual semantics to CEs. A CE $\kappa_1 \mid \kappa_2$ holds for a context if that context is a parallel composition of two contexts. A CE $n[\kappa]$ holds for a context if that context is an ambient named n such that κ holds inside that ambient.

A CE $\bigoplus \kappa$ holds for a context, on the condition that the context has a child context for which κ holds. For a CE $\quad \kappa$ to hold for a context there must be a sub-context present, somewhere in that context for which κ holds. The operator $\quad$ is known as *somewhere modality* whereas $\bigoplus$ is known as *spatial next modality*

[6].

5 FORMALISING ABNA IN *CCA*

In this section, we give a formalisation of ABNA by using the mathematical notation of *CCA*. As stated in the sequence diagram (Figure 3), there are three main agents that exchange messages during the negotiation process namely the initiating agent (iniAgent), the negotiator agent (negAgent) and the involved agent (invAgent). Each these above agents will be modelled as an ambient. Because of the significant role of the negotiator, during the negotiation session, there will be three extra ambients, i.e. delivering, compromising and prioritising, which work as children ambients. Therefore, the formal specification of ABNA is given by the following *CCA* process:

iniAgent [P$_{iniAgent}$]|$invAgent[P_{invAgent}]$|$device[P_{device}]$
| negAgent [P$_{negAgent}$|$delivering[P_{delivering}]$
| compromising [P$_{compromising}$]
| prioritising [P$_{prioritising}$]]

Table 2: Constants

Constants	
Notation	**Description**
accept	Accept proposal
reject	Reject proposal
compromise	Allow compromise on tasks
prioritise	Give priority
counterProposal	Place counterproposal
on	Switch on device
off	Switch off device
set	Set device to a certain value
sec_Agent	Security agent
saf_Agent	Safety agent
vent_Agent	Ventilation agent
AC_Agent	Air conditioning agent
ent_Agent	Entertainment agent
pwr_Agent	Power management agent

The specifications of these ambients are presented in the following subsections. We first give a convention of the notations used in these specifications, which include the constants (Table 2) and the variable symbols (Table 3).

Table 3: Variables

Variables		
Notation	Description	Values
r	Reply to the proposal	accept, reject, compromise, prioritise and counterpropsal
v1	Value of performance for the initiating agent	1,2,...,100
v2	Value of performance for the involved agent	1,2,...,100
v3	Compromised value	1,2,...,100
d	Device ID	window1, window2, door, light1, curtain1, AC, TV, ventilation_fan,..
a	Action to be performed	on, off, set
m	Master agent, which has priority	sec_Agent ,saf_agent, AC_Agent, vent_Agent,ent_Agent,pwr_Agent
iniAgent	Initiating agent	sec_Agent ,saf_agent, AC_Agent, vent_Agent,ent_Agent,pwr_Agent
invAgent	Involved agent	sec_Agent ,saf_agent, AC_Agent, vent_Agent,ent_Agent,pwr_Agent,none

A- *iniAgent* ambient

This ambient is responsible for submitting a proposal to negAgent to perform a task based on the triggering context. iniAgent then waits for the reply to its proposal, which could be accepted, rejected or counter proposal. If the reply is the latter, this ambient will check and then accept the counter-proposal only if the following is achieved:

1. The priority is for the involved agent.
2. The counter-proposal is one of the features that exist in the queue to be performed.

This behaviour is modelled as follows: //
$P_{iniAgent} \hat{=} negAgent :: send(sec\_Agent, window1, set, 0).$
$negAgent :: recv(r).(r = counterProposal)?negAgent :: recv(m).$
$(\neg(m = sec\_Agent))?negAgent :: recv(d, a, v1).\{$
$((a = set\ and\ d = window2\ and\ v1 = 0)\ or$
$(a = set\ and\ d = door\ and\ v1 = 0))?negAgent :: send(accept).0$

The symbols used here are as follows:

- d is the ID of the targeted device
- a is the action to be performed (on, off, set)

- $v1$ always follows the action "set" to describe the degree or performance level for the targeted device, such as speakers and lights.
- r is the reply received to the proposal
- m is the agent with the highest priority

B- *negAgent* ambient

This ambient is the negotiation master and has three child ambients, namely delivering, compromising and prioritising. After receiving a proposal from the iniAgent, it will deliver it to the other involved agents and then take action based on the reply, either accepting the proposal or compromising if there is an agreement between the agents involved; otherwise, it applies priority.

The behaviour of this ambient is specified as follows:

$P_{negAgent} \hat{=} !sec\_Agent :: recv(sec\_Agent, window1, set, 0).deleviring\downarrow$
$send(sec\_Agent, window1, set, 0).delivering \downarrow recv(invAgent).\{$
$(invAgent = none)?window1 :: send(set, 0).sec\_Agent :: send(accepted).0$
$|\neg(invAgent = none)?pwr\_Agent :: send(window1, set, 0).pwr\_Agent ::$
$recv(compromise, 100).\{$
$(r = accept)?window1 :: send(set, 0).sec\_Agent :: send(accepted).0$
$|(r = compromise)?compromising \downarrow send(window1, set, 0, 100).$
$compromising \downarrow recv(v3).sec\_Agent :: send(compromised).0$
$|(r = counterProposal)?invAgent :: recv(window2, set, 100).$
$sec\_Agent :: send(counterProposal).$
$sec\_Agent :: send(window2, set, 0).sec\_Agent :: recv(r).$
$(r = accept)?window2 :: send(set, 0).0$
$|(r = reject)?prioritising \downarrow send(sec\_Agent, pwr\_Agent).prioritising \downarrow$
$recv(m).iniAgent :: send(m).(m = sec\_gent)?window1 :: send(set, 0).0$
$\}\}$

C- delivering ambient

When a proposal is received by negAgent, the delivering ambient has to determine the involved agents who should be consulted in order for the proposal to be approved. To do this, it examines several conditions. A list of involved agents will then be sent to the negotiation agent.

The behaviour of the delivering ambient is specified as follows:

$P_{delivering} \hat{=} negAgent\, recv(iniAgent, d, a, v1).R,$

where

$R \hat{=} ((d = window1\ and\ a = set)\ or\ (d = window2\ and\ a = set))?$
$negAgent \uparrow send(pwr\_Agent).0|negAgent \uparrow send(none).0$

D- compromising ambient

If the reply from the invAgent ambient is a compromise, the compromising ambient will allow a compromise on the proposal to satisfy each party. To calculate the compromise level, the value which was proposed by the initiating agent, i.e. (0), will be added to the value proposed by the involved agent, i.e. (100), and the result will be divided by two. The final result, which is (50), will be sent to the negAgent.

The behaviour of this ambient is modelled as follows:

$P_{compromising} \hat{=} negAgent \downarrow recv(window1, set, v1, v2).\{$
$let\ v3 = (v1 + v2)/2\ in \uparrow send(v3).0\}$

E- prioritising ambient

If the invAgent ambient rejects the proposal, the prioritising ambient will give the advantage of controlling the device to the agent with priority. This means that if the initiating agent has the highest priority, the prioritising ambient will accept the proposal by implementing the proposal to control the targeted device according to the proposal parameters; otherwise, it will reject the proposal.

The behaviour of the prioritising ambient is modelled as follows:

$P_{prioritising} \hat{=} negAgent \uparrow recv(iniAgent, invAgent).S$

Where
$S \hat{=} \{\ (iniAgent = saf\_Agent\ or\ invAgent = saf\_Agent)? \uparrow send(saf\_Agent).0$
$|(iniAgent = sec\_Agent\ and\ invAgent = saf\_Agent)? \uparrow send(sec\_Agent).0$
$|\neg(iniAgent = sec\_Agent\ and\ invAgent = sec\_Agent)? \uparrow send(sec\_Agent).0\}$

F- invAgent ambient

This ambient is responsible for examining three conditions in order to decide whether to accept the proposal, allow a compromise on it, suggest a counterproposal or reject the proposal. The decision will be based on the current status of the involved ambient, i.e. 'off' or 'on'. The received proposal will be accepted if the current status id 'off'. However, if the current status is 'on', the involved ambient will not accept the proposal. Instead it will consider compromising or offerring a counter-proposal. After all, if there is no space to assist the initiating agent, the proposal will be rejected. To examine the current status of the involved agent, we use a memory cell [10].

The behaviour of invAgent ambient is specified as follows:
$P_{invAgent} \hat{=} negAgent :: recv(d, a).cur\_status \downarrow send().cur\_status \downarrow recv(status).P$

where

$P \hat{=} \{\ (\neg isON(status))?negAgent :: send(accept, 0).0$
$|(isON(status)\ and\ (d = window1\ and\ a = set\ and\ v1 = 0)\ or$
$(d = window2\ and\ a = 0))?negAgent :: send(compromise, 100).0$
$|(isON(status)\ and\ (d = air\_conditioner\ and\ a = on))?$
$negAgent :: send(counterProposal, 0).negAgent :: send(window2, set, 0).0\}$

G- device Ambient

This ambient represents the device that needs to be controlled. It simply receives the task from the negAgent ambient as an order. The task is subsequently performed according to the received action and value.

The behaviour of the device ambient is modelled as follows:

$P_{device} \hat{=} !negAgent :: recv(a, v1).0$

6 CASE STUDY

In this section, a case study is presented to illustrate the processes described in the above. This section is divided into two subsections: the first one will demonstrate the pragmatics of ABNA by using the case study, whereas the second one analyses the correctness of ABNA by using ccaPL.

6.1 Pragmatics of ABNA

The ventilation service is concerned with ventilating the air in a room. This service will be triggered, for example, if there is an unpleasant odour. To achieve the goal of this service, the following features need to be present:

Proposal ID	Action	Device	Value	Compromise
1	on	ventilation_fan	1	No
2	set	window1	100	No
3	set	window2	100	Yes
4	set	door	100	Yes

Table 4: Proposals of ventilation service

As Table 4 shows, the first two features, to switch on the ventilation fan and open window1, are essential and cannot allow to compromise, whereas the other features are preferable but can allow to compromise. In order for the first feature to be performed, a proposal to switch on the ventilation fan will be sent from the ventilation service agent (initiating agent) to the negotiator. The negotiator has a registry of all devices within the SHS and all agents who have a link to each device. Table V shows a view of the negotiator's registry.

Device ID	Device Name	Controlling Agents
101	air_conditioner	Security, Safety, AC, Ventilation, Heating, Power Management
102	ventilation_fan	Security, Safety, AC, Ventilation, Heating, Power Management
103	window1	Security, Safety, AC, Ventilation, Lighting, Entertainment
104	window2	Security, Safety, AC, Ventilation, Lighting, Entertainment
105	door	Security, Safety, AC, Ventilation, Lighting

Table 5: View of negotiator's registry

For each of the features that need to be performed, there is an associated device; in the case study presented here, these devices are the ventilation fan, window1, window2 and the door (Table 5). It is important to note that the device does not refer specifically to the window or the door, but rather to the device which opens and closes the window or the door. These devices have a value between 0 and 100, where 0 means fully closed and 100 means fully opened. Besides the ventilation service agent, there are several other agents who have a link with such devices, namely Security, Safety, air conditioning, Ventilation and Power Management, Lighting and Entertainment service agents (Table 5). The negotiator (as previously mentioned) sends each proposal to all agents who have a link to the device which is to be controlled in the proposal. However, for the purposes of this case study, only the response of the air conditioning service agent will be considered; the assumed condition here is that it is active because of the hot weather. The status of the features is presented in Table 6.

Proposal ID	Action	Device	Value	Compromise
1	on	air_conditioner	18	No
2	set	window1	0	No
3	set	window2	0	Yes
4	set	door	0	Yes
5	off	ventilation_fan	0	Yes

Table 6: Proposals of air conditioning service

Negotiation mechanism Although normally, agent priority is already established by the negotiator based on the services' weight and residents' preferences, for the purposes of the case study, both scenarios (i.e. the initiating agent has

priority and the involved agent has priority) will be considered. We will discuss two scenarios: in the first scenario, the ventilation agent has priority over the air conditioning agent. The ventilation service agent sends the first proposal to the negotiator and waits for a reply before sending the next proposal, explained in more detail below:

- The first proposal is to switch on the ventilation fan. It cannot allow a compromise, and it has no level of performance, so, the agent with the higher priority will have the advantage. The air conditioning agent will receive the proposal and check its current status and because it is 'ON' it cannot accept the proposal because one of its own features is to have the ventilation fan switched off (Table 6), but, it will inform the negotiator that there is a possibility for compromise. The negotiator will relay the compromise response 'compromise' of the air conditioning agent to the ventilation agent. As it is a feature that does not allow a compromise by the ventilation agent, it will be rejected. The negotiator will give the advantage to the ventilation agent because the feature is a necessity, and not to the air conditioning agent because a compromise on this feature can be allowed, and priority is not a consideration here. Accordingly, the order will be sent for the ventilation fan to be switched 'ON'.

- The second proposal from the ventilation service is to fully open window1, which cannot allow a compromise, and it will be received by the negotiator then delivered to the air conditioning agent. The air conditioning agent will check its current status, which is "ON" and, will inform the negotiator that it cannot accept the proposal and there can be no compromise for this feature (Table 6). However, because the air conditioning agent has other features which are less important than the proposed feature and a compromise on these can be allowed, it will send a counter-proposal offering the advantage of its least important feature. In this case, the counter-proposal will be 'set door 100', i.e., fully open the door. The negotiator will deliver the counter-proposal to the ventilation agent. Before considering the counter-proposal, the ventilation agent will first check if it has priority over the air conditioning agent. Because in this case, it has higher priority, it will reject the counter-proposal and the negotiator will send the original proposal to fully open window1 to the targeted device.

- The third and fourth proposals are similar to each other; the ventilation agent requests to fully open window2 and the door, both of which can allow a compromise by both the initiating and involved agents. The negotiator will send the proposals one by one to the air conditioning agent, who will examine its current state and send the compromise reply accordingly. Because these features can also allow a compromise by the ventilation agent, the negotiator will calculate a compromise performance value for each, and this is achieved through the following equation:
  (feature performance value in the proposal + feature performance value in the involved agent) / 2 = (100 + 0) /2 = 50

- The outcomes from the above scenario were to keep the ventilation fan and air conditioner switched on, to keep window1 fully opened and to keep window2 and the door half opened. In this negotiation, both services reached a compromise, allowing them to operate simultaneously albeit with a slight advantage for the ventilation agent, as it had priority.

Second, in scenario 2 in which the air conditioning agent has priority, the negotiation will be as follows:

- The first proposal is to switch on the ventilation fan. Although the air conditioning agent has priority, this proposal will be accepted because the particular feature involved, i.e. switching on the ventilation fan, allows a compromise by the air conditioning agent, whereas it is necessary for the initiating agent.

- The second proposal is to fully open window1, which is a feature that cannot allow a compromise for both the initiating agent (ventilation) and the involved agent air conditioning agent. Accordingly, the air conditioning agent, after checking its current status, will send a counter-proposal to set the door to 100 because it is its least important feature. The ventilation agent will receive the counter-proposal and it does not have priority and because the door being fully opened is one of its current features that it wants to perform, the counter-proposal will be accepted.

- The third proposal is to fully open window2, a feature that can allow a compromise for both agents. Therefore, the negotiator will instruct window2 to be opened halfway, similar to the proposal in the first scenario.

- The fourth proposal, which is to set the door to 100, i.e. to fully open the door, will not be sent because the included task is already done by processing the counter-proposal of the second proposal.

Because the priority in the second scenario was for the air conditioning agent, the outcome is slightly different from that of the first scenario, in which although the ventilation and air conditioning were switched on and window2 was half open, window1 was closed, and the door was fully open. In this case, both services were also able to function simultaneously, with an advantage for the air conditioning agent, as it had priority.

6.2 Correctness of ABNA

This subsection demonstrates the execution of the above scenarios using the CCA simulation tool ccaPL.

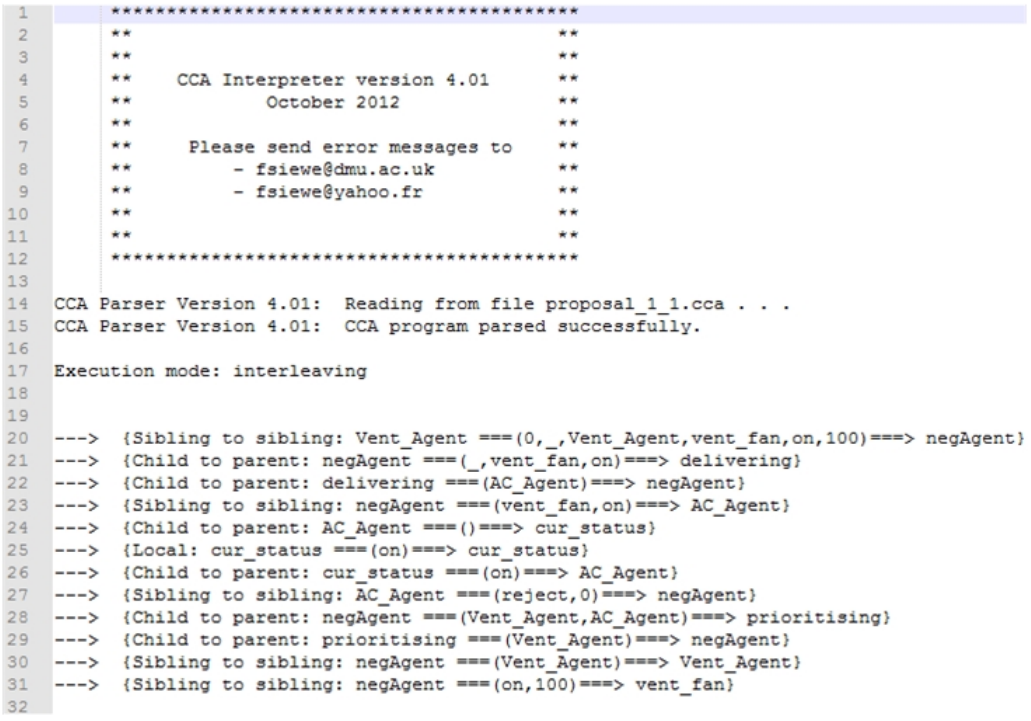

Figure 4: Negotiation algorithm

In the first scenario in which the priority is for the ventilation agent, the execution of ABNA in ccaPL is shown in Figure 4, where the line 1 up to line 19 display the ccaPL header information. In line 20, the proposal is sent from sibling to sibling, which is vent_Agent to negAgent. Line 21 shows that the negAgent ambient sends the proposal to its child ambient (delivering) in order to know the agent who should approve the proposal, which is AC_Agent, as has been shown in the next line. In line 23, negAgent sends the proposal to AC_Agent as sibling to sibling. From line 24 until line 26, there is an exchange of messages between AC_Agent and its child (cur_status), which is a memory cell to know whether the current status is 'on' or 'off'. As can be seen in Figure 4, the current status of AC_Agent is 'on'; therefore, in line 27, AC_Agent will send 'reject' to negAgent. negAgent then sends to its child ambient the proposal to prioritise the names of both the initiating and the involved agents to determine the priority for each of them. In line 29, the reply is vent_Agent so, the vent_Agent is informed that it has priority, as shown in line 30. The last line shows that negAgent sends to its sibling ven_fan the proposal to be switched on with a value of 100.

The execution of the second proposal is shown in Figure 5.

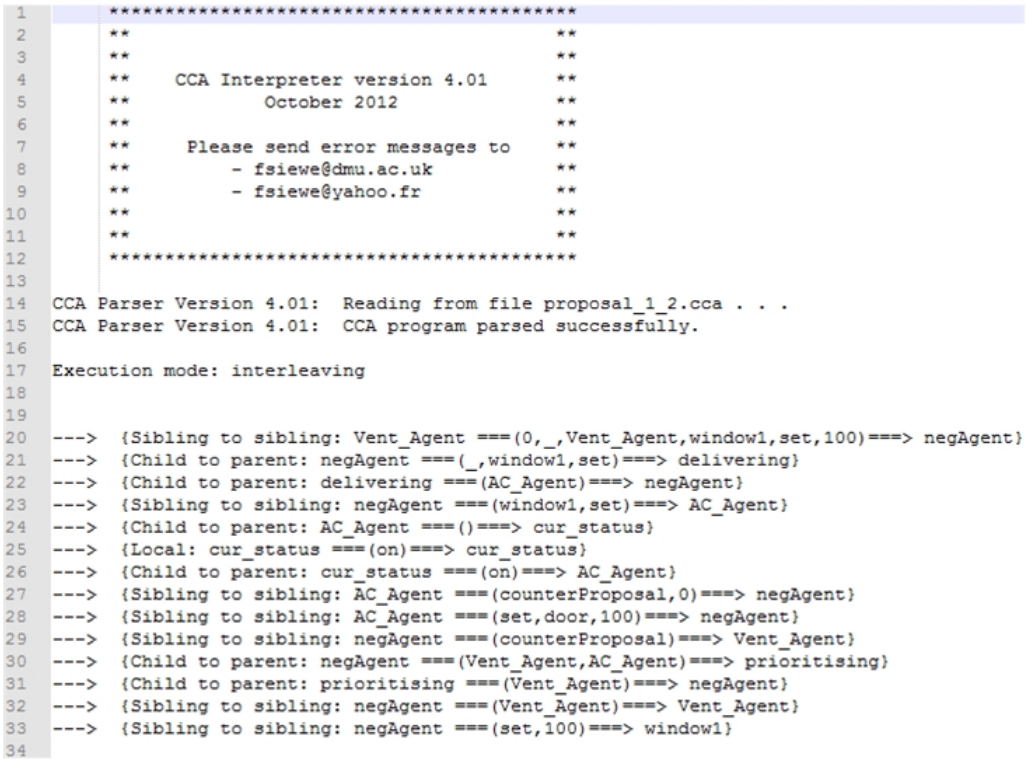

Figure 5: Execution of the second proposal in scenario 1

As can be seen in Figure 5, in line 21, the counter-proposal is sent to the initiating agent. However, the counter-proposal is rejected, and the original proposal is executed in line 33 because the initiating agent has priority.

In the third and fourth proposals, the agents reach a compromise. For example, in the fourth proposal execution in Figure 6, the agents managed to reach a compromise, so, the door will be half opened, as shown in line 30.

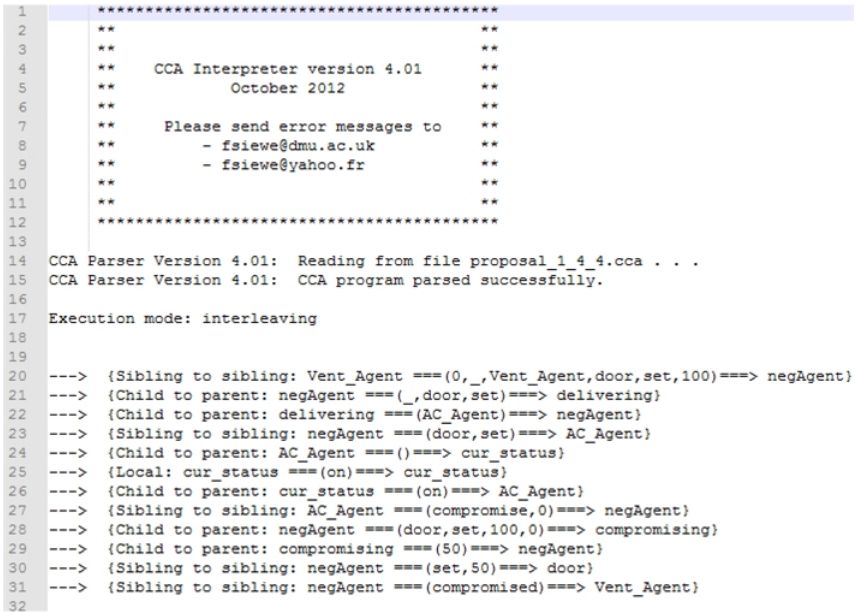

Figure 6: Execution of the third proposal in scenario 1

In the second scenario, as shown in Figure 7, in which the priority is for the air conditioning agent, the results of the negotiation processes are different from those in the first scenario, as explained in the first subsection. To show an example of the ccaPL execution, let us consider the second proposal (Figure 7). As can be seen, the proposal was to fully open window1, but because the initiating agent does not have priority, it accepts the counter-proposal to open the door instead (see lines 33 and 34). The counter-proposal will follow the same procedure as in the original procedure in which the negAgent ambient delivers the information in the counter-proposal (line 37) to its child delivering, in order to know if there is another agent who should approve the counter-proposal.

As can be seen in line 38, the reply was '(none)', which means that there is no agent who should be consulted. Therefore, the door will be ordered to be fully opened as in line 39.

```
1   ****************************************
2   **                                    **
3   **                                    **
4   **    CCA Interpreter version 4.01    **
5   **            October 2012            **
6   **                                    **
7   **     Please send error messages to  **
8   **         - fsiewe@dmu.ac.uk         **
9   **         - fsiewe@yahoo.fr          **
10  **                                    **
11  **                                    **
12  ****************************************
13
14  CCA Parser Version 4.01:  Reading from file proposal_2_2.cca . . .
15  CCA Parser Version 4.01:  CCA program parsed successfully.
16
17  Execution mode: interleaving
18
19
20  --->  {Sibling to sibling: Vent_Agent ===(0,_,Vent_Agent,window1,set,100)===> negAgent}
21  --->  {Child to parent: negAgent ===(_,window1,set)===> delivering}
22  --->  {Child to parent: delivering ===(AC_Agent)===> negAgent}
23  --->  {Sibling to sibling: negAgent ===(window1,set)===> AC_Agent}
24  --->  {Child to parent: AC_Agent ===()===> cur_status}
25  --->  {Local: cur_status ===(on)===> cur_status}
26  --->  {Child to parent: cur_status ===(on)===> AC_Agent}
27  --->  {Sibling to sibling: AC_Agent ===(counterProposal,0)===> negAgent}
28  --->  {Sibling to sibling: AC_Agent ===(set,door,100)===> negAgent}
29  --->  {Sibling to sibling: negAgent ===(counterProposal)===> Vent_Agent}
30  --->  {Child to parent: negAgent ===(Vent_Agent,AC_Agent)===> prioritising}
31  --->  {Child to parent: prioritising ===(AC_Agent)===> negAgent}
32  --->  {Sibling to sibling: negAgent ===(AC_Agent)===> Vent_Agent}
33  --->  {Sibling to sibling: negAgent ===(set,door,100)===> Vent_Agent}
34  --->  {Sibling to sibling: Vent_Agent ===(accept)===> negAgent}
35  --->  {Sibling to sibling: negAgent ===(0,AC_Agent,Vent_Agent,door,set,100)===> counterProp}
36  --->  {Sibling to sibling: counterProp ===(1,AC_Agent,Vent_Agent,door,set,100)===> negAgent}
37  --->  {Child to parent: negAgent ===(AC_Agent,door,set)===> delivering}
38  --->  {Child to parent: delivering ===(none)===> negAgent}
39  --->  {Sibling to sibling: negAgent ===(set,100)===> door}
40
```

Figure 7: Execution of the second proposal in scenario 2

Acknowledgment

This project was funded by the Deanship of Scientific Research (DSR) at King Abdulaziz University, Jeddah, under grant no. J-723-611-1438. The authors, therefore, acknowledge with gratitude DSR for technical and financial support.

7 CONCLUSION

SHSs are complex; they include several services with a multitude of features that respond to the preferences of multiple residents. FIs are an inevitable consequence which can prevent services from functioning properly and may result in an unpredictable behaviour of services provided by SHSs. In this paper, ABNA was presented. This approach is significant because it allows negotiation, enabling services with conflicting features to work simultaneously. What is more, it has the ability to deal with both types of FIs, i.e. policy- and service-based FIs.

ABNA applies agent-based negotiation, to enable the negotiation mecha-

nism; a hierarchy of agent features is presented to illustrate their priority in relation to the performance of each agent. To clarify ABNA, we have presented the formalisation for the approach in *CCA*.

ABNA is represented as a process so that it can be executed and analysed using the execution environment of *CCA*. We have illustrated how such an analysis can be done using a case study within SHS. Different scenarios have been analysed to test the various properties of ABNA. The ABNA makes a novel contribution to the detection and resolution of FIs in SHSs by seeking a compromise between conflicting agents before considering a priority. However, it is based on certainty in that its functionality depends on pre-defined criteria. Hence, future work should focus on making ABNA support uncertainty. This could be achieved by allowing the negotiator to be smart, employing a self-learning mechanism. This technique will enable the negotiator to monitor the behaviour of agents and record their reactions in different scenarios, so future decisions can be taken more quickly in similar situations.

References

[1] A. Nhlabatsi, R. Laney and B. Nuseibeh, "Feature interaction: the security threat from within software systems," Progress in Informatics, vol. 5, pp. 75-89, 2008.

[2] E. Pulvermueller, A. Speck, J. O. Coplien, M. D'Hondt and W. De Meuter, "Feature interaction in composed systems," in Object-Oriented Technology, A. Frohner, Ed. Springer, pp. 86-97,2002.

[3] M. Weiss and B. Esfandiari, "On feature interactions among web services," in Web Services, 2004. Proceedings. IEEE International Conference On, pp. 88-95, 2004.

[4] P. Zave and M. Jackson, "New feature interactions in mobile and multimedia telecommunications services." in FIW, pp. 51-66, 2000.

[5] E. C. Lupu and M. Sloman, "Conflicts in policy-based distributed systems management," Software Engineering, IEEE Transactions On, vol. 25, pp. 852-869, 1999.

[6] A. Almutairi and F. Siewe, "Formal specification of CA-UCON model using CCA," in Science and Information Conference (SAI), 2013, pp. 369-375, 2013.

[7] M. Kolberg, E. H. Magill and M. Wilson, "Compatibility issues between services supporting networked appliances," Communications Magazine, IEEE, vol. 41, pp. 136-147, 2003.

[8] A. Classen, P. Heymans and P. Schobbens, "What's in a feature: A requirements engineering perspective," in Fundamental Approaches to Software Engineering, F. José, Springer, pp. 16-30,2008.

[9] A. Alfakeeh and A. Al-Bayatti, "Feature Interactions Detection and Resolution in Smart Homes Systems," International Journal of Electronics and Electrical Engineering (IJEEE), vol. 4, 2016.

[10] F. Siewe, H. Zedan and A. Cau, "The calculus of context-aware ambients," Journal of Computer and System Sciences, vol. 77, pp. 597-620, 2011.

[11] L. Cardelli and A. D. Gordon, "Mobile ambients," Theor. Comput. Sci., vol. 240, pp. 177-213, 2000.

[12] M. Calder, E. Magill and D. Marples, "Hybrid approach to software interworking problems: managing interactions between legacy and evolving telecommunications software," Software, IEE Proceedings -, vol. 146, pp. 167-175, 1999.

[13] G. Misra, A., S. Misra and K. Agrawal, "Device to Device Millimeter Wave Communication in 5G Wireless Cellular Networks (A Next Generation Promising Wireless Cellular Technology)," International conference on Signal Processing, Communication, Power and Embedded System (SCOPES), pp. 89-93, 2016.

[14] M. Nakamura, H. Igaki and K. Matsumoto, "Feature interactions in integrated services of networked home appliance," in Proc. of Int'l. Conf. on Feature Interactions in Telecommunication Networks and Distributed Systems (ICFI'05, pp. 236-251, 2005.

[15] M. Shehata, A. Eberlein and A. O. Fapojuwo, "Managing policy interactions in KNX-based smart homes," in Computer Software and Applications Conference, 2007. COMPSAC 2007. 31st Annual International, pp. 367-378, 2007.

[16] H. Hu, D. Yang, L. Fu, H. Xiang, C. Fu, J. Sang, C. Ye and R. Li, "Semantic Web-based policy interaction detection method with rules in smart home for detecting interactions among user policies," IET Communications, vol. 5, pp. 2451-2460, 2011.

[17] M. Shehata, A. Eberlein and A. Fapojuwo, "Using semi-formal methods for detecting interactions among smart homes policies," Science of Computer Programming, vol. 67, pp. 125-161, 2007.

[18] C. Hsu and L. Wang, "A smart home resource management system for multiple inhabitants by agent conceding negotiation," in Systems, Man and Cybernetics, 2008. SMC 2008. IEEE International Conference On, pp. 607-612, 2008.

[19] P. Carreira, S. Resendes and A. C. Santos, "Towards automatic conflict detection in home and building automation systems," Pervasive and Mobile Computing, vol. 12, pp. 37-57, 6, 2014.

[20] H. Igaki and M. Nakamura, "Modeling and detecting feature interactions among integrated services of home network systems," IEICE Trans. Inf. Syst., vol. 93, pp. 822-833, 2010.

[21] N. D. Griffeth and H. Velthuijsen, "The negotiating agents approach to runtime feature interaction resolution," in Feature Interactions in Telecommunications Systems, pp. 217-235, 1994.

[22] M. Wilson, E. H. Magill and M. Kolberg, "Detection and prediction of insider threats to cyber security: a systematic literature review and meta-analysis," in Consumer Communications and Networking Conference, 2005. CCNC. 2005 Second IEEE, pp. 251-256, 2005.

[23] A. I. Gheyas, and E. A. Abdallah"Towards automatic conflict detection in home and building automation systems," Big Data Analytics, Springer, vol.1, pp.1-6, 2016.

[24] B. J. Muscedere, R. Hackman, D. Anbarnam and J. M. Atlee, I. J. Davis and M. W. Godfrey, "Detecting Feature-Interaction Symptoms in Automotive Software using Lightweight Analysis, " 2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER), pp. 175-185, 2019.

[25] S. Takeuchi, M. Takemoto and M. Matsuo," SPIRE Scalable and Unified Platform for Real World IoT Services with Feature Interaction ", IEEE 40th Annual Computer Software and Applications Conference (COMPSAC), pp.348-353, 2016.

[26] D. Mekuria, P. Sernani, N. Falcionelli and A. Dragoni," Reasoning in Multi-agent Based Smart Homes: A Systematic Literature Review ", Ambient Assisted Living - Italian Forum 2018, Ninth Italian Forum on Active and Assisted Living, (ForItAAL) 2018, Lecce, Italy, pp. 161-179, 2018.

[27] J. Walzberg, T. Dandres, R. Samson, N. Merveille and M. Cheriet," An agent-based model to evaluate smart homes sustainability potential ",28th IEEE Annual International Symposium on Personal, Indoor, and Mobile Radio Communications, (PIMRC) 2017, Montreal, QC, Canada, October 8-13, 2017, pp. 1-7, 2017.