Mobile Netw Appl
DOI 10.1007/s11036-017-0899-5

Modeling and Reasoning about Preference-Based Context-Aware Agents over Heterogeneous Knowledge Sources

Ijaz Uddin[1] · Abdur Rakib[2] · Hafiz Mahfooz Ul Haque[3] · Phan Cong Vinh[4]

Abstract This paper presents a conceptual framework and multi-agent model for context-aware decision support in dynamic smart environments based on heterogeneous knowledge sources. A Protégé plug-in for rules extraction from distributed ontologies has been developed, which allows us to model context-aware agents using the notion of multi-context systems. Extracted rules can be annotated to match the users' needs and to develop a preference model to support their preferences so as to provide a user with a more personalized services. The use of the proposed framework is illustrated using a simple fact-based preference model developed from ontologies considering two different smart environment domains.

Keywords Context-aware agents · Multi-context system · Rule-based distributed reasoning · Ontology · User preference

A preliminary version of this work under the title "Modelling and Reasoning About Context-Aware Agents over Heterogeneous Knowledge Sources" has appeared in the proceedings of the 5th International Conference on Context-Aware Systems and Applications (ICCASA'16) November 24–25, 2016, Vietnam.

✉ Abdur Rakib
Rakib.Abdur@uwe.ac.uk

Ijaz Uddin
khyx4iui@nottingham.edu.my

Hafiz Mahfooz Ul Haque
mahfoozul.haque@cs.uol.edu.pk

Phan Cong Vinh
pcvinh@ntt.edu.vn

1 School of Computer Science, The University of Nottingham Malaysia Campus, Semenyih, Malaysia

2 Department of Computer Science and Creative Technologies, The University of the West of England, Bristol, UK

3 Department of Computer Science and Information Technology, The University of Lahore, Lahore, Pakistan

4 Nguyen Tat Thanh University, Ho Chi Minh City, Vietnam

1 Introduction

There is no doubt that with an increasing number of smart devices such as smartphones in use, the vast amounts of contextual data being generated has great influence on context-aware mobile computing research. Smartphones have a variety of embedded sensors that can be used to automate data collection and provide a platform to infer rich contextual data about users, including location, time, and environmental condition, among others. This is known as customized information according to the specific context. To be more precise, these sensors can be used to gather the contextual information of a user or to manipulate the context. Different notions of context have been studied across various fields of computer science and various physical and conceptual environmental aspects can be included in the notion of context [1]. Among others, Dey et al. [2] define a context-aware system as a system which uses context to provide relevant information and/or services to its user based on the user's tasks. The formal context modeling and reasoning about context is one of the fundamental research areas in context-aware computing. In the literature, various context modeling and reasoning approaches have been proposed, including ontology and rule-based approach [3–5]. In our previous work [4, 5], we have developed formal logical frameworks and shown how context-aware systems can

Published online: 05 September 2017

be modeled as multi-agent reasoning agents. A formal logical model allows us to capture a system's behavior in a systematic and precise way. This is because a formal logic has simple unambiguous syntax and semantics, which also allows automated reasoning. Our approach to context modeling was based on a domain specific centralized ontology, which allows a formal representation of domain knowledge and advancing contextual knowledge sharing among the agents. However, in a real context-aware deployment setting, a coalition of heterogeneous domains often require to mutually share/exchange context knowledge. This needs different modeling approach to deal with distributed contextual knowledge considering more than one domain. In this connection, the notion of multi-context systems has been used for interlinking different knowledge sources in order to enhance the expressive capabilities of heterogeneous systems. A multi-context system (MCS) includes a set of contexts and a set of inference rules that allows information to flow among different contexts [7]. In MCS, each context is defined as a self-contained knowledge source which includes the set of axioms and inference rules to model the system and perform local reasoning. In the literature, many definitions of multi-context systems have been proposed (see e.g., [8, 9]). In [8], Brewka et al. define multi-context system as a number of people, agents, databases etc. to describe the available information from a set of contexts and inference rules and specify the information flow among these contexts. In [9], Benslimane et al. have described ontology as a context, which is itself an independent self-contained knowledge source having a set of axioms and inference rules with its own reasoner to perform reasoning. In this work, we consider the concept of context in two levels. The first level is based on multi-context system to model heterogeneous systems similar to contextual ontologies studied by [9]. For the second level, we follow the approach proposed in our previous work [4, 5], where a context is formally defined as a *(subject, predicate, object)* triple that states a fact about the subject where — the subject is an entity in the environment, the object is a value or another entity, and the predicate is a relationship between the subject and object.

The main contributions of this paper are: first, we extend our previous work [5] by introducing a different modeling approach to deal with distributed context handling considering more than one domain. This approach is novel in a sense that context-aware agents use contextual information which are extracted from different knowledge sources. Second, an implementation of a Protégé plug-in for rules extraction from distributed ontologies. This allows us developing context-aware systems that share different ontologies as heterogeneous knowledge sources. Third, we propose a fact-based preference model, which can be applied in order to give user a personalized service and to reduce load of an agent's inference engine by selecting a sub-set of available rules based on the user preference.

The rest of the paper is structured as follows. In Section 2, we briefly review distributed description logic, and present related work focusing on context-aware systems incorporating multiple ontologies, and preferences in context-aware systems. In Section 3, we present a Protégé plug-in that allows a user to select OWL 2 RL ontology files augmented with SWRL rules and translates them into a set of plain text Horn clause rules. In Section 4, we present a conceptual framework for modeling context-aware reasoning agents using the notion of MCS incorporating user preferences. In Section 5, we present a simple case study developed from ontologies considering two different smart environment domains, focusing our discussion on fact-based preference. Finally, we conclude this paper in Section 6.

2 Background study and related work

In the background study our main focus remains on the interoperability of different ontologies, context-aware systems that incorporate multiple ontologies, and how preferences can be integrated into it. It requires some human intervention especially when talking about the preferences. The sections below introduce some related work and give us more insight into extracting rules from heterogeneous knowledge sources and personalizations in context-aware systems.

2.1 Distributed description logics

Recent developments in the field of semantic web have led to a renewed interest in the distributed knowledge bases [11–13]. A growing body of research realizes the significance of extending the OWL based formalism by providing inter-ontology mappings through distributed description logics. Distributed description logic (DDL) is a formal logical framework which combines different description logics (DLs) knowledge bases to express heterogeneous information. A DDL is basically a generalization of the DL framework, which is designed to formalize multiple ontologies interconnected by semantic mappings [11]. One of the reasons for interconnecting ontologies is to preserve their own identity and specify their independence [12]. DDLs have introduced the notion of multiple ontologies with distributed reasoning where each local ontology has its own local knowledge base. Each local ontology knowledge base consists of TBox and ABox axioms. The correspondence of different ontology axioms is called inter-ontology axioms or bridge rules. Bridge rules map the TBox axioms of one ontology with the TBox axioms of other ontology in an implicit manner. In other words, distributed TBox expresses the semantic relations among local TBoxes via bridge rules. These bridge rules allow concepts of an ontology to subsume a concept from another ontology, and they express the semantic mappings among different ontologies. A bridge rule is an inter-ontology axiom having one of the following forms: $C_i \overset{\sqsubseteq}{\rightarrow} D_j$; $C_i \overset{\sqsupseteq}{\rightarrow} D_j$; where C_i, D_j are concepts of ontologies O_i and O_j respectively. A distributed DL knowledge base (DKB) is a set of different DL knowledge bases, expressed as a pair $\langle \mathfrak{T}, \mathfrak{A} \rangle$, which consists of distributed TBoxes and ABoxes. Let us assume that we have a collection of DLs and each DL is represented by $\{\mathcal{DL}_i\}$, where $i \in I$ is an element of a non empty set of indexes used to identify ontologies.

A distributed TBox (DTBox) defines TBoxes $\{T_i\}_{i \in I}$ of all local DLs from their corresponding domain ontologies, and bridge rules between these TBoxes which are of the form $\mathfrak{B} = \{\mathfrak{b}_{ij}\}$ (which states a set of bridge rules $\mathfrak{B}$ from $\mathcal{DL}_i$ to $\mathcal{DL}_j$ and $\{\forall i, j (i \neq j) \in I\}$). So, DTBox is represented as $\mathfrak{T} = \langle \{T_i\}_{i \in I}, \mathfrak{B} \rangle$.

A distributed ABox (DABox) $\mathfrak{A} = \langle \{A_i\}_{i \in I}, \mathfrak{C} \rangle$ consists of ABoxes $\{A_i\}_{i \in I}$ of all local DLs from their corresponding domain ontologies, and a set of individuals that may either be partial or complete are of the form $\mathfrak{C} = \{\mathfrak{c}_{ij}\}$ which means the individuals corresponds from $\mathcal{DL}_i$ to $\mathcal{DL}_j$ and $\{\forall i, j (i \neq j) \in I\}$.

2.2 Multi-context systems

There has been a renewed research interest in making multiple heterogeneous ontologies interoperate. For example, the work by [11] has introduced a system which can carry out reasoning services with multiple ontologies. The authors have discussed the reasoning problem in multiple ontologies interrelated with semantic mappings, where the results of local reasonings performed in single ontologies are combined via semantic mappings to reason over distributed ontologies. In [14], a framework is presented for multi-context reasoning systems, which allows combining arbitrary monotonic and nonmonotonic logics and nonmonotonic bridge rules are used to specify the information flow among contexts. In [15], authors have proposed a distributed algorithm for query evaluation in a Multi-Context Systems framework based on defeasible logic. In their work, contexts are built using defeasible rules, and the proposed algorithm can determine for a given literal P whether P is (not) a logical conclusion of the Multi-Context Systems, or whether it cannot be proved that P is a logical conclusion.

2.3 Context-aware systems incorporating multiple ontologies

In the literature, there has been considerable work on context-aware systems that incorporate multiple ontologies. In [16], Weiβenberg et al. have developed FLAME2008, a prototype system for intelligent personalized Web services for the Olympics 2008 in Beijing. This is basically an integration platform for service customization by using the information based on individual situations and personal demands of users. In [17], Sheshagiri et al. have developed a prototype semantic web environment known as MyCampus, which allows users acquiring different sets of task-specific agents that help them with different tasks. These agents require knowledge of one or more contextual attributes of users and the sources of contextual information are modeled as semantic web services that can be automatically discovered and accessed by agents. In [18], Bouzeghoub et al. have presented a context aware semantic recommend system. The proposed system considers situation aware adaptive recommendation of information to assist mobile users in a campus environment. The recommendations are based on a multidimensional ontology that models people, buildings, events and available resources. In [19], Garcia-Sola et al. have proposed a context-aware systems development architecture using distributed semantic web reasoning. The framework is based on modular ontologies and the reasoning process considers SWRL rules.

These reviewed systems consider complementary technological mechanisms and modeling approaches, such as the notion of context-awareness, incorporating multiple ontologies, and mobile technology. However, our proposed approach to context-aware system modeling and reasoning is quite different in a sense that heterogeneous knowledge sources are translated into a set of Horn-clause rules. The translated rules are then used to model context-aware non-monotonic rule-based agents.

2.4 Preferences in context-aware systems

In context-aware computing, user preferences play an important role in adapting the behaviour of systems to satisfy the individual user's need. There have been numerous attempts to incorporate preferences in context-aware applications, particularly in manipulating the context, storing, management and its use in future (see, e.g., [20–23]). Furthermore, preference or personalization has seen its way in database queries, where a query result also depends on the current contexts available [24]. However, the databases are used for quite large storage and the authors have used the OLAP paradigm for storage of user preferences. The model also keeps track of the previous results and store them in a context tree, indicates the abundance of memory availability. Personalization in context aware or intelligent homes/spaces has been in research for quite some time. For example, IBM blue space [25], in which office spaces are personalized based on the user preferences. A user has to carry his active badge, which is detected by a sensor installed at the office space. Once the user is identified, his preferences are applied to the sitting place that may include the temperature, ambiance, air flow etc. This system only provides preferences to the user at some particular location and does not have any inference engine. It is more like a profile saving system which is applied automatically whenever required. Furthermore, it can notify the other users via a panel about his current status as busy or available, and in case of busy status when will he change his status to available. IBM had implemented this smart space in 2002 with both positive and negative responses, as for the employees it was a good idea and everyone was interested to install the smart space at their places. However, the companies were more into the productivity rate instead of user privacy etc. As they argued that all employees will mostly use the do not disturb signs when using the smart space and stay permanently. Another such example is Intelligent Home [26], the primary concern of this project is to find the resource coordination among autonomous home controlled agents, installed in distributed way. The protocol used for the communication or allocation of resources is SHARP. The results are based on simulations. The authors have concluded to use better protocols for better results, but its not apparent if the system is based on rules or otherwise. In project AURA [27], the author intended to lower the distraction of the users' attention and to create an environment that can adapt to users' contexts and needs. As an example, it provides a scenario of a user walking from his office towards a presentation room and all his works are carried with him from a desktop to the handheld PC, and from handheld PC to the projection screen. It integrates a lot of variables such as face detection in audience and give feedback to the user about the audience. It also provides a user mobility and avoiding the resources variations. It can shift a user work into new environment while trying to keep the users resources available at best. Some other works based on the preferences a user may provide include [28]. It uses techniques that observe the environment and learn the users' anticipation. The author argues that, the techniques can be better where a user do not want cumbersome programmable agents or dashboards to provide user preferences. Furthermore, it is believed to be operating transparent to the user. In [29], authors have presented a working mechanism that is mainly based on the prediction algorithms. It uses various techniques and meta-predictor to anticipate the future for the user. These methods try to keep the user control minimum. Some recent works on preference, including [30] use the profiling of a user. Different profiles are maintained to store the context for each user. Profiles define the services and context for every context available. Furthermore, the preferences are provided in terms of context and services. The matching mechanism adopted in this approach uses redundant variable substitution unless all the variables and combination of variables are achieved. Rules are modified by a adjust profile module. It checks all the instances with the working memory facts. Based on the context, the profile is loaded. This approach uses a lot of memory space, as it substitutes the variables and checks instances of rules in the working memory. Another approach [31] also uses profiling, the framework is able to capture and process context dependent preferences. It uses different level of certainty and based on the certainty it provides results to a tourist or a user. It can alter the query of a user by removing the least certain query in order to get more results. It needs preference registration and profile creation for a user, and the user has to interact actively to update the profile and for reasoning purposes. Another approach is proposed in [32], which focuses on implicitly building and maintaining preference set by monitoring and learning mechanism for a user. It uses IF-THEN-ELSE preference interface for user control. It requires extensive preference set to be created for a user. Continuous monitoring of the user is required to record the actions a user performs. Every action is attached with the context snapshot (attribute, op, value), e.g., (location, =, home) if a user is doing some action while at home. Short term memory (STM) stores an action which is transformed to long term memory (LTM) for further training. In [33], a multi-layered model is proposed which actually based on the context history and predicts the preferences of the user. This model has four main layers such as data gathering, context management, preference management, and application layer. The users have profiles for saving their context histories and related information.

Above literature demonstrate significant progress on preference based context-aware research, however, none of the proposed system considers resource constraints while modeling and/or developing the systems in mobile computing environment. This paper considers a preference model to personalization of resource-bounded context-aware applications, and provides services based on user's preference of mobile device combined with environmental contexts in use. In the following sections, we first introduce Protégé plug-in development to extract Horn-clause rules from multiple ontologies. The extracted rules will be used to design our rule-based non-monotonic context-aware agents.

3 D-Onto-HCR Protégé plug-in

To extract the rules from different ontologies, we have developed a Protégé plug-in. It allows a user to select OWL 2 RL ontology files augmented with SWRL rules and translates them into a set of plain text Horn clause rules. Once translated, the user can further edit the rules e.g., to inset priorities, flags or preferences. Once the editing has been completed, the translated Horn clause rules could be used to

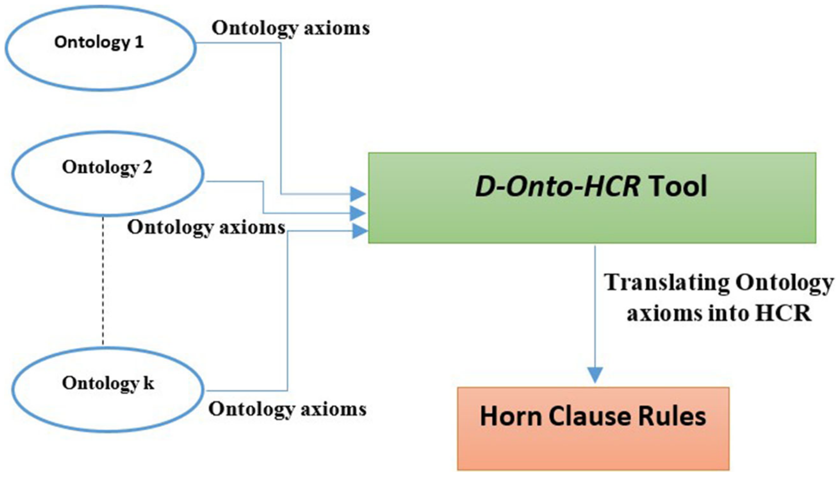


Fig. 1 Distributed semantic knowledge translation process

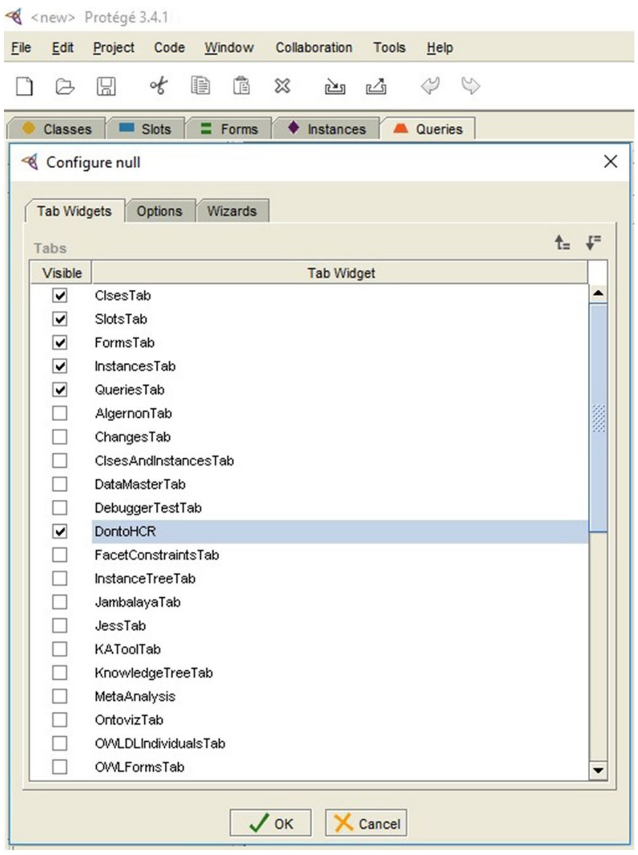
model the desired context-aware system. Over all, the plug-in is a few step process. First, it allows the user to select the downloaded ontologies which are in OWL/XML format. Next, the OWL parser parses the ontologies into OWL API objects, which gathers the ABox and TBox axioms. After that, it translates the TBox axioms into Horn-clause rules as the rest are already in the Horn-clause rule format. Finally, the user selects the generated file for editing, once the edits are done the resulting file can be used to create any sematic web rule based application (Fig. 1). Figures 2, 3 and 4 show the various steps involved in translating the ontologies. This step can be repeated for different ontologies which will be translated into a single file containing the plain text Horn-clause rules extracted from multiple ontologies. The following subsections describe the relevant details of various aspects of the plug-in.

3.1 Development environment

The development of Protégé plug-in is carried out in the Eclipse IDE for Protégé version 3.4.1. In order to develop a plug-in, there are quite a few steps involved. First, we setup the project and named it after the plug-in as Donto-HCR. Once a project has been setup, we need to add all the external libraries that are required by the plug-in, by adding external JAR's within the IDE to our project. Since we have developed a tabbed plug-in, we had to create a Java plug-in class. To create a tab widget, we extended the AbstractTab-Widget class from the protege.jar to implement the initialize() method. This method runs when the plug-in starts on the Protégé main interface. This gives us a place where we can put our code that can appear in the Protégé as a tabbed plug-in. However, in order to run it on Protégé environment, we first set the manifest file which makes the Protégé to recognize the new plug-in. For testing, we had to change the run configuration and set the working directory besides some other changes to the Protégé main directory. This way when we try to run the program it launches the Protégé interface from where we can run the plug-in and check its functionalities. So far, we have generally discussed the plug-in, the technical components of the plug-in are described below.

3.2 Working mechanism of the plug-in

The plug-in is an OWL-API based translator, to extract rules from different ontologies. It is a high level Java based



Visible	Tab Widget
☑	ClsesTab
☑	SlotsTab
☑	FormsTab
☑	InstancesTab
☑	QueriesTab
☐	AlgernonTab
☐	ChangesTab
☐	ClsesAndInstancesTab
☐	DataMasterTab
☐	DebuggerTestTab
☑	DontoHCR
☐	FacetConstraintsTab
☐	InstanceTreeTab
☐	JambalayaTab
☐	JessTab
☐	KAToolTab
☐	KnowledgeTreeTab
☐	MetaAnalysis
☐	OntovizTab
☐	OWLDLIndividualsTab
☐	OWLFormsTab

Fig. 2 Enabling the plug-in in Protege

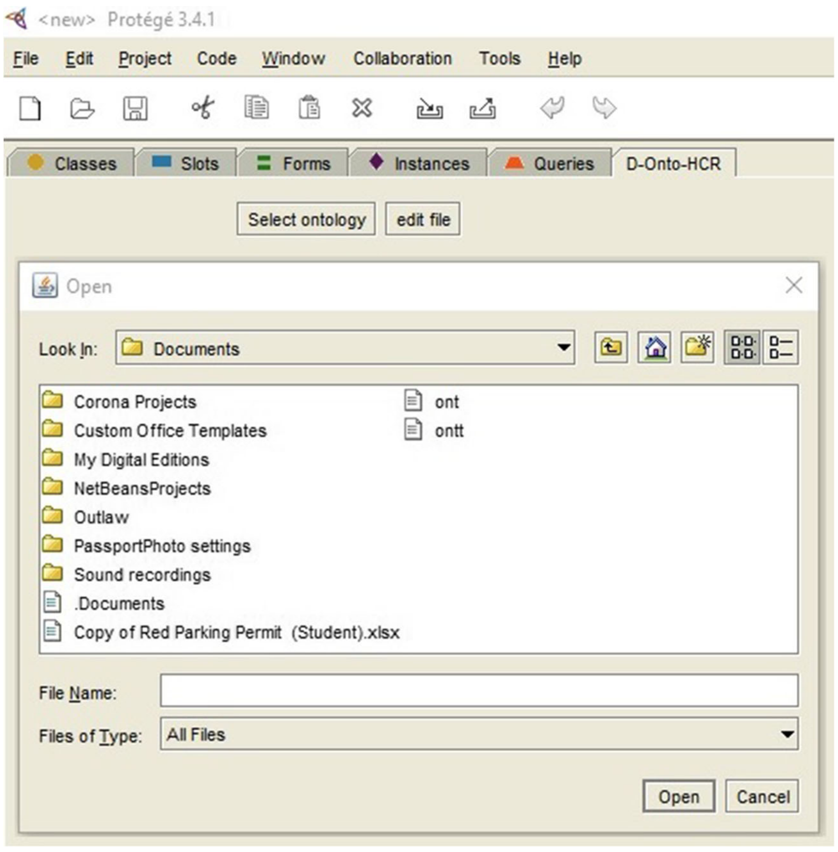
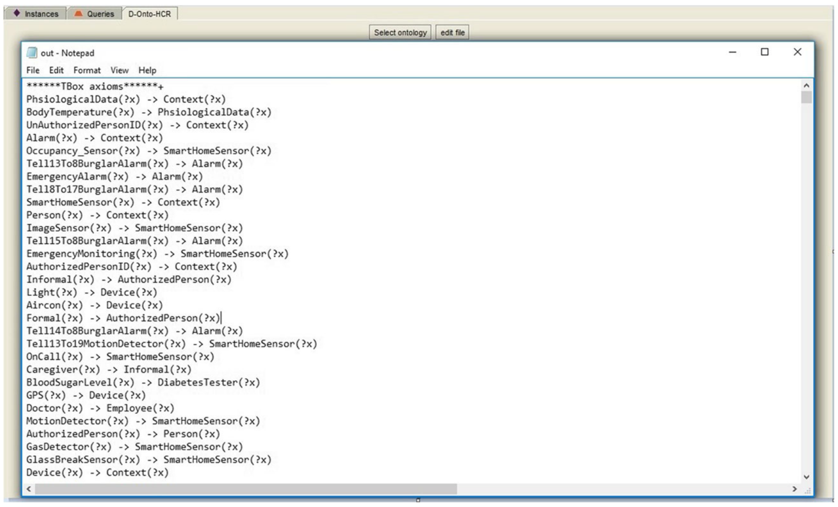


Fig. 3 Selecting ontology for translation

API that supports the creation of OWL ontologies and also enables us to manipulate the ontologies. The OWL API enables third party developers, to create and/or customize different implementations for their components. It is helpful in loading, saving, parsing and serializing ontologies in different syntaxes such as, OWL/XML, RDF/XML, functional syntax, Manchester syntax, KRSS, Turtle syntax, etc. Furthermore, it has set of interfaces for probing, manipulating and reasoning with OWL ontologies. Some of the main features of OWL API are axiom-centric abstraction, reasoner interfaces, validations for different OWL 2 profiles and first class change support. In terms of functionalities the plug-in input is the ontology. Which translates the set of axioms into Horn-clause rule. The set of of axioms can be OWL 2RL and SWRL form. The plugin translates DL-safe rules axioms into Horn-clause rules. In addition to that, it


Instances Queries D-Onto-HCR
Select ontology edit file
out - Notepad
File Edit Format View Help

```
******TBox axioms******+
PhsiologicalData(?x) -> Context(?x)
BodyTemperature(?x) -> PhsiologicalData(?x)
UnAuthorizedPersonID(?x) -> Context(?x)
Alarm(?x) -> Context(?x)
Occupancy_Sensor(?x) -> SmartHomeSensor(?x)
Tell13To8BurglarAlarm(?x) -> Alarm(?x)
EmergencyAlarm(?x) -> Alarm(?x)
Tell8To17BurglarAlarm(?x) -> Alarm(?x)
SmartHomeSensor(?x) -> Context(?x)
Person(?x) -> Context(?x)
ImageSensor(?x) -> SmartHomeSensor(?x)
Tell15To8BurglarAlarm(?x) -> Alarm(?x)
EmergencyMonitoring(?x) -> SmartHomeSensor(?x)
AuthorizedPersonID(?x) -> Context(?x)
Informal(?x) -> AuthorizedPerson(?x)
Light(?x) -> Device(?x)
Aircon(?x) -> Device(?x)
Formal(?x) -> AuthorizedPerson(?x)
Tell14To8BurglarAlarm(?x) -> Alarm(?x)
Tell13To19MotionDetector(?x) -> SmartHomeSensor(?x)
OnCall(?x) -> SmartHomeSensor(?x)
Caregiver(?x) -> Informal(?x)
BloodSugarLevel(?x) -> DiabetesTester(?x)
GPS(?x) -> Device(?x)
Doctor(?x) -> Employee(?x)
MotionDetector(?x) -> SmartHomeSensor(?x)
AuthorizedPerson(?x) -> Person(?x)
GasDetector(?x) -> SmartHomeSensor(?x)
GlassBreakSensor(?x) -> SmartHomeSensor(?x)
Device(?x) -> Context(?x)
```

Fig. 4 Editing the translated file

extracts concepts from multiple ontologies and maps them correspondingly in the form of bridge rules. These bridge rules are first converted into OWL 2 RL rule format and then into Hron-Clause rules. Figure 1 shows the process of translation. When the plug-in is loaded, an ontology file is provided as an input. OWL parser is used then to parse the ontology into OWL API objects which then extracts the set of TBox and ABox axioms. The resultant set of TBox and ABox axioms is then translated into Horn-clause rules. The bridge rules are extracted from different ontologies and translated into Horn-clause rules format. The process is repeated for any number of ontologies and the final output is a single file which contains a set of plain text Horn-clause rules.

4 Multi-agent model over heterogeneous knowledge sources

We extend the logical framework presented in [5] by incorporating the notion of multi-context systems where rules are derived from heterogeneous semantic knowledge sources. The system consists of $n_{Ag}(\geq 1)$ individual agents $Ag = \{1, 2, ..., n_{Ag}\}$. Each agent $i \in A_g$ has a program, consisting of a finite set of strict, defeasible, and bridge rules, and a working memory, which contains facts. Each agent in the system is represented by a triple $(\Re, \mathcal{F}, \succ)$, where $\mathcal{F}$ is a finite set of facts contained in the working memory, $\Re = (\Re^s, \Re^d, \Re^{br})$ is a finite set of strict, defeasible, and bridge rules, and $\succ$ is a superiority relation on $\Re$. Strict rules $(\Re^s)$ are non-contradictory whereas defeasible rules $(\Re^d)$ can be defeated based on contrary evidence. Bridge rules $(\Re^{br})$ are non-contradictory rules which represent the distributed knowledge base concepts. In this framework, each context-aware agent is designed to solve a specific problem. Agents in the system acquire contextual information from domain specific ontologies (rules and facts of an agent can be derived from one or multiple ontologies), perform reasoning (based on the information they have in their knowledge bases), communicate with each other, and adapt the system behaviour accordingly.

4.1 Preference based multi-agent model

In [10], we have extended the logical framework presented in [5] to accommodate rules that are derived from heterogeneous semantic knowledge sources. In this paper, we further extend our previous work [10] to practically implement the plug-in and incorporate preferences of users and providing the personalized services. The structure of inference engine and setup remain the same with a minor change to the input rules only. A typical rule format of our framework can be found in [5], while some changes are made when preferences are intended. The typical structure of the rules after adding the preference is as follows:

$$m : P_1, P_2, \ldots, P_n \rightarrow P_0 \ : F : CS \ \text{ where } n \geq 0.$$

where m is the rule priority. Each P_i is an atomic formula of the form $p(t_1, t_2)$, $Ask(i, j, \ p(t_1, t_2))$ or $Tell(i, j, p(t_1, t_2))$, where i and j $(i \neq j)$ represent agents, p is a predicate symbol and the t_k are terms. Where Ask and $Tell$ are special atoms used for communication between the agents [5]. The flag F, a placeholder, associated with every rule is used to specify the type of the rule. For instance, the character 'G' is used to represent a rule containing a Goal statement, which indicates that a certain rule execution results in goal achievement. The character 'C' represents the communication rules, which can trigger a communication between agents (devices). The character 'D' represents the deduction rules. The indicator CS can be a literal(s), a fact(s), or –, indicating which set the rule belongs to, and is mainly used for the preference set generation. It is explained in more detail in the following. To incorporate the preference, an extra preference manager layer has been added to the actual framework presented in [6], while keeping the rest of the core system intact. The general idea of the preference is to create a subset of rules based on certain preferences defined by the user, doing so will make the process execute faster as the inference engine only needs to go through selected rules instead of the whole rule base. Different modules under the preference manager layer work together to achieve the overall design objective. The preference manager layer is composed of Preference Set Generator (PSG), Context Monitor (CM), Context Set (CS), and Context of Interest (COI). While COI is provided in advance to the system before a subset of the rules is selected. Figure 5 shows the relationship between these components.

Context set (CS) is an addition to the rule base. It carries literals, - or facts to represent a relationship of a rule with any context or user preference. For example, *Person(?p), OfficeRoom(?o), hasLocation(?p, ?o)* $\rightarrow$ *inOffice(?p, ?o)* having indicator "L" in CS suggests the relationship of the rule is with *location*, all the rules having the literal "L" means they represent a group of rules belonging to *location* as preference. More than one literal as indicators are also allowed in which case a single rule can belongs to various preference sets of rules. If a rule does not carry any literals or facts, it means that it is a general rule represented with "-" in the context set, and will be added to every subset that is created for the preference set. **Context monitor (CM)** component holds the Context of Interests (COI) of a user, i.e., it holds the values provided by the user. Context monitor after reading the values passes them to the Preference Set Generator(PSG). The PSG defines a sub set of rules based on the user preferences, called a preference set. This subset is then passed to the inference engine for processing. Context

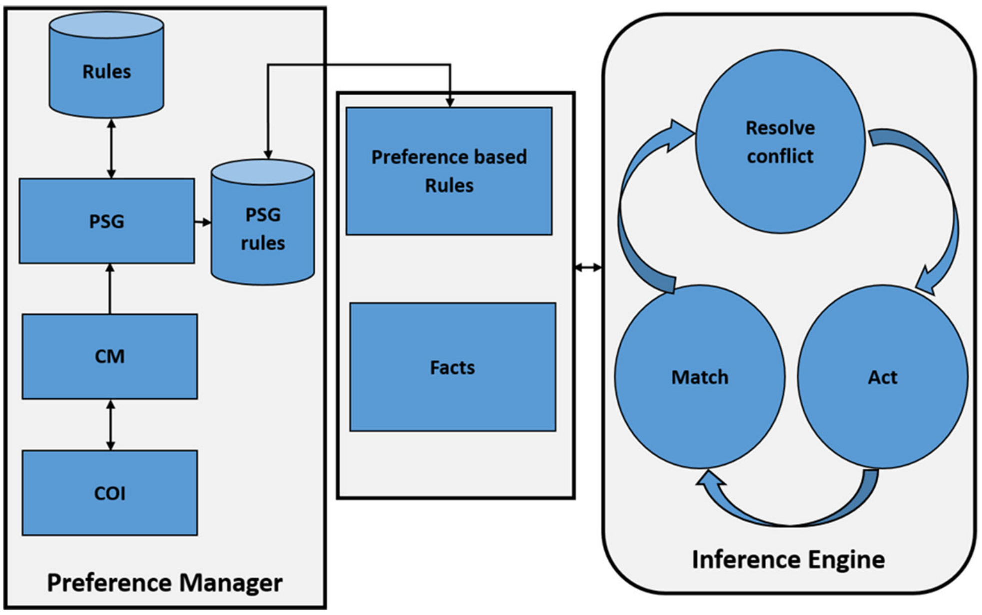


Fig. 5 Preference generation overview

monitor actively monitors the contexts of interests. Any change in it's contents results in a new preference set. **Preference set generator (PSG)** gives the framework an ability to provide personalized services. PSG receives instructions from the CM to generate a subset of personalized rules. In order to generate a subset of the rules, the PSG considers the contexts that are of interest to the user. The values carried by the CM give the PSG instructions to which rules to be selected.

4.2 Working mechanism of preference

The above discussed components work together to provide personalized services to the user. The rule-base of an agent contains all the rules provided by the system designer. When the reasoning process starts, the CM component retrieves the values provided by the user in terms of COI, which are forwarded to the PSG. PSG then communicate with the rule-base and selects a subset of the available rules, which will be provided to the inference engine. The PSG in reality looks for the CS indicator in each rule and select those rules required by the user in terms of preferences provided. The whole system still works as described in [6], however the rules in memory are actually replaced with the preference set.

This is fine as far as we can group up the rules based on a wide notion of preference, e.g., all rules that belong to the *location* preference. However, we can further narrow it down to the extent that we can actually split the context into its fact based value, e.g., selecting rules that are related to a particular location The University of Nottingham Malaysia Campus (UNMC) only, we call such a preference as the fact-based preference. Usually, a general indicator will only select rules having something to do with the location. However, a location can be home, office, hospital, university or any other places. Thus, general preference will still select unnecessary rules. To make the preference set smaller, we need to use the facts. For instance, rules related to the location UNMC will only pick those rules which are required when the user location is at UNMC. Thus discarding all the other location based rules, saving memory and providing a light weight rule set to an agent's inference engine. We discuss fact-based preference in more detail in the following section.

4.3 Fact based user preference

As discussed above, apart from the general preference, a user might also need to prefer some specific location rather than all the available locations. As a simple example consider a person traveling from location A to location C, and he needs to check at hospital in location B. He can then enable preference for location B only. This can be achieved by using the facts received from the GPS device. Since the system designer is already aware of the set of rules and the expected outcomes of facts, the indicators used for the fact based preferences are instances of concepts defined in the domain ontology. For example, *Location(UNMC)* can be used as an indicator if it is also defined in the ontology. It is not necessary that the same predicate must be used in the rule to be considered for preferences. Once the preference is found, it can reload a new set of rules to match the user preference. For this reason, we introduce the Preference Level Monitor (PLM).

4.4 Preference level monitor (PML)

Preference levels give user a choice of what type of preferences are desired and upto which level the preferences are desired. The PLM can accommodate both the simple preference along with the fact-based preferences. In fact-based preference, we need to validate the provided facts before they can be considered for preference set generation. In simple preference, a CS determines which rules should be chosen, it does not require any validation. For instance, when we enable a preference corresponding to location, it simply put all the rules that are associated with the context set of location. But when we assume that a location is *UNMC* or when the location is detected as *UNMC*, it needs to check if the user's current location is indeed *UNMC*. To achieve this, we need to capture the facts before those are added to an agent's working memory. In Fig. 6, it can be observed that, between the CM and PSG we added another PLM layer. This layer validates the facts with the COI, and upon validation it forwards them to the PSG. This layer first checks the type of the preference whether it is a simple preference or fact-based preference, as also shown in Algorithm 2. The algorithm runs on startup and then if any change is detected within the PLM, it re-initiate the algorithm. The available contents of the PLM produce a distinct code at state S, if at S' the code is changed i.e., $S_c \neq S'_c$. This ensures that the contents to be visited only when a change is detected. In this algorithm, we define the distinct code so that it can help in re-initiating the algorithm while generating a preference set. Its working mechanism is simple, it generates a code which can be a Hash code based on the elements within PLM. As an example, we consider that given COI as {GPS(Home)} and external facts as {GPS(UNMC)}, a MD5 hash code could be produced as 08ef88c567c2de7eb44166841cb26c59. Now in the next state if we assume that the {GPS(UNMC)} has been replaced with the {GPS(Home)} then a new Hash code 43e3bbf9eab28646ad2d35a77716a856 will be generated. This is clearly a different code, and hence $S_{cd} \neq S'_{cd}$. When this change of code is detected it indicates a change within the contents of external facts or in COI, and ultimately re-initiating the algorithm for further processing. Note that a simple flag can not be used here. As a flag can only indicate any new COI or received external facts, however, it can not identify if the newly received facts or COI are already members of PLM or not. Thus a flag will indicate a change detection every time a new external fact is received. For example, if we have external facts as {GPS(Home)} and we again receive the same fact from the GPS, the simple flag will indicate a change while the distinct code will not. This is because the code will remain the same for the same set of external facts and COI. These algorithms have been developed to incorporate the preferences and the whole system works as described in [6].

Algorithm 1 Checking PLM states

```
  Input: $S_{cd}$: Distinct code at state S, $S'_{cd}$ Distinct code
         at state $S'$
  Output: Invoke PLM Algorithm 2
1 START
2 do
3 |  Invoke Algorithm 2
4 while $S_{cd} \neq S'_{cd}$;
5 END
```

Algorithm 2 PLM working

```
   Input: COI: Current Context of Interest, R: Rules, $F_e$:
          Facts from external agents or sensors, CS:
          Context Set, Regex: regular expression
   Output: Preference Set based on COI
1  START
2  if Regex(COI)==[a-zA-Z] then
3  |  Fetching Simple preference
   |  for r→[R] do
4  |  |  if $\exists x \in$ COI such that $x \in$ CS[r] then
5  |  |  |  Add r to Preference Set
6  |  |  end
7  |  end
8  end
9  else if Regex(COI)==[a-zA-Z]+([a-zA-Z0-9]+) OR
   [a-zA-Z]+([a-zA-Z0-9]+,[a-zA-Z0-9]) then
10 |  Fact-based preference of the form A(b) or B(b,c)
   |  for r→[R] do
11 |  |  if $\exists x \in$ COI such that $x \in$ CS[r] AND $x \in F_e$
   |  |  then
12 |  |  |  Add r to Preference Set
13 |  |  end
14 |  end
15 end
16 else if CS[r]== "-" then
17 |  Add r to general rule
18 end
19 END
```

In this work, since a context-aware system is modeled as a multi-agent system, the agents in the system interact with each other while achieving a goal. That is, agents exchange contexts and perform some common tasks. This behavior makes it easier for a designer or knowledge expert to design rules which are expected to fire at certain state. Similarly, when designing the system for a particular domain, we can always know in advance the expected output while certain rules are fired. For example, given a set of rules and appropriate facts related to blood pressure measurement, the

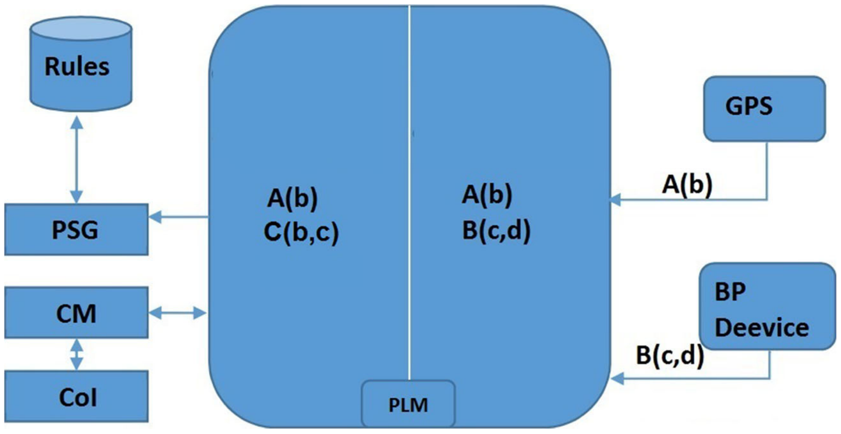


Fig. 6 Facts based preference validation

outcome after firing rules can either be HIGH, NORMAL, or LOW blood pressure. Thus, these values can be used in preference modeling. For example, only rules that generate HIGH blood pressure could be of user's interest to notify appropriate personnel so that action can be taken in case of an emergency situation.

4.5 Preference selection

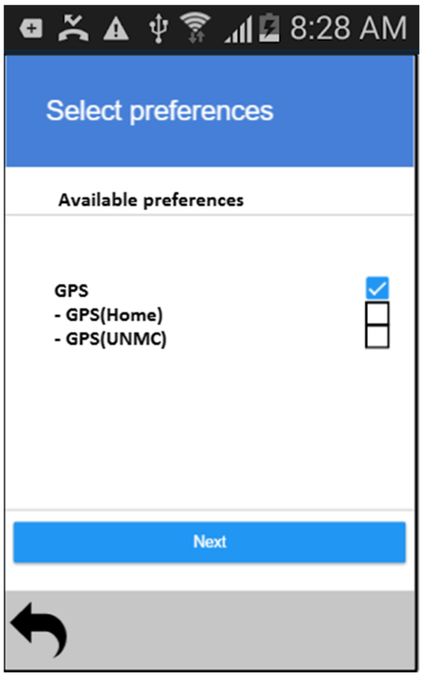
As discussed earlier, the knowledge expert can define the normal as well as fact-based preferences whenever the requirements are met. This, however, gives a user less control. Alternatively, the preferences selected for the system can be further filtered by mapping the CS to a distinct check boxes, which gives a user to enable/disable certain preferences and override the designer preference model, as shown in Fig. 7. In order to effectively use the fact-based preference, currently the working model supports preferences with external facts received from sensors and other agents. For example, facts received from the GPS, BP and Heart monitoring devices can be used to enforce the preferences of a patient care device. Agents also derive facts during execution which are stored in the working memory, however, in this discussion we do not consider those facts to be used in the preferences. Instead, the modeling is more focused on effective usage of the facts received from external sensors/agents.

5 Case study: heterogeneous smart environment modeling

We model a smart environment facilitator system considering two domain ontologies, namely Smart Office and Patient Care. The purpose is to model context-aware reasoning agents in healthcare environments which require sharing of knowledge across the domains, including data generated by embedded sensors and wearable smart badges in that environments, while dealing with semantic heterogeneity that exists across the knowledge sources. Fragments of the ontologies are depicted in Figs. 8 and 9 and the corresponding translated Horn-clause rules using the Protégé plug-in are shown in Fig. 4. These OWL 2 RL ontologies are augmented with a number of SWRL rules. Additionally, we construct the bridge rules which are semantically mapped using distributed DL knowledge bases. Some of the bridge rules are given below:

O_{PCO}:Doctor $\underset{\rightarrow}{\sqsubseteq}$ O_{SOO}:Employee

O_{PCO}:Nurse $\underset{\rightarrow}{\sqsubseteq}$ O_{SOO}:Employee


8:28 AM

Select preferences

Available preferences

GPS ☑
- GPS(Home) ☐
- GPS(UNMC) ☐

Fig. 7 Preferences selection interface

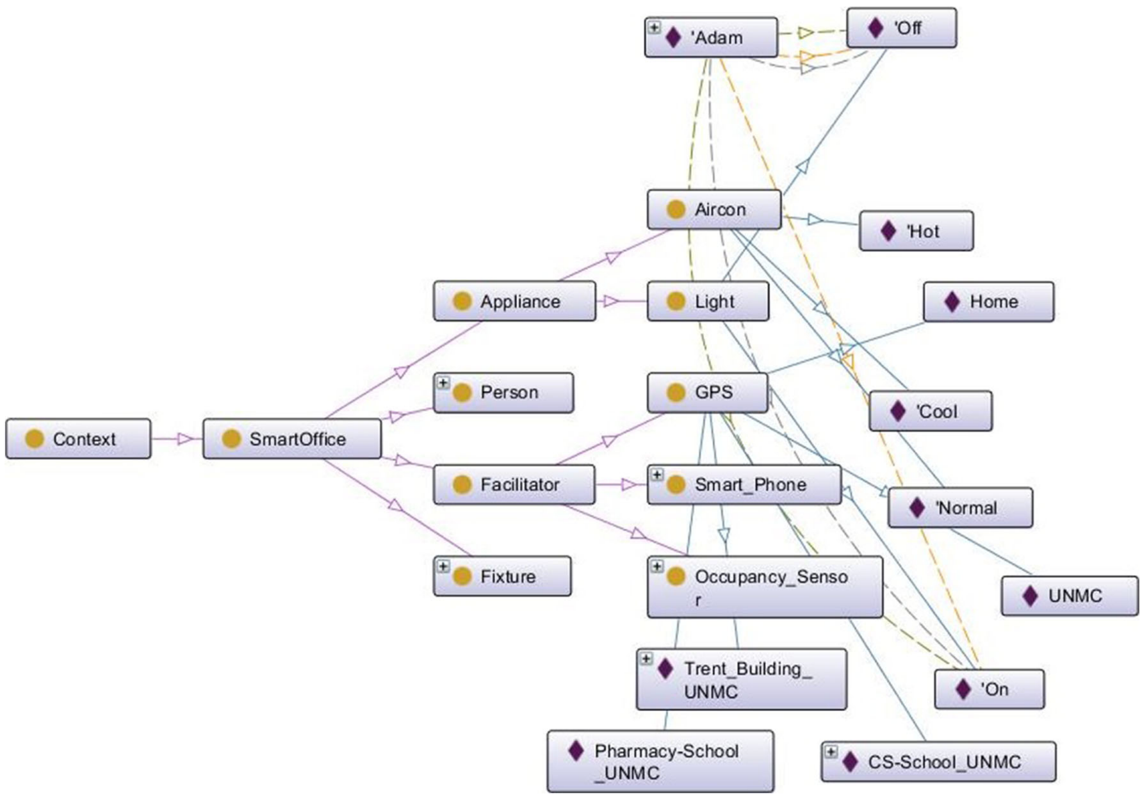


Fig. 8 Smart office ontology

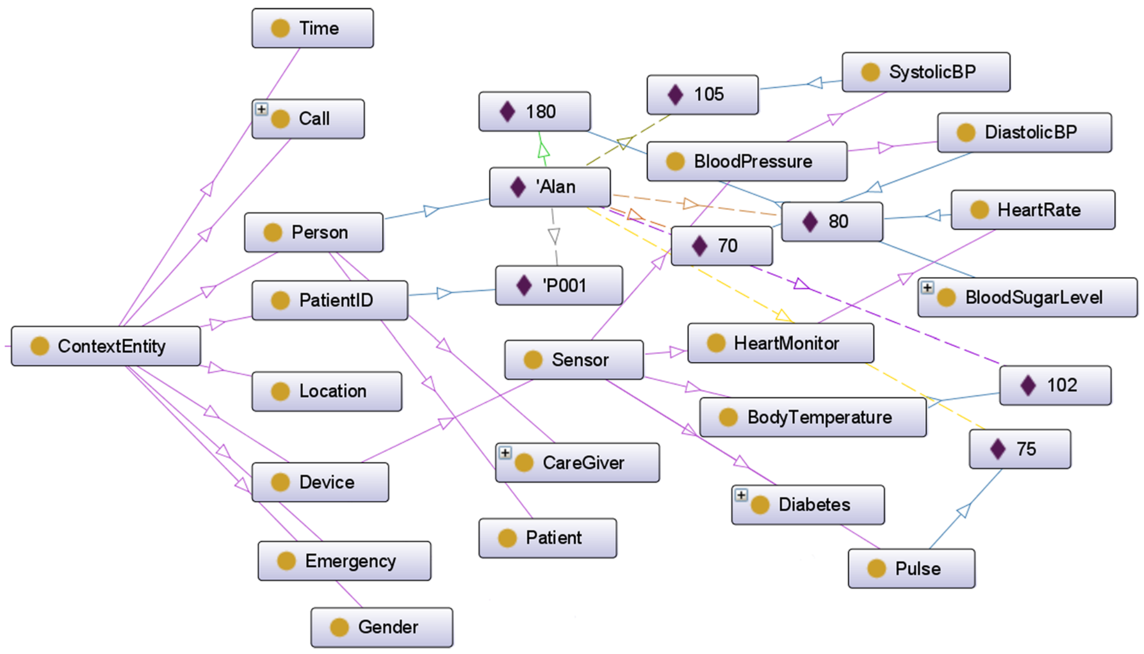


Fig. 9 Patient care ontology

Table 1 Example rules for agent 1 with identifiers

Id	Rule	Identifier
R1	Tell(2,1,hasBloodPressure(?p, Normal))⟶ hasBloodPressure(?p, Normal)	GPS(Home)
R2	Tell(2,1,hasBloodPressure(?p, High))⟶ hasBloodPressure(?p, High)	GPS(Home)
R3	Patient(?p), hasBloodPressure(?p, Normal)⟶ ∼hasSituation(?p, Emergency)	GPS(Home)
R4	Patient(?p), hasBloodPressure(?p, High)⟶ hasSituation(?p, Emergency)	GPS(Home)
R5	Person(?p), GPS(?loc) ⟶ hasLocation(?p, ?loc)	–
R6	Tell(3,1,hasOccupancy(?p, Yes)) ⟶ hasOccupancy(?p,Yes)	GPS(UNMC)
R7	hasOccupancy(?p, Yes) ⟶ Tell(1, 4, hasAircon(?p, On))	GPS(UNMC)

O_{SOO}:Employee $\underset{\rightarrow}{\sqsubseteq}$ O_{PCO}:Patient

O_{SOO}:GPS $\underset{\rightarrow}{\sqsubseteq}$ O_{PCO}:Device

In [10], we have discussed how we model context-aware agents as multi-agent reasoning systems over heterogeneous knowledge sources. Here we limit our discussion more on the preference side, and how the preference sets are generated, while the execution of the rules remains the same as described in [10]. In our recent work [34], we have presented initial experiment on a real world scenario considering the basic preferences and not the fact-based preference. In this system modeling, we focus our discussion on fact-based preference. Let us consider the set of seven rules in Table 1 that models a Patient/Person agent (1). For the sake of simplicity, rule priorities have been omitted and the Id column simply representing rule numbers to identify rules. This agent interacts with other agents as well as receives information from various embedded sensors. For example, it receives blood pressure information from the the BloodPressure measurement agent (2), it receives occupancy status from a Occupancy sensor (3), and it also interacts with the Aircon agent (4) to make it On or Off as appropriate. For illustration purposes, we assume that Aircon is available only at the office located in UNMC. It can be observed that rules R6 and R7 are not useful when the person is at home. Therefore, we can annotate the rules to provide preferences based on the user choices. In this scenario we assume that a user wants to have different set of preferences at home and at UNMC. He may want to switch on the Aircon whenever he is available in his office, and he wants to monitor his blood pressure when at home and in case of an emergency situation he wants to be notified. Thus, the preference for the user at home is to check his blood pressure and not to include the rules that deal with the UNMC environment. This will give user a personalized service and to reduce the overall rules of the agent 1's inference engine by considering only relevant rules applicable to a given situation.

In Table 2, we explain how the preference sets are generated for a given COI and received external facts. We assume that the user moves from the location Home towards UNMC. The COI column defines the preferences provided by the user, which we assume to be constant throughout the execution and can only be changed on demand. The Externally Received Facts column represents the high level contexts received form the GPS sensor. The Preference Set column represents selected subset of rules as a result of given the COI and the externally received fact(s). Whenever an externally received fact(s) matches with the COI, the corresponding preference set is generated. In this table, we assume a whole cycle when a user moves from his Home to UNMC and back to Home. Where, T2, T3, and T5 represent locations when a user is neither at his Home nor at UNMC.

Table 2 Preference set generation

	COI	Externally Received Facts	An element of COI found in Externally Received Facts	Preference Set
T1	{GPS(Home),GPS(UNMC)}	{GPS(Home)}	Yes	R1,R2,R3,R4,R5
T2	{GPS(Home),GPS(UNMC)}	{GPS(Semenyih)}	No	R5
T3	{GPS(Home),GPS(UNMC)}	{GPS(TTS)}	No	R5
T4	{GPS(Home),GPS(UNMC)}	{GPS(UNMC)}	Yes	R5,R6,R7
T5	{GPS(Home),GPS(UNMC)}	{GPS(BTK)}	No	R5
T6	{GPS(Home),GPS(UNMC)}	{GPS(Home)}	Yes	R1,R2,R3,R4,R5

6 Conclusion and future work

In this paper, we presented a conceptual framework and multi-agent model for context-aware systems based on heterogeneous knowledge sources. We have developed a Protégé plug-in for rules extraction from distributed ontologies, which allows us to model personalized resource-bounded context-aware applications. The proposed modeling approach emphasizes on the knowledge expert/designer's role to get preference from the user to be integrated into the system's rule base. This actually gives an end user a bit less control. To overcome this issue, upto certain level, a user can be provided with preference check boxes in which the CS contents from the rule base can be mapped to check boxes. Using such check boxes will enable user to select/deselect different preferences. However, a user still depends on the system designer to include or remove more preferences from the rule base. In future work, we would like to implement an interface to overcome this issue and let the user add/delete preferences from the CS column directly. We would also like to further narrow down the preference level so that it can be applied to the derived contexts of an agent's working memory. In that case, it will not only be applied before hand but any context that a user expects to be derived can also be opted for preferences.

References

1. Lieberman H, Selker T (2000) Out of context: computer systems that adapt to, and learn from, context. IBM Syst J 39(3-4):617–632
2. Dey AK (2001) Understanding and using context. Pers Ubiquit Comput 5(1):4–7
3. Esposito A, Tarricone L, Zappatore M, Catarinucci L, Colella R (2010) A framework for context-aware home-health monitoring. IJAACS 3(1):75–91
4. Rakib A, Haque HMU, Faruqui R (2014) A temporal description logic for resource-bounded rule-based context-aware agents. In: Context-aware systems and applications. Lecture notes of the institute for computer sciences, social informatics and telecommunications engineering, volume 128. Springer, pp 3–14
5. Rakib A, Haque HMU (2014) A logic for context-aware non-monotonic reasoning agents. In: Proceedings of the 13th Mexican international conference on artificial intelligence, MICAI 2014. Lecture notes in computer science, vol 8856. Springer, pp 453–471
6. Uddin I, Rakib A, Haque HMU (2017) A framework for implementing formally verified resource-bounded smart space systems. Mobile Networks and Applications 22(2):289–304
7. Eiter T, Fink M, Schüller P, Weinzierl A (2014) Finding explanations of inconsistency in multi-context systems. Artif Intell 216:233–274
8. Brewka G, Roelofsen F, Serafini L (2007) Contextual default reasoning. In: Proceedings of the 20th international joint conference on artifical intelligence, pp 268–273
9. Benslimane D, Arara A, Falquet G, Zakaria M, Thiran P, Gargouri F (2006). In: Contextual ontologies. Lecture notes in computer science, vol 4243. Springer
10. Haque HMU, Rakib A, Uddin I (2016) Modelling and reasoning about context-aware agents over heterogeneous knowledge sources. In: International conference on context-aware systems and applications. Springer, pp 1–11
11. Serafini L, Tamilin A (2005) Drago: distributed reasoning architecture for the semantic web. In: Proceedings of the second european semantic web conference. Lecture notes in computer science, vol 3532 . Springer, pp 361–376
12. Grau BC, Parsia B, Sirin E (2004) Working with multiple ontologies on the semantic web. In: Proceedings of the third international semantic web conference. Lecture notes in computer science, vol 3298. Springer, pp 620–634
13. Borgida A, Serafini L (2002) Distributed description logics: directed domain correspondences in federated information sources. In: Proceedings of the confederated international conferences on the move to meaningful internet systems. Lecture notes in computer science, vol 2519 . Springer, pp 36–53
14. Brewka G, Eiter T (2007) Equilibria in heterogeneous nonmonotonic multi-context systems. In: Proceedings of the twenty-second AAAI conference on artificial intelligence. AAAI Press, pp 385–390
15. Bikakis A, Antoniou G, Hasapis P (2011) Strategies for contextual reasoning with conflicts in ambient intelligence. Knowl Inf Syst 27(1):45–84
16. Weißenberg N, Voisard A, Gartmann R (2004) Using ontologies in personalized mobile applications. In: Proceedings of the 12th annual ACM international workshop on geographic information systems, GIS, pp 2–11
17. Sheshagiri M, Sadeh N, Gandon F (2004) Using semantic web services for Context-Aware applications. In: Proceedings of mobisys2004 workshop on context awareness
18. Bouzeghoub A, Ngoc K, Krug L (2009) Situation-aware adaptive recommendation to assist mobile users in a campus environment. In: Proceedings IEEE international conference on advanced information networking and applications, pp 503–509
19. Garcia-Sola A, Garcia-Valverde T, Botia JA, Munoz A (2014) Reasoning with modular ontologies for context-aware applications. J Res Pract Inf Technol 46(4):235–261
20. Stefanidis K, Pitoura E, Vassiliadis P (2006) Modeling and storing context-aware preferences. Springer, Berlin, pp 124–140
21. Coutand O (2009) A framework for contextual personalised applications. Kassel University Press GmbH
22. Hong J, Suh EH, Kim J, Kim S (2009) Context-aware system for proactive personalized service based on context history. Expert Syst Appl 36(4):7448–7457
23. Barkhuus L, Dey A (2003) Is context-aware computing taking control away from the user? Three levels of interactivity examined. Springer, Berlin, pp 149–156
24. Stefanidis K, Pitoura E, Vassiliadis P (2006) Modeling and storing context-aware preferences. In: Advances in databases and information systems: 10th east European conference, ADBIS 2006, Thessaloniki, Greece, September 3–7, 2006, Proceedings, Berlin, Heidelberg. Springer, Berlin, pp 124–140
25. Lai J, Levas A, Chou P, Pinhanez C, Viveros M (2002) Bluespace: personalizing workspace through awareness and adaptability. Int J Hum Comput Stud 57(5):415–428

26. Lesser V, Atighetchi M, Benyo B, Horling B, Raja A, Vincent R, Wagner T, Xuan P, Zhang SX (1999) The umass intelligent home project. In: Proceedings Of the third annual conference on autonomous agents. AGENTS '99. ACM, New York, pp 291–298
27. Garlan D, Siewiorek DP, Smailagic A, Steenkiste P (2002) Project aura: toward distraction-free pervasive computing. IEEE Pervasive Comput 1(2):22–31
28. Mozer M (2005) The adaptive house. The IEE Seminar on Intelligent Building Environments
29. Cook DJ, Youngblood GM, Heierman EO III, Gopalratnam K, Rao S, Litvin A, Khawaja F (2003) Mavhome: an agent-based smart home. In: PerCom, vol 3, pp 521–524
30. Hoque MR, Kabir MH, Seo H, Yang SH (2016) Pare: profile-applied reasoning engine for context-aware system. Int J Distrib Sens Netw 12(7)
31. Confalonieri R, Inan H, Palau M (2012) Handling uncertain user preferences in a context-aware system. In: International conference on information processing and management of uncertainty in knowledge-based systems. Springer, pp 88–97
32. McBurney S, Papadopoulou E, Taylor N, Williams H (2008) Adapting pervasive environments through machine learning and dynamic personalization. In: 2008 IEEE international symposium on parallel and distributed processing with applications, IEEE, pp 395–402
33. Hong JY, Suh EH, Kim SJ (2009) Context-aware systems: a literature review and classification. Expert Syst Appl 36(4):8509–8522
34. Uddin I, Rakib A (2017) A preference-based application framework for resource-bounded context-aware agents. In: Proceedings of the 4th international conference on mobile and wireless technology (ICMWT'17). (to appear)