Mobile Networks and Applications (2019) 24:82–99
https://doi.org/10.1007/s11036-018-1140-x

An Efficient Rule-Based Distributed Reasoning Framework for Resource-bounded Systems

Abdur Rakib[1] · **Ijaz Uddin**[2]

Published online: 23 October 2018

Abstract
Over the last few years, context-aware computing has received a growing amount of attention among the researchers in the IoT and ubiquitous computing community. In principle, context-aware computing transforms a physical environment into a smart space by sensing the surrounding environment and interpreting the situation of the user. This process involves three major steps: context acquisition, context modelling, and context-aware reasoning. Among other approaches, ontology-based context modelling and rule-based context reasoning are widely used techniques to enable semantic interoperability and interpreting user situations. However, implementing rich context-aware applications that perform reasoning on resource-bounded mobile devices is quite challenging. In this paper, we present a context-aware systems development framework for smart spaces, which includes a lightweight efficient rule engine and a wide range of user preferences to reduce the number of rules while inferring personalized contexts. This shows rules can be reduced in order to optimize the inference engine execution speed, and ultimately to reduce total execution time and execution cost.

Keywords Rule-based reasoning · Expert systems · Preferences · Context-aware systems

1 Introduction

In recent years, context-aware computing, an important subfield of mobile and ubiquitous computing technology, has been the focus of much attention from the computer science, artificial intelligence, and many other research communities. This emerging technology can be described as the next generation of information and communication technology which has large scale impact on our daily lives. Its application encompasses many safety critical domains including health care [1]. In these systems, information can be collected by using tiny resource-bounded devices, including, PDAs, smartphones, and wireless sensor nodes. With the emergence of the smartphone industry, cloud computing and easy connectivity among devices, a significant change can be seen from ubiquitous computing towards the Internet of Things or IoT [2]. The basic idea is that the devices can perform computing anywhere, at any time and are also connected to each other and the Internet, especially handheld devices such as mobile phones. Mobile devices, and more specifically smartphones, are becoming one of the first feasible platforms for ubiquitous computing [3]. It is a fact that smartphones are slowly replacing desktop systems [4] and are becoming a must have device for the general user especially because of their versatility. A smartphone is a cellphone with advanced features that supports a wide range of functionality, including, but not limited to, web browsing, email, voice and instant messaging over the internet, capturing, storing and transmitting audio, videos and photos, social networking, precise location, and many more other activities. In general, a modern smartphone is equipped with a number of sensors that can collect a lot of data from location to the device orientation to environment conditions. These sensors could produce a huge amount of data, both in structured and unstructured forms. Thus, with an intelligent smart device in hand, capable of processing different kinds of data and with a variety of sensors attached to it along with the capacity to connect to external devices/sensors, it can be used more effectively as a context-aware device in building context-aware application systems. A context-aware system

✉ Abdur Rakib
Rakib.Abdur@uwe.ac.uk

Ijaz Uddin
khyx4iui@nottingham.edu.my

[1] Department of Computer Science and Creative Technologies, The University of the West of England, Bristol, UK

[2] School of Computer Science, The University of Nottingham Malaysia Campus, Semenyih, Malaysia

is a system which uses context to provide relevant information and/or services to its user based on the user's tasks. In the literature, various definitions of context exist (see e.g., [5, 6]). Dey et al. [6] define context as any information that can be used to identify the status of an entity. An entity can be a person, a place, a physical or a computing object. This context is relevant to a user and an application, and reflects the relationship between them. According to this widely accepted definition, we can consider the person as an entity while the data generated by the smartphone or sensors about the person is its context. Based on the context, if any device or the system is taking any action then it is one of the many examples of a context-aware system. Such a system can be designed as an expert system to make it context-aware and intelligent enough to realize its environment and act accordingly or to take decisions based on its own knowledge. Rule-based reasoning is one of the most popular approaches which is often used for designing a system as an expert system [7]. However, most of the existing rule engines used to develop expert systems rely on resource hungry algorithms and high-end technology, while the usage of such systems on small scale devices is no where to be seen. In this paper, we propose a lightweight efficient rule engine and a wide range of user preferences to reduce the number of rules in order to optimize the inference engine execution speed. We design a context-aware system as a rule-based multi-agent system that run on Android devices, where we use an ontology-based context model and a rule-based reasoning technique to represent contexts and infer the context changes.

The rest of the paper is structured as follows. In Section 2, we review background concepts. In Section 3, we present related work. The first part focusses on mobile-based context-aware frameworks, and the second part presents the well known existing inference engine algorithms of rule-based systems and their complexity analysis. In Section 4, we present a motivational analysis of the proposed research approach. In Section 5, we present our proposed framework, a lightweight efficient rule engine algorithm and its complexity analysis. In Section 6, we discuss the preferences that provide a novel approach to reduce the overall load from the inference engine. In Section 7, we present a case study implemented from ontologies considering several smart space agents, and conclude the paper in Section 8.

2 Background literature

2.1 Structure of rule-based systems

In the field of artificial intelligence (AI), rules are often used for building knowledge-based expert systems [7]. Usually, any system that works on the basis of rules is called a rule-based system (RBS) [8]. Rule-based systems are an important class of AI reasoning systems, and such systems are rapidly becoming an important component of mainstream computing technologies, for example in business process modelling, the semantic web, sensor networks etc. Rules can be traced back to early production systems as a well-known and popular way of encoding expert knowledge, and they play a significant role in the field of AI for modelling human reasoning and problem-solving processes in a specific domain. Human reasoning can be closely defined in terms of IF-THEN statements. Therefore, RBS becomes an obvious choice when it comes to encoding a human expert's knowledge [9]. Each rule can carry a minute amount of knowledge, and backed with the facts from the environment, it acts similar to a human brain. The rules act as long-term memory while the facts are considered to work as a short-term memory [10, 11]. The RBS technology is used widely in various types of software of different domains, and within these domains a RBS can operate as a consultant, problem solver, an expert or a decision maker [11]. The best usage of RBSs is applied to systems where the solution of certain problems cannot be achieved using conventional programming, or where an algorithmic approach cannot provide an easy solution. A rule-based system consists of a *rule-base*; an *inference engine*; and a *working memory*. In some applications, a *user interface* may be present through which input and output signals are received and sent, however, it is not necessarily a part of the basic reasoning process.

- Rule-base contains a set of rules, specifically the appropriate knowledge encoded into IF-THEN rules for a given problem;
- Working memory contains a set of facts which represent the initial state of the system;
- Inference engine controls the system execution, consisting of three phases: the *match* phase, the *select* phase and the *execute* phase. The *match* phase compares the conditions (IF) of all rules to working memory. A match for every condition in a rule constitutes an instantiation of that rule. A rule may have more than one instantiation. The set of all rule instantiations collectively form a set, called the *conflict set*, which is passed through the *select* phase. In the *select* phase a *reasoning strategy* (or a conflict resolution strategy) determines a single instantiation, all instantiations or a subset of conflict set, which is passed to the execute phase. In the absence of an explicit reasoning strategy, all the instantiations are selected for execution. The *execute* phase then performs the actions of those instantiations passed specified in its THEN clause. These actions can modify the working memory, for example newly generated facts can be added to the working memory, some old facts can be

deleted from the working memory or do anything else specified by the system designer. The cycle begins again with the *match* phase and the process continues until no more rules can be matched or a problem is solved.

One of the advantages of a RBS is that the rules are stored separately from the code. The rule-base can be altered without making any changes to the program code. The rule of a system has to follow a syntax, however, there are not any specific guidelines for a rule. For example, in some frameworks, a rule can have multiple right-had-side (THEN) or actions, while some support only one action (based on Horn-clause rules). The condition (IF) of the rule carries the knowledge part. The set of rules into a rule-base makes a knowledge base. The knowledge base is iterated for pattern matching with facts and that is the most expensive part of execution in terms of computation and time [12]. The effects can be seen on memory too. In order to speed up the matching process, various algorithms and solutions have been proposed, including the RETE algorithm [13]. In order to get a much clearer picture, in a later section of this paper, we will discuss different match problems that are common in RBSs.

2.2 Rule-base design

In AI, knowledge engineering is an area that develops knowledge-based systems. It follows a systematic process that creates rules to apply to data in order to simulate the thought process of human experts. In that process the main tasks include knowledge base design and the implementation of inference engine [10]. While we are not going to present here the whole knowledge engineering process, we would like to briefly discuss the importance of knowledge representation pertaining to the rule-base design. Knowledge representation is a method by which a knowledge engineer can model the facts and relationships of the domain knowledge, and it is of major importance in expert systems. This is due to the fact that expert systems are often designed for a certain type of knowledge representation based on inference rules. Furthermore, knowledge representation affects the overall development process, including, the efficiency, speed and maintenance of the systems. However, there does not exist any single general formalism suitable to represent knowledge for all purposes [14]. In rule-based expert systems, much of the knowledge is represented as rules. There are various knowledge representation techniques exist, the logic-based technique is one of them which is popularly used both in theory and practice of rule-based expert systems. Among many feasible logical knowledge representation languages, propositional logic serves as a useful language for encoding rule-based systems [15]. The basic logical form of propositional rules is Horn clause of the form $P_1, P_2, \ldots, P_n \rightarrow P$. The premise of the rule (left-hand side), which is a conjunction of positive literals, is called the antecedent or body of the rule, while the right-hand side of the arrow is called consequent or head of the rule. If a Horn clause has no body at all, it is called a definite clause or a fact. A more complex rule may contain consequent part composed of several propositions. Although propositional logic has many practical applications, being a simple knowledge representation language it is often not suitable for modelling real life complex systems. For example, propositional logic cannot directly talk about properties of individuals or relations between individuals. Thus most modern knowledge representation and reasoning approaches are based on description logics (decidable fragments of first-order logic) and rule-based formalisms (including SWRL) constituting the most prominent language families [16]. In this regard, ontologies are an important knowledge representation technique, often used widely in many applications of web-oriented intelligent systems [17]. Since we are interested in designing expert systems as multi-agent context-aware reasoning systems, knowledge that is exchanged and shared between agents is interpreted according to a model which is achieved using ontology. In the context of knowledge-based systems, an ontology can be considered as the definition of the objects and relations forming the basis for the conceptualization and model of an expert system. The Protégé ontology editor and knowledge-base framework [18], an open-source platform, helps to construct domain models and knowledge-based applications with ontologies. In [19], a tool has been developed to extract Horn-clause rules from multiple OWL 2 RL ontologies. The extracted rules are used to design our rule-based context-aware agents.

2.3 Ontology-based context representation and reasoning

We view context is any information that can be used to identify the status of an entity [6]. A context can be 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. According to [6]—"if a piece of information can be used to characterize the situation of a participant in an interaction, then that information is context". For example, we can represent a context "Mary has a caregiver named Fiona" as *(Mary, hasCareGiver, Fiona)*. Here, the caregiver *Fiona* of a patient *Mary* is dynamically identified based on the care status of *Fiona*. This context can be expressed in predicate calculus as *hasCareGiver(Mary, Fiona)*.

For context modelling we use OWL 2 RL, a language profile of the new standardization OWL 2, and based on pD^* [20] and the description logic program (DLP) [21]. We choose OWL 2 RL because it is more expressive than the RDFS and suitable for the design and development of rule-based systems. An OWL 2 RL ontology can be translated into a set of Horn clause rules based on [21]. Furthermore, we can express more complex rule-based concepts using SWRL [22] which allow us to write rules using OWL concepts. In our framework, a context-aware system composed of a set of rule-based agents, and firing of rules that infer new facts may determine context changes and representing overall behaviour of the system.

2.4 Context-aware systems as resource-bounded agents

A key application of multi-agent systems research is distributed problem solving (DPS). Distributed approaches to problem solving allow groups of agents to collaborate to solve problems. Smith and Davis argue that—"*distributed problem solvers offer advantages of speed, reliability, extensibility, the ability to handle applications with a natural spatial distribution, and the ability to tolerate uncertain data and knowledge. Because such systems are highly modular they also offer conceptual clarity and simplicity of design*" [23]. However, while working on the DPS setting, the computational (time and space) and communication resources required by a reasoning agent(s) to solve a given problem is of considerable interest. In our framework, we consider systems having constraint on various resources namely time, memory, and communication. This is because many context-aware systems often run on tiny devices including PDAs, mobile phones, smart phones, GPS system, and wireless sensor nodes. These devices usually operate under strict resource constraints, e.g., battery energy level, memory, processor, and quality of wireless connection. In [16], a formal framework has been presented for modelling context-aware systems and a logic $\mathcal{L}_{DROCS}$ is developed which extends the temporal logic CTL^* with belief and communication modalities and incorporates defeasible reasoning [24] technique to reason about agents' behaviour. Each agent's memory usage is modelled as the maximal number of contexts to be stored in the agent's memory at any given time. That is, we assume that each agent in a system has bounded memory size which allows maximal number of contexts to be stored at any given time. We divide agent's memory into two parts as rule memory (knowledge base) and working memory. Rule memory holds set of rules, whereas the facts are stored in the agent's working memory. Working memory is divided into static memory and dynamic memory. The dynamic memory of each agent is bounded in size, where one unit of memory corresponds to the ability to store an arbitrary context. The static part contains initial information to start up the systems, e.g., initial working memory facts, thus its size is determined by the number of initial facts. The dynamic part contains newly derived facts as the system moves. Only contexts stored in dynamic memory may get overwritten if it is full or a conflicting context is derived. Similarly, each agent has a communication counter, which starts with value 0 and incremented by 1 each time while interacting (sending/receiving a message) with other agents, and is not allowed to exceed a preassigned threshold value.

To clarify these ideas, let us consider a simple example of a distributed problem solving consisting of two agents. Agents reason using (Horn clause) rules and communicate via message passing. The knowledge bases and initial working memories of agent 1 and agent 2 are shown in Fig. 1. The goal is to derive context $C_5(a)$. Note that in the rule $R_{ik}\ m\ :\ body \rightarrow head$, R_{ik} represents k^{th} rule of agent i and the number m represents annotated priority of the rule. Note also that OWL 2 is limited to unary and binary predicates and it is function-free. Therefore, when we develop ontologies and translate them into a set of Horn clause rules, in the Protégé editor all the arguments of Ask and $Tell$ are represented using constant symbols [16]. An agent can update its working memory by performing one of the three possible actions:

Rule firing a matching rule instance in the current state (possibly overwriting a context from the previous state);
Comm if agent i has an $Ask(i, j, P)$ (or a $Tell(i, j, P)$) in its current state, then agent j can copy it to its next state (possibly overwriting a context from the previous state);
Idle which leaves its configuration unchanged.

That is, each transition (result of an action) corresponds to a single execution step and takes an agent from one state to another. States consist of the rules, facts (contexts), and other resource counters of the agent. A *step* of the whole system is composed of the actions of each agent, in parallel. A problem is considered to be solved if one of the agents has derived the goal. An example run of the system is shown in Table 1. In the table, a newly inferred context at a particular step is shown in blue text. For example, antecedents of rule R_{11} of agent 1 match the contents of the memory configuration and infers new context $C_3(a)$ at step 1. A context which gets overwritten in the next state is shown in red text, and a context which is inferred in the current state and gets overwritten in the immediate next state is shown in cyan text. In the memory configuration, for each agent, left side of the red vertical bar | represents static part of the working memory and its right side represents its dynamic part. It shows that the size of the dynamic part of agent 1

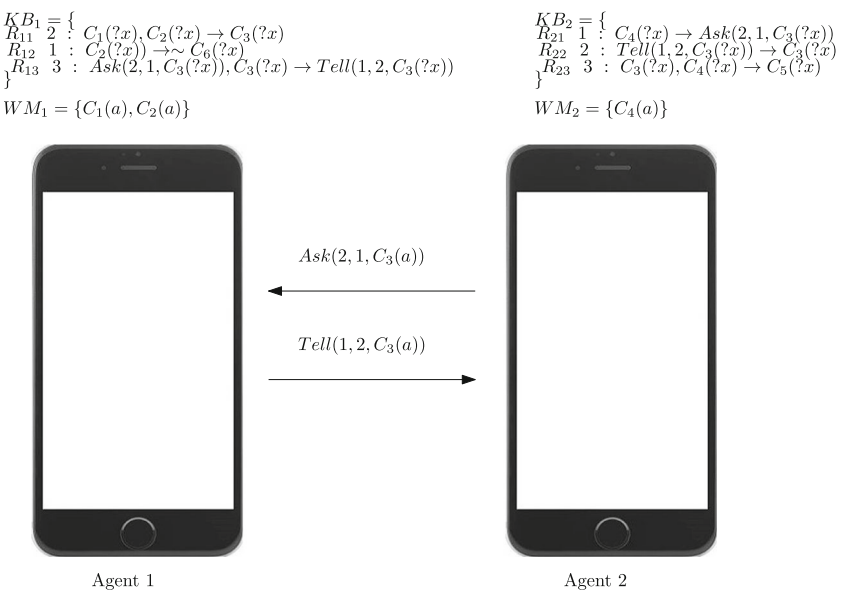

Fig. 1 Distributed problem solving

is 2 units and that is of agent 2 is 1 unit. Note that, there may not be any matching rule instance or there could be multiple matching rule instances at a particular step. Note also that only selected matching rule instances can be fired. That is one rule instance may be selected from the conflict set that has the highest priority. If there are multiple rule instances with the same priority, then rule instance to be executed is selected non-deterministically. In this case the integers represent rule priorities, and we use the convention that smaller integers represent lower priorities. It is evident that in Fig. 1 neither agent can derive (infer) $C_5(a)$ alone. We can observe in Table 1 that the resource requirements for the system to derive the goal context $C_5(a)$ are 2 messages that need to be exchanged by each agent and 6 time steps. Time taken to communicate a fact between agents depends on how we model the communication mechanism. In this example, communication between agents is modelled using Ask and $Tell$ communication primitives, where the cost of communication is paid by both agents, communication takes more than one tick of time, and communication is non-deterministic. We can also observe that, if we reduce the dynamic memory size for agent 1 (and for agent 2) by 1, then the system will not be able to achieve the desired goal. This is a very simple case; however, if we model a more realistic scenario and increase the problem size, the verification task would be hard to do by hand. Therefore it is more convenient to use an automatic method to verify them, for example using model checking techniques [25]. Further discussion of these aspects is beyond the scope of this paper, interested readers are referred to [16].

Table 1 One possible run of the system

	Agent 1			Agent 2		
#Step	Config 1	Action 1	#Msg 1	Config 2	Action 2	#Msg 2
0	$\{C_1(a), C_2(a)\|-, -\}$	–	0	$\{C_4(a)\|-\}$	–	0
1	$\{C_1(a), C_2(a)\|C_3(a), -\}$	Infer	0	$\{C_4(a)\|Ask(2, 1, C_3(a))\}$	Infer	1
2	$\{C_1(a), C_2(a)\|C_3(a),$ $Ask(2, 1, C_3(a))\}$	Comm	1	$\{C_4(a)\|Ask(2, 1, C_3(a))\}$	Idle	1
3	$\{C_1(a), C_2(a)\|C_3(a),$ $Tell(1, 2, C_3(a))\}$	Infer	2	$\{C_4(a)\|Ask(2, 1, C_3(a))\}$	Idle	1
4	$\{C_1(a), C_2(a)\|C_3(a),$ $Tell(1, 2, C_3(a))\}$	Idle	2	$\{C_4(a)\|Tell(1, 2, C_3(a))\}$	Comm	2
5	$\{C_1(a), C_2(a)\|C_3(a),$ $Tell(1, 2, C_3(a))\}$	Idle	2	$\{C_4(a)\|C_3(a))\}$	Infer	2
6	$\{C_1(a), C_2(a)\|C_3(a),$ $Tell(1, 2, C_3(a))\}$	Idle	2	$\{C_4(a)\|C_5(a))\}$	Infer	2

3 Related work

The related work presented in this section is divided into two parts. The first part discusses mobile-based frameworks, and research work which has focused on context-awareness theme based on different techniques including the rule-based approach. The later part focuses on the well known existing inference engine algorithms of rule-based systems.

3.1 Android-based RBS and context-aware systems

A considerable research work has been conducted in the area of social networks. While the discussion of social networks, themselves, is beyond the scope of this paper, they can be regarded as the online presence of a user where a user helps in generating his contextual data along with preferences and interacts with other users with the same interests. In a social network, users put a lot of their personal details, preferences, likes and dislikes etc. These give a considerable amount of contextual information related to a user as can be observed in different research projects, including SociaCircuit platform [26], which monitors different social factors between the users. Based on these factors, it measures the shift in user preferences e.g., habits and opinions. The work presented in [27] focused on finding social relationships among the users, and this provides results based on some data mining tools. Sociometric badge presented in [28] monitors an employee's different activity patterns in the office. It records different data related to the user, and based on that data, within the organization, the user's job satisfaction and interactions quality can be predicted. Similarly, the work presented in [29] monitors a user's activity based on his different mobile sensors, his locations visited, call logs etc. This monitoring then further tries to infer the significant location based on his social activities, different relationships and related information. Recent work based on inferring results or mobile based expert systems still lacks different aspects. For example, in [30], a small expert system is developed which acts as an academic advisor. It has a set of rules which fire based on user provided inputs, and then the system provides advise accordingly. The system is monotonic. It will give the same answer for the same inputs every time, and there is no capacity to run a different set of rules as the interface is linked with its own current set of rules. There also exists work based on client-server architecture such as [31], where a server works as a knowledge base and an Android phone works as a client agent with an application installed to connect to the server and sending some contextual information e.g., location. Similarly, another research work [32] based on iPhone platform, uses the same client-server architecture combined with a rule-based system on the server to provide a safe evacuation in case of emergency cases at a university (case scenario). However, the set of rules used as expert knowledge is not defined in their work. Most of the systems discussed above lack at least one of the following major issues: Context re-usability, Generic modelling, Resource efficiency in terms of space and communication, and efficient rule-based reasoning.

Regarding the issue of *re-usability* of contexts, some existing frameworks provide ontology-based approach such as the work by [33, 34]. However, they do not address the issue of context-aware mobile application development. Some more recent work has effectively used the ontologies for modelling with better resource handling. They have modelled their systems using ontologies, with the bound on resources such as memory and communication [35]. In [36], the authors present an ontology-based framework for rapid prototyping of context-aware application development. It supports a wide user category and their collaboration and cooperation in the applications development. Since it is based on collaborative environment, users have to accept the shared conceptualization of the domain. Three main categories of users based on their technical abilities are High level, Middle level, and Low level. Based on the level of the users, users can use the framework in different environments. The framework, while supporting collaboration and sharing of context, also focuses on the cooperation between users. This cooperation can be synchronous, asynchronous, individual or group based. The cooperation pattern based on the technical abilities can be between developers, developers and end users, and between end users. It has some important components such as context providers, the context manager, programming toolkits and the resource sharing server. However, the use of resource sharing server suggests a limitation on distributed approach, and also the Android limitations demand a more compact and Android compatible framework. The part of matching the rules and facts needs an algorithm, while to implement it on resource-bounded devices it needs an appropriate algorithm that can be tailored according to the chosen platform. The next section discusses some of the state-of-the-art inference engine algorithms and their analysis where required.

3.2 Matching problems, precautions and algorithms

According to Forgy [37], in rule-based reasoning the matching phase can take up to 90% of the whole execution time. The matching phase repeats numerous times, and it starts when new working memory elements are added or removed. This certainly has a vital impact on the overall execution time. The matching time is affected by the size of the rule-base, the number of conditions (IF) in a given

rule, and the number of working memory elements. Since in each rule we have to match the rule conditions with the working memory facts, the time for execution takes longer when there are many conditions in a rule. Other factors that affect it further can be attributed to the number of variables on the LHS of the rule. If some variables are repeated in other rules then it should be binding to the same fact every time. Semi matching rules also create problems as they are not added to the conflict set, however they are tested for qualifying the facts. Rules that are never fired are also checked for eligibility. Long rules with many conditions also create problems and this is called the long chain effect. These are some of the frequently occurring problems. There are some precautions rather than solutions to avoid the match problems. The precautions include saving the state of the rule conditions, keeping track of the facts in view of the rules which are most probably be affected with the changes in the WM, sharing the conditions of rules with similar rules etc. However, these precautions/solutions have their own drawbacks. As we have already mentioned, most of the expert system research has tended to focus on high end computers with a lot of available resources, and the solutions to the match problems take advantage of using the abundant resources as state saving, condition saving, and similar other strategies that consume a lot of memory. Our concern, however, is to avoid such issues and to deliver comparable or better results on small devices. These results can be affected with simple strategies such as, e.g., efficient rule-base management. Matching rules part can be improved by different methods, the rule being the main component can drastically improve the overall performance. Simple ordering in conditions of a rule can have a huge effect. If a rule has ten conditions and the first nine conditions match while the last one doesn't, the rule is not eligible for firing and this check wastes the resources for calculating the nine conditions. Instead, if the tenth condition of the rule is checked at first place it will save a lot of computational resources. Researchers have proposed several matching techniques to match the rule conditions in an attempt to improve the overall performance [10, 38]. They include strategies such as sharing conditions, rules ordering, facts ordering etc. These can be carried out while in the design phase of the rule base. One of the prominent pattern matching algorithms that is often used in rule-based expert systems is the RETE algorithm. The RETE algorithm provides a base for many well-known algorithms, including RETE Gator [39], RETE* [40] and Treat [41]. These are eager evaluation algorithm. This kind of algorithm creates the complete conflict set before deciding which one to fire. These algorithms differ from each other on technical differences between their respective networks and making them efficient than their counterparts. However, the basic working mechanism somehow remains the same. Treat claims to use low memory consumption compared to RETE. RETE* is considered as a hybrid of both the RETE and Treat. In all the eager evaluation algorithms the major drawback is considered the worst-case complexity of $O(WM_e^{RC})$, where WM_e refers to the working memory elements and RC represents the number of conditions in a rule. This is not always the case but it is always possible to encounter the worst case when the number of working memory elements is large and the rules carry many conditions. In order to overcome this issue, the idea of lazy evaluation algorithm was proposed. This kind of algorithm provides a concept that in a cycle only one rule has to be activated since we have to fire one rule at the end. This approach has its own advantages, for instance, if a match is found it does not search for more rules. Other algorithms search the rules which are never fired but consume a lot of computational resources. Based on this concept the only algorithm made was Leaps [42]. As mentioned earlier, the RETE algorithm is one of the most widely used algorithms [43], however, many people find it difficult and has more than 1000 lines of pseudo code [44]. In the next section, we elaborate working mechanism of the RETE algorithm and provide its complexity analysis both in terms of time and space.

3.2.1 The RETE algorithm

The RETE algorithm, introduced by Charles Forgy as part of his doctoral studies [37], is widely used in systems where pattern matching is required such as rule-based systems. Based on it, several other algorithms have been developed for high-end computers. It may be noted that the RETE algorithm is used widely in centralized systems. Although it has been computationally improved, it still consumes huge amounts of memory [10] and could create a potential problem especially when it comes to execution on a single device or on small devices [9]. There have been some attempts to port the current RBS systems into the Android platform with little to no success [31, 45]. These have been discussed in our survey work [46], including JESS which is based on the RETE algorithm. The RETE algorithm is no doubt one of the most popular algorithms which is commercially used in large corporates encompassing a large number of business rules. Before going into details, some of the basic terminologies of the RETE algorithm are introduced. It considers the production memory (PM) and working memory (WM). The PM contains different productions or rules. Each rule is represented as a set of conditions on the LHS and its respective actions on the RHS. The WM contains items

which represent facts. The structure of a particular rule is provided below.

```
(
name of the rule
Left-hand-side (one or more conditions)
-->
Right-hand-side (one or more actions)
)
```

Usually, the matching algorithms ignore the action part or the consequent and handle the conditions. The conditions may contain constants and variables. The actions are taken care of by another part of the system once a conflict set has been created. The RETE algorithm makes use of a data flow algorithm for the better presentation of the rule conditions. The network can be further broken down into two main parts, namely the Alpha part and the Beta part. The Alpha part carries out the constant tests on the WM and stores the results in the Alpha memory. This Alpha memory contains the elements of the working memory which successfully pass the constant test for a given condition of a rule. The Beta part handles the joins and beta memory. It does the necessary variable binding between conditions, and stores the results in the join node. Beta memories are then stored along with the semi-matched production rules, as more and more steps are taken. The process is repeated for the rest of the conditions and finally a fully matched production is acquired. Changes in the working memory are conveyed to the Alpha network, and related Alpha nodes adopt the changes. Ultimately, these changes are passed to the Beta network nodes and joins. Any new matches found in the Beta network are updated accordingly until it reaches the end. At the end of the network, we have the production node. When a production node is produced it indicates that a newly matched rule has been found. In the middle of the process, there are two types of activations, namely the left activation and the right activation. The left activation corresponds to the activation of a node by any other node in the Beta network. The right activation refers to the activation of a node by the Alpha memory. The joins in the Beta network can have these two types of activations. Both activations are handled by different procedures and are discussed in the analysis of the algorithm section. An important feature of the RETE algorithm is that it is state saving. It saves the states of the matching process in the Alpha and the Beta memories. A change in the WM does not always affect many nodes in the network. However, the RETE algorithm is not recommended for systems where major changes occur in the working memory [44]. Another feature of the RETE algorithm is its node sharing with productions with similar conditions. Single Alpha memory is used for a rule which has the same common conditions. Figure 2 depicts a logical network illustration.

3.2.2 Analysis of the RETE algorithm

An algorithm based on RETE has been proposed in [44], the author pointed out that RETE slows down with an increased number of production rules. Furthermore, the author argues that the RETE algorithm is not designed for systems where the WM is frequently updated. Moreover, as mentioned before, in the worst case the RETE algorithm can reach to $O(WM_e^{RC})$ [9, 42]. Similarly, in [47] the authors in their comparative study argue that the asymptotic complexity of the RETE algorithm is of $O(n^m)$. In the context of multi-agent systems, the memory consumption of the RETE algorithm is problematic [9]. Since the working memory is not fixed, and the RETE algorithm is well known for its large use of memory, it is not a suitable option for small devices. Memory is mainly consumed when the network is developed to store the states at different levels. It uses a lot of memory when creating Alpha-Beta networks and the space complexity is exponential for both RETE and TREAT [48]. The working memory itself is not fixed and practically all available memory could be occupied with the WM elements. Another issue is that when there are many WM elements and a complex rule with varying conditions. This can lead to the cross-product problem and can take the system into the worst case scenario. Furthermore, an issue with the RETE algorithm as pointed out in [49] is the creation of a lot of child nodes when we have an attribute with multiple values, for example, colour and its values. In that case, the attribute colour node will spread into the number of values (blue, green, black and so on) available. Besides these problems which are pointed out in the research, if we consider the pseudo code of the RETE algorithm and analyse the complexity for the sake of comparison, we find that the Left and Right node activation has a complexity of $O(n^2)$. Let us consider the (Join Node Left activation) high-level Pseudo code from [44] of the RETE algorithm shown in Table 2. Similarly, the procedure of join node right activation from the same source has complexity as shown in Table 3.

These two fragments particularly have the complexity of $O(n^2)$. While the rest of the pseudo code itself is beyond the scope of this paper and is too lengthy to be mentioned here. However, from these code fragments, it is affirmed that the complexity of RETE algorithm cannot be lower than $O(n^2)$.

The typical problems of RBS combined with context-aware systems, re-usability, low resource usage etc. as

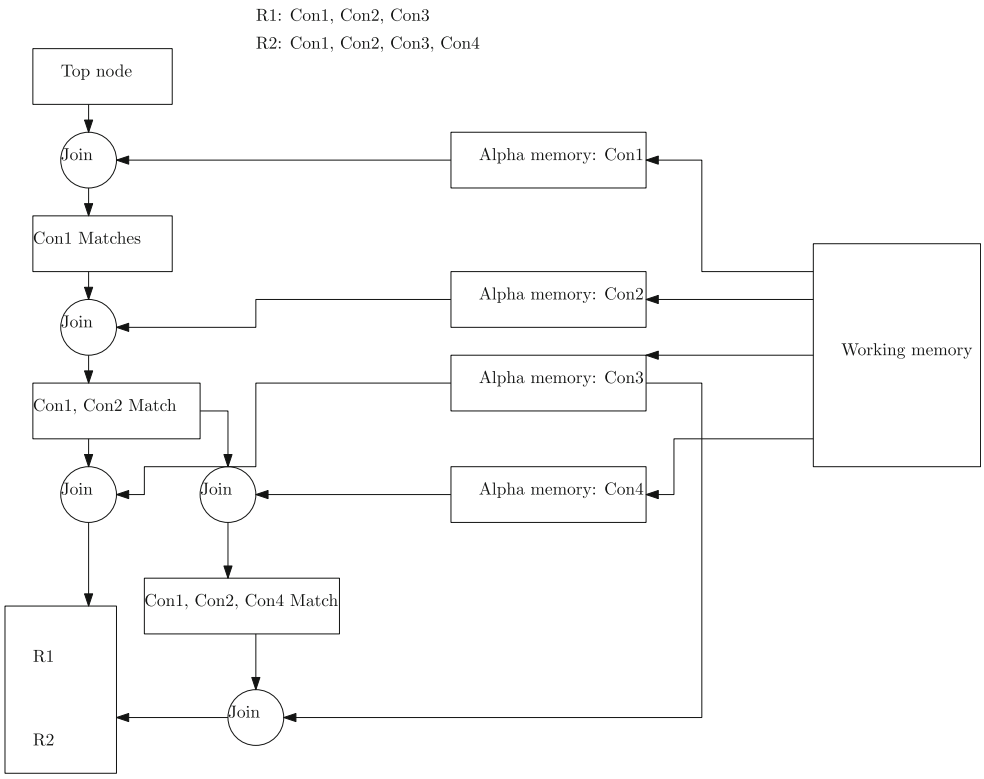

Fig. 2 Rete network illustration

discussed in this section, have provided us with an opportunity to explore further the problems on small devices and to devise an algorithm and context-awareness model that can perform in comparable computation and better memory usage in resource-bounded devices. The development of the framework is designed as an agent-based reasoning system, with each agent having its own set of rules and inference engine and having the capacity to communicate with other devices in a distributed fashion.

4 A motivational analysis of the proposed research approach

With the advancement of resource sharing large-scale cloud computing, the expert systems have also seen their growth on such platforms. Similarly, the use of social networks, which keeps the user engaged and extracts a variety of contextual information from the user such as location, timestamps and related contexts make it an easy task to

Table 2 Left activation algorithm running cost

Algorithm	*Cost*	*Frequency*
START		
IF node.parent just became non empty **then**	c_1	n
relink.to.alpha.memory(node)	c_2	n
If node.amem.items =nil **then**	c_3	n
remove node from the list node.parent.children	c_4	n
For each item **in** node.amem.items **do**	c_5	n
If perform-join-tests(node.tests,t,item.wme) **then**	c_6	n
For each child **in** node.children	c_7	n^2
do left-activation(child,t, item.wme)	c_8	n^2
END		

Table 3 Right activation algorithm running cost

Algorithm	*Cost*	*Frequency*
START		
IF node.parent just became non empty **then**	c_1	n
relink.to.beta.memory(node)	c_2	n
If node.parent.items =nil **then**	c_3	n
remove node from the list node.amem.successors	c_4	n
For each t **in** node.parent.items **do**	c_5	n
If perform-join-tests(node.tests,t,w) **then**	c_6	n
For each child **in** node.children	c_7	n^2
do left-activation(child,t, w)	c_8	n^2
END		

connect the expert systems with the social networks and utilise the available contextual data. However, the part where the resources are limited is widely ignored. Also, if a system is supposed to care for elderly people/patients or is a remote monitoring system, the social network does not have a big role to play in such cases. This is because the chances are that a patient might not have his social networking account or may not be using it actively. Expert systems which are supposed to help humans are now relying more and more on resource hungry algorithms and high-end technology, while the usage of such systems on small-scale devices is nowhere to be seen. There is certainly a need for such dedicated systems [50]. The scenario above provides the main motivation for undertaking this study and proposing a context-aware system development framework considering resource-bounded devices. That is a system which instead of learning from the behaviour and saving it into a massive memory, should be working on the current inputs it has and the contexts that are available to the system. The author in [51] has provided a very good insight into scenarios as to where and when a context-aware application should respond. The work has human being opinions on how the reaction of context should be. In other words, the response should be non-monotonous and should be intelligent enough to use the context wisely for a better user experience, rather than annoying a user whenever a trigger is found for a particular context e.g., an alarm or alert. A context-aware system should detect if a user is working on his smartphone and then the wakeup alarm should not start. Instead, it should be aware that the screen is already on and that the user is awake. The development of rule-based context-aware systems will cater to the needs of the expert systems deployment in remote areas. The scope of such a system is not only limited to human use, but an expert system deployed in a remote area could be used for flood forecasting, and provide remote care for patients more easily. Elder-care or assisted living systems could be managed, to mention but a few of the possibilities. Furthermore, as we explored, it is realized that the computation time is directly related to the input provided. Since the rules provide the main expert knowledge, a rule-based system has to iterate through all the rules. Naturally, if the number of rules is kept to a minimum then it will take less time to generate output. Therefore, we also proposed a very interesting method of personalization in the rules, which drastically reduces the number of rules to be processed without affecting the expected system output. The output produced considering all the rules and the set of reduced rules should be the same provided the preferences are set accordingly [52, 53].

5 Proposed framework

In order to have an efficient RBS on small devices, we need to take into account particularly the memory consumption, the communication system and the rule-base size along with the rest of the components. Contrary to other algorithms, the proposed algorithm does not store any kind of states of conditions. Only variables and their values are stored in key-value pairs whenever a variable and its value are found, thus occupying space only when a variable needs binding. Once a variable has been bound to a value, it can be re-used for the same variable in the future. In order to run a system on a small device, the rule base has to be small in size. Reducing rules can affect the accuracy of a system, and our novel approach towards the reducing of rules is based on the preferences provided by the system designer as well as the end user. This only process a subset of rules that are required for a particular scenario. As an example, a user who is in office does not need rules which deal with his home. Processing the home-based rules would simply increase the complexity of the whole system. Therefore, we do not consider them unless required. The enhanced preference mechanism is based on our previous works from [52, 53]. The structure of rule ordering can be opt-in as an added optimization feature. Our rule matching mechanism to create a conflict set checks the predicate first.

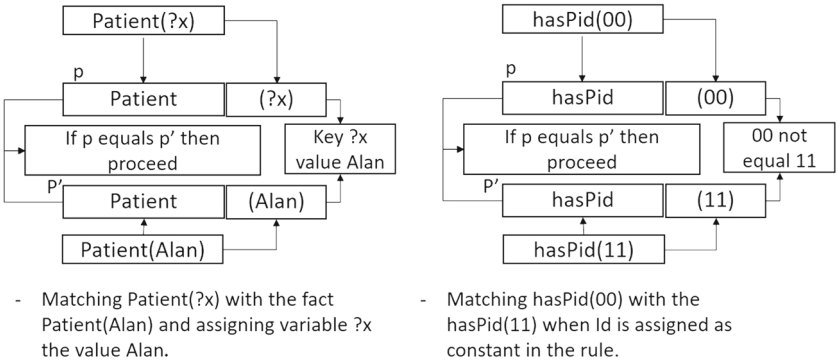

Fig. 3 Different matching scenarios of the proposed algorithm

If a predicate match is found in the working memory then it further checks the rule condition, otherwise discards the rule without moving any further. A flag is set to monitor each match. If a flag value is 1 for a given condition in a rule then it proceeds to the next condition. Whenever 0 is encountered it represents that the rule cannot fully match with the facts, the process is terminated and the next rule is selected for a check. A typical rule format of our framework can be found in [52], while some changes are made when preference is intended. The typical structure of a rule looks like this:

$$m : P_1, P_2, \ldots P_n \rightarrow P : F : CS$$

where $n \geq 0$, m represents the priority of the rule followed by the LHS and RHS, F is a flag which shows the nature of the rule, and CS is used for preferences.

5.1 Matching algorithm

In this section, we would like to illustrate how the matches are performed with a simple example. This will give a basic idea of how the algorithm works. Although there are other checks performed on different levels which decide if the next condition is worth checking or not, which are not discussed in Fig. 3. In Fig. 3, two different scenarios are provided. In the left-hand side, there is a condition containing a variable. The algorithm first matches the predicate part. Once the predicate part matches with any one of the WM facts then it proceeds to the next step and performs different checks. In this case, we have a variable $?x$. The algorithm then assigns the variable $?x$, and terms it as a key. The key then stores the value as Alan. On the right-hand side, there is a comparison of a constant. In the rule condition, instead of a variable this time we have a constant. Since the predicate matches, it checks that there is no variable in the rule and the only term available is a constant which does not match with the constant term 11. Hence the rule is discarded without any further processing of the remaining rule conditions. The algorithm can process both the variable and the constant in the same condition of a rule. Our proposed algorithm is based on the simplicity. The size of the working memory can be adjusted automatically via different strategies (see Section 5.2). The devices can trigger communication when rules are specifically required to communicate. Furthermore, the preferences as discussed in the next section reduce the number of processable rules to the least possible number without affecting the outcome of the system.

5.1.1 The time and space complexity of the proposed algorithm

In this section, we analyse the asymptotic complexity of the proposed algorithm (depicted in Table 4) both in terms of the time and space. As discussed earlier, the complexity of the RETE algorithm as well as other eager evaluation algorithms is of $O(WM_e^{RC})$ [9, 42]. Charles Forgy has also mentioned the same as a worst case for the effect of working memory size on the number of tokens. However, the proposed algorithm shows a very promising result especially in terms of space management which is one of our primary goals to achieve. In the Algorithm presented in Table 4, the symbols are defined as **R:** Rule-Base, **WM**: Working Memory [$\mathbf{R}_s$: A single rule, $\mathbf{R}_i$: A rule instance, $\mathbf{R}_b$: Rule body, $\mathbf{R}_{ib}$: Rule instance body, $\mathbf{R}_c$: Rule consequent, $\mathbf{R}_a$: Rule atoms in the body, $\mathbf{R}_{ap}$: Rule atom predicate, $\mathbf{R}_{at}$: Rule atom terms, $\mathbf{F}_c$: Current fact, $\mathbf{F}_{cp}$: Current fact predicate, $\mathbf{F}_{ct}$: Current fact terms, **PM**: Pattern matching, $\mathbf{P}_{ra}$: Patterns in rule body, **VAR**: Arraylist to hold KEY and VALUE.

The complexity of the proposed algorithm is of $O(n^2)$. Its worst-case complexity is considerably low and it is efficiently usable on any resource-bounded devices. The conflict resolution input depends on the size of the conflict set. It iterates through the conflict set and finds the highest priority rule instance for execution. The time complexity of the conflict resolution code is of $O(n)$, depicted in Table 5. Its space complexity is also $O(n)$ as there is only one array that holds the conflict set elements. The rule execution is quite straightforward. When a rule instance is passed by

Table 4 Conflict set generation algorithm complexity

Algorithm	*Cost*	*Frequency*
For r=0 to size of **R**	c_1	$n+1$
Clear VAR	c_2	n
$\mathbf{R}_s$ =R[r]	c_3	n
Find patterns in R_b of R_s	c_4	n
Add to Array P_{ra}	c_5	n
Flag: Array of size equal to $\mathbf{P}_{ra}$	c_6	n
For ra = 0 to size of $\lvert\mathbf{R}_a\rvert$ **do**	c_7	$n(n+1)$
Select $\mathbf{R}_a$[ra]	c_8	n^2
Seperate R_{ap} from R_{at}	c_9	n^2
For f=0 to size of **WM do**	c_{10}	mn^2
$\mathbf{F}_c$ = **WM[f]**	c_{11}	mn^2
Seperate $\mathbf{F}_{cp}$ from $\mathbf{F}_{ct}$	c_{12}	mn^2
if $\mathbf{R}_{ap}$ == $\mathbf{F}_{cp}$ **then**	c_{13}	mn^2
if $\mathbf{R}_{at}$ == $\mathbf{F}_{ct}$ | pattern($\mathbf{R}_{at}$==$\mathbf{F}_{ct}$) **then**	$c_{14}+c_{15}$	mn^2
Add 1 to **flag**	c_{16}	mn^2
KEY = R_{at}	c_{17}	mn^2
VALUE = $\mathbf{F}_{ct}$	c_{18}	mn^2
if(VAR does not contain KEY) **then**	c_{19}	mn^2
Add KEY to **VAR**	c_{20}	mn^2
Add VALUE to VAR	c_{21}	mn^2
else Add 0 to **Flag** and Exit Loop	c_{22}	mn^2
if Flag does not contain 0 **then**	c_{23}	n
For var=0 to size of **VAR do**	c_{24}	$n(n+1)$
Key = VAR[var]	c_{25}	n^2
Value = VAR[var + 1]	c_{26}	n^2
$\mathbf{R}_i$ Replace **Key** with **Value** in $\mathbf{R}_s$	c_{27}	n^2
$\mathbf{R}_c$ = consequent(R_i)	c_{28}	n^2
var ← var+2	c_{29}	n^2
if WM !contain R_c **AND CS** !contain R_i **then**	$c_{30}+c_{31}$	n
Add R_i to **CS**	c_{32}	n

the conflict resolution phase, it is ready to be fired. The fired rule can have different impacts. For example, it can add something to the working memory, delete something from the working memory, initiate communication as in the case of *ask/tell* rules or simply reach the goal and terminate the process. The terms used in Table 5 are defined as **CS**: Conflict set, $\mathbf{P}_o$: Priority Operator, **SPR**: Same priority rules, $\mathbf{C}_{ics}$: An element of CS, $\mathbf{R}_{ip}$: Rule instance priority.

In Table 6, the algorithm for executing a selected rule instance and its corresponding complexity is analysed. The complexity of this algorithm is of $O(1)$. In terms of space, the algorithm only reads from the memory which is already calculated in the previous algorithms and creates no new space to be added. The terms used in the algorithm are defined as **to_ fire**: A selected rule instance to be fired, $\mathbf{R}_c$: A communication rule instance, $\mathbf{R}_g$: A rule instance contains a goal context, $\mathbf{R}_d$: A deduction rule instance, $\mathbf{R}_f$: Rule Flag, $\mathbf{R}_{cons}$: Consequent, **MAX_SIZE**: memory size.

5.2 The working memory adaptation

According to the theoretical framework [16], the working memory of an agent has to be bounded. This also helps to achieve a better running time. If the working memory size is not fixed the worst case complexity can be increased drastically. In order to maintain a balanced working memory, a few methods are proposed, and a user can opt for any one of them.

Distinct consequences In the database analogy, the distinct returns all the results so that the duplicated values are only shown once instead of repeating them. Similarly, the working memory's maximal limit can be put equal to the

Table 5 Conflict resolution algorithm running cost

Algorithm	*Cost*	*Frequency*
P_o **=0**	c	1
For cs = 0 to size of **CS**	c_1	n
C_{ics} = **CS[cs]**	c_2	n
get R_{ip} from C_{ics}	c_3	n
if $R_{ip} > P_o$	c_4	n
$P_o = R_{ip}$	c_5	n
$to\_fire = C_{ics}$	c_6	n
end		
else if $P_o == R_{ip}$ **then**	c_7	n
Add C_{ics} to **SPR**	c_8	n
end		
end		
if SPR > 0 then	c_9	1
Add to_fire to **SPR**	c_{10}	1
to_fire = random(SPR)	c_{11}	1
end		
END		

number of distinct consequences of the rules. If there are n number of rules and n' number of distinct consequences then the size of the WM would be n'. Note that there could be some rules which have the same consequences, therefore $n' \leq n$.

Maximal size of the preference sets In this technique, the preference sets are taken into consideration [52]. It is more complex than the previous method, and with a more space saving mechanism when preferences are supposed to be implemented. This mechanism considers the rules

Table 6 Cost for executing a selected rule instance

Algorithm	*Cost*	*Frequency*
START		
If R_g then	c_1	1
If R_{cons} is a conflicting context **Then**	c_2	1
Overwrite the contradictory context with $\mathbf{R}_{cons}$	c_3	1
Else If |**WM**| < MAX_SIZE then	c_4	1
Add $\mathbf{R}_{cons}$ to **WM**	c_5	1
Else Overwrite an existing context with $\mathbf{R}_{cons}$	c_6	1
Goal Reached	c_7	1
Execution Halts	c_8	1
Else		
If $\mathbf{R}_{cons}$ is a conflicting context	c_9	1
Overwrite the contradictory context with $\mathbf{R}_{cons}$	c_{10}	1
If R_c then	c_{11}	1
initiate communication module	c_{12}	1
Else If |**WM**| < MAX_SIZE	c_{13}	1
Add $\mathbf{R}_{cons}$ to **WM**	c_{14}	1
If R_c then	c_{15}	1
initiate communication module	c_{16}	1
Else Overwrire an existing context with $\mathbf{R}_{cons}$	c_{17}	1
If R_c then	c_{18}	1
initiate communication module	c_{19}	1
END		

in different preference sets and takes the maximal size of distinct consequences of the preference sets.

System designer assigned In this case, it's up to the system designer to assign the minimal memory size and verify the system behaviour before its implementation [16].

6 Preferences

The preferences provide a novel approach to reduce the overall load from the inference engine. The mechanism of preferences is designed in a way so that it can cater to personalized services to the user and it also reduces the number of rules an inference engine has to process. In this way, it handles two different operations simultaneously. The preferences are further divided into three sub-approaches. The sub-approaches depend on the scenario and are purely based on the user's choice.

Context-based preference The context-based preference is the simplest one. It makes a subset of rules, based on the user's selected context. The rules are grouped together by the same context set indicator or CS. A single rule can be a member of different subsets. Once the user selects the context, it can proceed to the next step by creating a subset from the main rule base. Although it has advantages, it may not work when a user anticipates some context in the future to appear and it is not selected in the preference set. For that reason, the derived context based preference is used.

Derived-context based preference When a user is expecting some context to appear in the future and the user wants to enable the preference on that context, then it can be enabled by putting certain rules in a category which a user then keep under watch until the preferred context is derived. As an example scenario, if a user visits a hospital for some reason other than for a check-up, then the rules associated with the person being a patient should not execute. However, if the user is visiting the hospital and his condition is detected as being ill, then the patient rules should apply. Thus, depending on the user context derived, the corresponding subset of rules will be selected to be processed by the inference engine.

Live preference Live preference comes in handy when a user wants to monitor some context continuously until it occurs. For example, if a user wants to keep logging the GPS unless a certain point comes to execute some rules. Once the system detects the context, the preference set is enabled and vice versa. A good example is a user applying some specific rules on a Sunday of the week, in that case, the context of a day is monitored until it becomes Sunday. On other days normal rules will be selected for processing by the inference engine.

We refer the interested reader to [52] for a more detailed discussion on preferences.

7 A case study

To illustrate the functionality of the proposed framework, a prototype system has been implemented which is based on a case scenario of a user's daily activities. In fact, we reimplemented an example scenario introduced in [52] by incorporating more agents into the system in order to design and capture a more complex interactive behaviour of context-aware applications. This also helps us exploring context-aware cross-device interactions between smart-phones and a blood pressure and heart rate monitoring device. We implemented the application system on four sensor-rich Android smart devices (i.e., smart-phones as well as a blood pressure and heart rate monitoring device) and collected data from typical daily activities of a real user. As explained earlier, carrying these devices implies that various sensors and reasonable computational power are always available. Therefore, acquiring low-level contextual data can be used to infer high-level contexts. That is, contexts can be recognized by analysing the data from the sensors. The system consists of 24 agents, some of them are Android-based agents having their own knowledge-base and inference engine. However, some of the agents are only capable of sensing environmental data and send them to other agents in the system and are not able to make inferences on their own. In order to capture the complete scenario, some agents have been simulated. For example, most of the sensors used to model the Smart home and Smart office are simulated. Each of these sensors is assumed to have the capability of generating (simulated) sensor data values. For example, to sense information about milk availability, it is assumed that the milk container is labelled with an RFID tag containing information about the available amount and expiration date, and the smart refrigerator is equipped with an RFID reader, which reads the tag when milk is placed inside the refrigerator. The case study focuses on a normal routine of a user. A system response is checked at several different locations e.g., home, office, market, and health centre. In order to model the agents, the D-Onto-HCR tool [19] has been used to extract 200 rules from three smart domain ontologies. In the following, we briefly describe the agents which are used to model the example system.

7.1 Agents

- **Smart home:** the smart home provides services that are specific to the home user. It has eleven different agents, which work together to serve the user in a better way.

  1. *Authorization sensor:* it checks if the user is authorized to use the services or not.
  2. *Motion detector:* this agent detects the motion of the user to determine the user presence.
  3. *Light sensor:* it works with lights, especially turning the lights on and off.
  4. *Aircon controller:* this sensor controls the working of the air conditioner based on temperature.
  5. *Home controller sensors:* this sensor checks the occupancy and authorization of the user at home.
  6. *Temperature sensor:* it senses the temperature of a room and accordingly interacts with the Aircon controller.
  7. *Door control sensor:* this sensor is attached to a door and it can open/close the door OR it can identify if the door is opened.
  8. *Gas leak detector:* it detects the gas leakage, and if the leakage is found it can alert the user.
  9. *Smoke sensor:* similar to the gas detector, it detects the smoke and fires the alarm in case of smoke is detected.
  10. *GPS sensor:* it detects the user location and when required sends it to the other agents.
  11. *Smart fridge:* it monitors different items in the fridge and their quantity.

- **Smart office:** the smart office facilitates the user in the office by keeping the environment comfortable according to its rules.

  12. *Authorization sensor:* it checks if the user is authorized to use the services or not.
  13. *Smart chair:* it detects if the user is sitting in the office, it also reminds the user to change posture or walk in case it detects that the user has been sitting for a long time.
  14. *Light lamp:* it controls the lights at the office.
  15. *Windows blinds:* it controls the window blinds to open or close them.
  16. *Aircon controller:* this sensor controls the air conditioner based on the sensed environmental temperature.
  17. *Temperature sensor:* it senses the temperature of a room and accordingly interacts with the Aircon controller.
  18. *GPS sensor:* it detects the user location and when required sends it to the other agents.

- **Smart health-care:** It is responsible for monitoring the user health.

  19. *Patient care agent:* this agent is responsible for tracking and monitoring user medical conditions.
  20. *Blood pressure monitor:* it tracks user's blood pressure and sends the reading values to the patient care agent.
  21. *Diabetes monitor:* it tracks user's blood glucose levels and sends the reading values to the patient care agent.
  22. *Fever monitor:* it tracks user's temperature and sends the reading values to the patient care agent.
  23. *GPS sensor:* it detects the user location and when required sends it to the other agents.
  24. *Care giver:* it gets notified in an emergency and/or non-emergency situation.

7.2 Example of preferences

We briefly explain here the use of preferences. Let us consider the patient care agent which has a variety of rules besides those presented in Table 7. In the table, the rule categories are labelled at the top and the left side of the table. Any rule that does not have any CS indicator is a general rule, represented by " − " in the context set, and will be added to every subset that is created for a preference set. However, there are some context based as well as derived context based preferences are also shown in the table. The CS indicator *hasHRCategory(Alan, Poor)* implies that if this context appears in the agent's working memory (which is basically a context deduced or inferred from other rules), then the corresponding Emergency rule will be added to the preference set. This means that Emergency rules will be added to the preference set only when the context *hasHRCategory(Alan, Poor)* is derived from the previously used active rules, otherwise, these rules will not be processed. The CS indicator *hasLocation(Alan, Home)* indicates a live preference and will be invoked only when the GPS sensor sense the location of Alan is at Home. Similarly, the rest of the rules will only be added to the preference set when the user is physically present at Home and the device(GPS) detects the user location indeed at Home. Thus, instead of considering all the rules at once to form the knowledge-base of the agent, the size of the preference set (active rules) will increase or decrease based on the CS indicators and the preference mechanism applied in a given scenario.

7.3 An execution scenario analysis

In our experiment, we have executed various case scenarios to understand the system behaviour, with preferences applied

Table 7 Blood pressure and heart rate rules

Category	m	Corresponding rule	F	CS
Blood pressure category rules				
Low BP	1	Person(?p), hasSystolicBloodPressure(?p,?sbp), hasDiastolicBloodPressure(?p, ?dbp), lessThan(?sbp, '90), lessThan(?dbp,60) ⟶ hasBPCategory(?p,LowBP)	D	–
Normal	1	Person(?p),hasSystolicBloodPressure(?p,?sbp), hasDiastolicBloodPressure(?p, ?dbp), greaterThan(?sbp,90), greaterthan(?dbp,60), lessThan(?sbp,120), lessThan(?dbp,80) ⟶ hasBPCategory(?p,Normal)	D	–
Pre high	1	Person(?p), hasSystolicBloodPressure(?p,?sbp), hasDiastolicBloodPressure(?p, ?dbp),greaterThan(?sbp,120), greaterThan(?dbp,80),lessThan(?sbp,140), lessThan(?dbp,90)⟶ hasBPCategory(?p,PreHigh)	D	–
High	1	Person(?p), hasSystolicBloodPressure(?p,?sbp), hasDiastolicBloodPressure(?p, ?dbp), greaterThan(?sbp,140), greaterThan(?dbp,90)⟶ hasBPCategory(?p, HighBP)	D	–
Heart rate category rules				
Athlete	1	Person(?p), hasHeartRate(?p,?hrt), greaterThan(?hrt,48), lessThan(?hrt,55) ⟶ hasHRCategory(?p, Athlete)	D	–
Excellent	1	Person(?p), hasHeartRate(?p,?hrt), greaterThan(?hrt,54), lessThan(?hrt,62) ⟶ hasHRCategory(?p,Excellent)	D	–
Good	1	Person(?p), hasHeartRate(?p,?hrt), greaterThan(?hrt,61), lessThan(?hrt,66) ⟶ hasHRCategory(?p,Good)	D	–
Above Average	1	Person(?p), hasHeartRate(?p,?hrt), greaterThan(?hrt,65), lessThan(?hrt,71) ⟶ hasHRCategory(?p,AboveAverage)	D	–
Average	1	Person(?p), hasHeartRate(?p,?hrt), greaterThan(?hrt,70), lessThan(?hrt,75) ⟶ hasHRCategory(?p,Average)	D	–
Below Average	1	Person(?p), hasHeartRate(?p,?hrt), greaterThan(?hrt,74),lessThan(?hrt,82) ⟶ hasHRCategory(?p,BelowAverage)	D	–
Poor	1	Person(?p), hasHeartRate(?p,?hrt), greaterThan(?hrt,81) ⟶ hasHRCategory(?p,Poor)	D	–
Some example rules to derive different situations				
Emergency	2	Patient(?p), hasBPCategory(?p,HighBP), hasHRCategory(?p,Poor) → hasSituation (?p,Emergency)	D	hasHRCategory (Alan, Poor)
Emergency	2	Patient(?p), hasBPCategory(?p,PreHigh), hasHRCategory(?p,Poor) → hasSituation (?p,Emergency)	D	hasHRCategory (Alan, Poor)
Emergency	2	Patient(?p),hasBPCategory(?p,Normal), hasHRCategory(?p,Poor) → hasSituation (?p,Emergency)	D	hasLocation (Alan, Home)
Emergency	2	Patient(?p),hasBPCategory(?p,LowBp), hasHRCategory(?p,Poor) →hasSituation (?p,Emergency)	D	hasBPCategory (Alan, LowBP)
Non Emergency	1	Patient(?p),hasBPCategory(?p,Normal), hasHRCategory(?p,Average) → ∼hasSituation (?p,Emergency)	D	hasLocation (Alan, Home)
Non Emergency	1	Patient(?p),hasBPCategory(?p,Normal), hasHRCategory(?p,AboveAverage) → ∼hasSituation (?p,Emergency)	D	hasLocation (Alan, Home)
Non Emergency	1	Patient(?p),hasBPCategory(?p,Normal), hasHRCategory(?p,Good) → ∼hasSituation (?p,Emergency)	D	hasLocation (Alan, Home)

to the rule-base. The preferences are applied at different levels. Here, we just discuss the Smart health-care Emergency scenario. In the Table 8, the first column refers to the Agent. The second column shows the total number of rules used to model the corresponding agent. The third column represents the number of rules after certain preferences are applied,

Table 8 Smart health-care: preference impact on rule base size

Agent	No. of rules	No. of Rules after preference	Reduction in %
19	42	29	30.95
20	9	2	77.77
21	10	2	80
22	8	2	75
23	3	3	0
24	3	3	0
Total	75	41	45.33

and the last column represents the reduction percentage of the rule base. The last row shows the same results but it reflects the overall results considering all the six agents. In the table, it is observed that the number of rules required for the patient care agent to handle the emergency situation is 29, reducing almost 31% of its rule-base size. The system output received is satisfactory and the same results were obtained without applying any preferences. The applied preferences played an important role in reducing the redundant rules in a given scenario. This shows rules can be reduced in order to optimize the inference engine execution speed, and ultimately to reduce total execution time and execution cost.

8 Conclusions and future work

In this paper, we have discussed existing pattern matching algorithms, their drawbacks and usage on small devices focussing specifically on the RETE algorithm. We then proposed an algorithm, tailored according to the needs of resource bounded-devices, especially Android devices. The complexity of the proposed algorithm is of $O(n^2)$. Its worst-case complexity is considerably low and it is efficiently usable on any resource-bounded devices. Using the Android Studio, our proposed rule engine model has been implemented in Android phones, which can be used to run on different Android devices. The application of preferences further enhances the usability of the proposed rule engine by reducing the total execution time and execution cost. In future work, we would like to provide more independent mobility to devices by implementing the system on other technologies, for example, on the LEGO mindstorm robot platform [54]. It will give us more control over the sensors of the LEGO framework while connected to the rest of the agents, to provide better services. Also, as the LEGO agent will be moving, it can make use of the sensors to be used at any place it is deployed. In that case, we can reduce the number of sensors. For example, a single LEGO-based robot can sense the temperature of different rooms by physically moving there instead of using a temperature sensor for every room.

Acknowledgements This work was partially supported by the FET-Computer Science and Creative Technologies at UWE and a Grant received through the Ministry of Science, Technology and Innovation (MOSTI), Govt. of Malaysia as project number 01-02-12-SF0269.

References

1. Bardram JE, Nørskov N (2008) A context-aware patient safety system for the operating room. In: Proceedings of of the 10th international conference on ubiquitous computing, pp 272–281
2. Andrade RMC, Carvalho RM, de Araújo IL, Oliveira KM, Maia MEF (2017) What changes from ubiquitous computing to internet of things in interaction evaluation? Springer International Publishing, Cham, pp 3–21
3. Ballagas MRR, Borchers J, Sheridan JG (2006) The smart phone: a ubiquitous input device. In: IEEE pervasive computing, vol 5, pp 70–77
4. Karlson AK, Iqbal ST, Meyers B, Ramos G, Lee K, Tang JC (2010) Mobile taskflow in context: a screenshot study of smartphone usage. In: Proceedings of the SIGCHI conference on human factors in computing systems. ACM, pp 2009–2018
5. Perera C, Zaslavsky A, Christen P, Georgakopoulos D (2014) Context aware computing for the internet of things: a survey. Communications Surveys & Tutorials, IEEE 16(1):414–454
6. Abowd GD, Dey AK, Brown PJ, Davies N, Smith M, Steggles P (1999) Towards a better understanding of context and context-awareness. In: Handheld and ubiquitous computing. Springer, pp 304–307
7. Durkin J (1994) Expert systems: design and development. Prentice Hall, New York
8. Russell S, Norvig P (2002) Artificial intelligence: a modern approach, 2nd edn. Prentice-hall, New York
9. Lagun E Evaluation and implementation of match algorithms for rule-based multi-agent systems using the example of Jadex. Master's Thesis, Universität Hamburg
10. Giarratano JC, Riley G (2005) Expert systems: principles and programming. Thomson Course Technology
11. Luger G (2001) Artificial intelligence: structures and strategies for complex problem solving. Addison Wesley, Reading
12. Friedman-Hill E (2003) Jess in Action: Java rule-based systems. Manning Publications
13. Forgy C (1982) Rete: a fast algorithm for the many pattern/many object pattern match problem. Artif Intell 19(1):17–37
14. Reichgelt H, van Harmelen F (1984) Criteria for choosing representation languages and control regimes for expert systems. Knowl Eng Rev 1(4):2–17
15. Alechina N, Logan B, Nga NH, Rakib A (2009) Verifying time, memory and communication bounds in systems of reasoning agents. Journal Synthese@Springer-Verlag 169(2):385–403
16. Rakib A, Haque HM (2014) A Logic for context-aware non-monotonic reasoning agents. In: Gelbukh A et al (eds) MICAI'14,

human-inspired computing and its applications, vol 8856. LNCS, Springer, pp 453–471
17. Skillen K-L et al (2014) Ontological user modelling and semantic rule-based reasoning for personalisation of Help-On-Demand services in pervasive environments. Futur Gener Comput Syst 34:97–109
18. The protégé ontology editor and knowledge-base framework (Version 4.1), 2011, http://protege.stanford.edu/
19. Mahfooz Ul Haque H, Rakib A, Uddin I (2017) Modelling and reasoning about context-aware agents over heterogeneous knowledge sources. In: 5th international conference on context-aware systems and applications (ICCASA'16), vol 193. LNICST, Springer-Verlag, pp 1–11
20. ter Horst HJ (2005) Completeness, decidability and complexity of entailment for RDF Schema and a semantic extension involving the OWL vocabulary. Web Semant Sci Serv Agents World Wide Web 3(2-3):79–115
21. Grosof B, Horrocks I, Volz R, Decker S (2003) Description logic programs: combining logic programs with description logic. In: WWW2003. ACM Press, pp 48–57
22. Horrocks I, Patel-Schneider PF, Boley H, Tabet S, Grosof B, Dean M (2004) SWRL: a semantic web rule language combining OWL and ruleML. Acknowledged W3C submission, standards proposal research report: Version 0.6
23. Smith RG, Davis R (1981) Frameworks for cooperation in distributed problem solving. IEEE Trans Syst Man Cybern 11(1):61–70
24. Pollock JL (1987) Defeasible reasoning. Cogn Sci 11(4):481–518
25. Eker S, Meseguer J, Sridharanarayanan A (2003) The maude LTL model checker and its implementation. In: Ball T, Rajamani SK (eds) SPIN2003, vol 2648. LNCS, Springer-Verlag, pp 230–234
26. Chronis I, Madan A, Pentland AS (2009) Socialcircuits: the art of using mobile phones for modeling personal interactions. In: Proceedings of the ICMI-MLMI'09 workshop on multimodal sensor-based systems and mobile phones for social computing. ACM, pp 1–4
27. Jung JJ (2009) Contextualized mobile recommendation service based on interactive social network discovered from mobile users. Expert Syst Appl 36(9):11950–11956
28. Olguín DO et al (2009) Sensible organizations: Technology and methodology for automatically measuring organizational behavior. IEEE Trans Syst Man Cybern, Part B (Cybernetics) 39(1):43–55
29. Eagle N, Pentland AS (2006) Reality mining: sensing complex social systems. In Pers Ubiquit Comput 10(4):255–268
30. Aly WM, Eskaf KA, Selim AS (2017) Fuzzy mobile expert system for academic advising. In: IEEE 30th Canadian conference on electrical and computer engineering (CCECE). IEEE, pp 1–5
31. Sartori F, Manenti L, Grazioli L (2013) A conceptual and computational model for knowledge-based agents in android. WOA@ AI* IA 2013:41–46
32. Abulkhair MF, Ibrahim LF (2016) Using rule base system in mobile platform to build alert system for evacuation and guidance. In Int J Adv Comput Sci Appl 7(4):68–79
33. Gu T, Pung HK, Zhang D (2004) A middleware for building context-aware mobile services. In: IEEE 59th vehicular technology conference, vol 5. IEEE, pp 2656–2660
34. Chen H (2004) An intelligent broker architecture for pervasive context-aware systems. PhD thesis, University of Maryland, Baltimore County
35. Ul-Haque HM (2017) A formal approach to modelling and verification of context-aware systems. PhD thesis, University of Nottingham
36. Guo B, Zhang D, Imai M (2011) Toward a cooperative programming framework for context-aware applications. In Pers Ubiquit Comput 15(3):221–233
37. Forgy CL (1979) On the efficient implementation of production systems. PhD thesis, Carnegie-Mellon University
38. Kang JA, Cheng AMK (2004) Shortening matching time in ops5 production systems. IEEE Trans Softw Eng 30(7):448–457
39. Hanson EN, Hasan MS (1993) Gator: an optimized discrimination network for active database rule condition testing. University of Florida.–Gainesville: CIS Departement
40. Wright I, Marshall JAR (2003) The execution kernel of rc++: Rete*, a faster rete with treat as a special case. Int. J Intell Games & Simulation 2(1):36–48
41. Miranker DP (1990) Treat: a new and efficient match algorithm for ia production systems
42. Miranker DP, Brant DA, Lofaso BJ, Gadbois D (1990) On the performance of lazy matching in production systems. In: AAAI, vol 90, pp 685–692
43. Havelund K (2015) Rule-based runtime verification revisited. Int J Softw Tools Technol Transfer 17(2):143–170
44. Doorenbos RB (1995) Production matching for large learning systems. Technical report, CARNEGIE-MELLON UNIV PITTSBURGH PA DEPT OF COMPUTER SCIENCE
45. Slazynski M, Bobek S, Nalepa GJ (2014) Migration of rule inference engine to mobile platform. challenges and case study. In: Knowledge engineering and software engineering (KESE10), p 71
46. Uddin I, Ul Haque HM, Rakib A, Rahmat MRS (2016) Resource-bounded context-aware applications: A survey and early experiment. In: International conference on nature of computation and communication. Springer, pp 153–164
47. Ronszcka AF, Banaszewski RF, Linhares RR, Tacla CA, Stadzisz PC, Simão JM (2015) Notification-oriented and rete network inference: a comparative study. In: 2015 IEEE international conference on systems, man, and cybernetics (SMC). IEEE, pp 807–814
48. Armstrong D (2014) Memory efficient stream reasoning on resource-limited devices. PhD thesis, Trinity College
49. Liu G, Huang S, Zhang D, Du Y (2014) A rete rule reasoning algorithm based on the audit method ontology. Int J Hybrid Inf Technol 7:211–244
50. Nalepa GJ, Szymon B (2014) Rule-based solution for context-aware reasoning on mobile devices. Comput Sci Inf Syst 11(1):171–193
51. Pinder C, Jo V, Wicaksono A, Beale R, Hendley RJ (2016) If this, then habit: exploring context-aware implementation intentions on smartphones. In: Proceedings of the 18th international conference on human-computer interaction with mobile devices and services adjunct. ACM, pp 690–697
52. Uddin I, Rakib A (2017) A preference-based application framework for resource-bounded context-aware agents. In: International conference on mobile and wireless technology, vol 425. LNEE, Springer, pp 187–196
53. Ijaz U, Rakib A (2017) A resource-aware preference model for context-aware systems. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, vol 217. LNICST, Springer, pp 3–13
54. Bell M, Kelly JF (2017) LEGO® MINDSTORMS® EV3: the mayan adventure. Apress