Memory-Constrained Context-Aware Reasoning

Ijaz Uddin[1], Abdur Rakib[2], Mumtaz Ali[1], and Phan Cong Vinh[3]

[1] Department of Computer Science
City University of Science and Information Technology, Peshawar
ijazktk@gmail.com,mumtazali@cusit.edu.pk

[2] Department of Computer Science and Creative Technologies
The University of the West of England, Bristol, UK
Rakib.Abdur@uwe.ac.uk

[3] Nguyen Tat Thanh University, Ho Chi Minh City, Vietnam
pcvinh@ntt.edu.vn

Abstract. The context-aware computing paradigm introduces environments, known as smart spaces, which can unobtrusively and proactively assist their users. These systems are currently mostly implemented on mobile platforms considering various techniques, including ontology-driven multi-agent rule-based reasoning. Rule-based reasoning is a relatively simple model that can be adapted to different real-world problems. It can be developed considering a set of assertions, which collectively constitute the working memory, and a set of rules that specify how to act on the assertion set. However, the size of the working memory is crucial when developing context-aware systems in resource constrained devices such as smartphones and wearables. In this paper, we discuss rule-based context-aware systems and techniques for determining the required working memory size for a fixed set of rules.

Keywords: Context-Aware Systems · Rule-based Reasoning · Working Memory.

1 Introduction

Rule-based reasoning attempts to emulate the capabilities of human reasoning and problem solving. The technique models how a human expert analyses a particular scenario by applying rules to the facts so that a conclusion can be drawn [1]. The reasoning process in a forward chaining rule-based system starts with known facts and progresses by using inference rules to extract more data. The facts are represented in a working memory which is continually updated. In rule-based systems, the knowledge is represented as a set of rules. The rules serve as long-term memory, whereas the facts serve as short-term memory. [2, 3]. While humans perceive their surroundings through their senses of sight, hearing, smell, taste, and touch, computers employ the context-awareness technique to become aware of their surroundings. In sensor-rich systems, sensor data is collected from a variety of sources by a device with various sensors attached to it. It is often difficult to interpret this sensed raw data. As a result, in the literature

Semantic Web technologies have been used in the design and implementation of sensor-rich systems [4, 5]. In Semantic Web technology, ontological representation allows context representation in terms of concepts and roles, context sharing and semantic interoperability of heterogeneous systems [6]. A framework for constructing context-aware systems in resource-bounded devices has been proposed in [6], where a context-aware system is modelled as a distributed rule-based agents. In the proposed model, the working memory containing facts (contexts) is conceptually divided into two parts, such as static and dynamic parts. The dynamic part of the working memory stores the newly derived contexts while the static part holds the initial contexts. The initial facts in the static memory are vital to start running any system, and therefore can not be removed or overwritten. More details of the setting can be found in [6]. In resource constrained systems, as a system moves, all the derived contexts cannot be stored in the working memory. That is old contexts could be overwritten by the newly derived contexts. In [5], it was shown how we can formally model and verify resource requirements of a system of rule-based context-aware reasoning agents. In this paper, we discuss techniques for determining the required working memory size of a rule-based context-aware reasoning system with a fixed set of rules.

The rest of the paper is structured as follow. Section 2 reviews related literature. Section 3 discusses an approach to smart space system modelling. Section 4 describes multi-agent rule-based reasoning and the basic components of rule-based systems. Section 5 discusses the management of working memory. Finally, Section 6 concludes the paper and discusses the scope for future work.

2 Related Work

Users can access context-aware services in the mobile environment due to the advent of small-scale microelectronic sensors and recent improvements in smartphones and other modern mobile devices such as smartwatches and wearable devices. Furthermore, the social network plays an important role in this regard, as users provide information primarily about their preferences, likes, and dislikes. In a different context, it can also provide contextual information about users, their surroundings, and their behavioural activities [7]. In other words, these smart devices are now a valuable source of data for determining context and understanding user behavioural activities in various situations [8]. The SociaCircuit Platform [9], for example, tracks various social interactions among users and, as a result, causes a change in the user's preferences. The developers of [10] came up with the idea of identifying a social interaction between users using data mining tools. The sociometric badge [11] tracks different activities of employees throughout office hours, forecasts job satisfaction, and coworker interaction based on that data/patterns. Although recent work on monitoring user activities and relationships with other users has provided us with a wealth of data and relevant information to analyse and draw conclusions, the domain still lacks various features. The authors of [12], for example, developed an expert system that functions as an academic advisor. It's a monotonous system that takes six user inputs and provides recommendations based on those inputs. However, this system is unable to run an alternative set of rules since the rules are linked to the given interface. Other research [13–18] focused on developing a client-server architecture model, in which the smartphone acts as an interface for an application installed on it, with the server acting as a knowledge base. Another iPhone platform research article [19] employs the same client-server architecture for a safe emergency evacuation from a university campus (case scenario). However, the authors did not specify the set of rules that define expert knowledge.

There have already been attempts to bring advanced expert systems to the Android platform. Although Android is based on Java, it lacks some classes that are only available in desktop environments. A book for Android smart applications has been published by [20]. Throughout the book, the rule engines that can be used with the Android platform are thoroughly described. These rule engines, however, lack context awareness and resource friendliness, as well as the ability to apply preferences for dynamic context awareness. Furthermore, according to the author, these engines have several significant limitations. For example, the Jrule engine and Zilonis do not support OR operators; in Termware, rules must be written in code and are difficult to update later; and in Roolie, each rule must be coded in its own file, which is also a time-consuming task that is nearly impossible in larger systems. Some technical challenges were encountered while porting various other rule engines. Since Drools is a memory-intensive engine, Eclipse quickly runs out of memory while converting files to Dalvik format. Take requires a Java compiler at runtime, and JLisa fails to function on Android due to a stake overflow issue. Jess, on the other hand, is extremely expensive and not recommended because its significant licencing costs, it is not compatible with Android, and it consumes a significant amount of memory.

The majority of rule engines are based on the RETE algorithm, according to our review and analysis of the literature. RETE is a memory-intensive algorithm that loads all of the rules into memory before checking them one by one in practise to see if they should be fired [21]. Similarly, checking all of the rules for each instance takes a long time. To save memory, the work by [21] introduced preferences. Instead of putting all of the rules into memory at once, their technique just loads the rules that are in the preference set. Although the approach consumes less memory during loading due to preference sets, it does not remove the already loaded rules from memory over time. In [6], the size of the memory allocation is fixed. If the algorithm claims a random fixed size memory while accounting for the size of the preference sets, it may run into issues if the system has to invoke rules that are not currently in the preference set. In [21, 6], there is no systematic technique for removing rules if the memory limit is reached. The rules are remove from the memory at random, which is a problem because the rule(s) selected for removal could be a key one that will be needed again soon for the execution.

3 Smart Space System Modelling

Developing intelligent, autonomous, and adaptable systems that work in complex dynamic environments is a goal that has been around for decades. In recent years work on this area has been a more focus of discussion and research [22–26]. A smart space provides an interoperable heterogeneous environment within which users, devices and services communicate. Such an environment ultimately provides users with unobtrusive assistance based on contextual knowledge [27]. Interoperability is critical here, since smart spaces contain many different devices and software components. It is a system's ability to exchange information, so that information is correctly interpreted by the receiving device in the same way that the transmitting device intends. One way to achieve this is by using the Semantics Web technology [28]. While Semantic Web technologies can be used to achieve interoperability on mobile devices, it is important to take into account the resource limitations and unique characteristics of mobile and embedded devices.

Context awareness is a key aspect of smart spaces, and context modelling is a fundamental step in developing context-aware systems that operate in these environments. By context, we refer to any physical or conceptual 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 application, and reflects the relationship among themselves [29]. A smart space serves its users by sensing and interpreting the situation they are in, identifying their needs and delivering the necessary functionality according to the available resources. This process involves three major steps, namely 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 in smart spaces [6]. The context acquisition process involves acquiring context in raw format from a wide variety of sensors. It may also allow the user to manually provide the contextual information [30]. To model a context and adapt it to any domain, an ontology is requires which captures the generic concepts to a higher level. The context model needs to provide frameworks for expanding the specific context knowledge in a hierarchical way. The context ontology provides a shared vocabulary for representing knowledge about a domain and for describing specific situations in a domain. The goal of contextual reasoning is to deduce higher-level contexts from the sensed low-level contexts as well as to deduce new relevant knowledge for the use of applications and users from various context-data sources. This stage is therefore primarily responsible for interpreting the situation in a smart environment, and for deciding how its users can be assisted [31]. The OWL 2 Web Ontology Language is one possible knowledge representation language for ontologies. The W3C recommended OWL specification includes the definition of three variants of OWL, with different levels of expressiveness, namely OWL Lite, OWL DL and OWL Full. OWL DL and OWL Lite are based on Description Logics, for which sound and complete reasoners exits. The W3C also recommended three profiles OWL 2 EL, OWL 2 QL and OWL 2 RL, which are restricted sublanguages of OWL 2. For context modelling and reasoning, OWL 2 RL and SWRL [32] languages can be used. OWL 2 RL is suitable for the design and development of rule-based systems, which can be implemented using rule-based reasoning engines [5]. An OWL 2 RL ontology can be translated into a set of Horn clause rules based on [33]. Moreover, more complex rule-based concepts can be expressed using SWRL 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. A more detailed discussion of this approach can be found in [34].

4 Multi-Agent Rule-Based Reasoning Systems

Multi-agent systems (MASs) are based on the concept of a decentralised working group being able to deal with problems that are difficult to solve using the conventional centralised computing approach. Intelligent agents are used to work more effectively and in a versatile and interactive way to solve problems. Multi-agent systems are considered to be a promising approach to dealing with ubiquitous systems development because of their ability to adapt themselves to dynamic environments. Such features together with collaborative behaviour, autonomy, reactivity and pro-activity facilitate the modelling and reasoning about complex context-aware system behaviour [5].

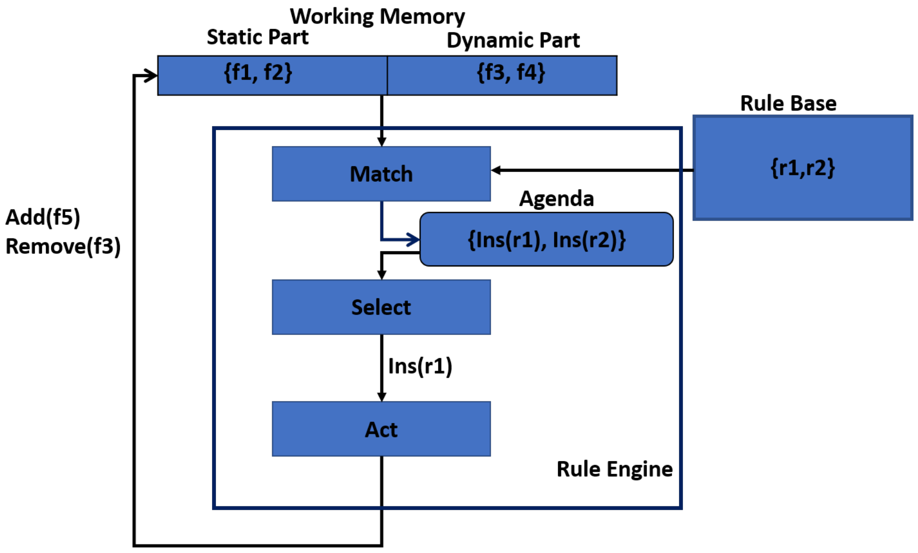

Fig. 1. Reasoning process of a rule-based agent

An agent is called a rule-based agent, if its behaviour and/or its knowledge is expressed by means of rules. A rule-based reasoning agent, as can be seen in Figure 1, has few components such as a rule-base, an inference engine and a working memory. A rule-base contains a collection of rules, where rules are IF(antecedent)—THEN(consequent) statements. The antecedent is a sufficient condition for the consequent and the consequent is a necessary condition for the antecedent. The antecedent part is matched with the elements in the working memory. If the antecedent of a rule is fully matched with the elements of the working memory then the rule is said to be eligible for firing and the corresponding rule instance(s) can be added to the agenda. The working memory of an agent consists of facts. The facts are initial contexts, derived contexts and/or communicated contexts received as messages from other agents in the system. The inference engine loops through three sequential phases: Match, Select and Act. In the Match phase, rules are matched with facts, if more than one rule is eligible to fire then a conflict arises. The strategy to decide which is the next rule to be fired is called conflict resolution. The next step is to resolve this conflict in the Select phase. Once a conflict is resolved than its time to fire the rule in the Act phase. Firing a rule instance can add a context to the working memory by possibly overriding an element from its dynamic part. Mostly the literature discusses the rule-based reasoning from a point of view where the resources (e.g., working memory size) are not considered. In our work, however, we consider memory to be a limited resource. In the following, we will concentrate our discussion on the rule-based reasoning algorithm and working memory management and updating.

5 Management of Working Memory

As previously discussed, the working memory is divided into two parts. Each part plays its own role. Part of the emphasis in this discussion is dynamic working memory. A newly created context may be added to the dynamic working memory by firing a rule instance, or a context may be added as a message received from another device or agent. In order to accommodate this new context, we must first review our current working mechanism and then suggest new techniques to substitute the current framework for efficient use of the working memory.

5.1 Working Memory Updating

The working memory acts as a holder for the currently available contexts and helps to perform context-aware reasoning. Where the emphasis is on resource limitations, memory is one of the primary resources we aim to save during the entire system design and implementation stages. The restriction on the size of the working memory is to ensure that it does not exceed the maximum number of contexts it can hold at any given time. However, contexts can be generated at every iteration, and preserving those contexts that are more crucial for execution is a critical task. In our implementation, the working memory is basically a fixed size container, which is divided into static memory and dynamic parts. The dynamic memory is bounded in size, where one unit of memory corresponds to the ability to store an arbitrary context. Only facts stored in the dynamic memory may get overwritten, and this happens if an agent's memory is full or a contradictory context arrives in the working memory (even if the memory is not full) [5]. Whenever a newly derived context arrives in the memory, it is compared with the existing contexts to see if any conflict arises. If so then the corresponding contradictory context will be replaced with the newly derived context, otherwise, the new context will be added to the working memory by overwriting an arbitrary context if the working memory is full. Since the dynamic memory is bounded in size, the system possibly can go through an infinite execution if a goal is not achievable and there is no way to forcefully stop the inference engine. For example, let's assume an instance of a rule r_1 generates a new context that can activate another rule r_2 and vice versa. If we have a single memory unit then the system will be in the infinite running state, unless otherwise controlled. To overcome this issue, we set the number of iterations equal to the number of rules. This will ensure that each rule is tested and, if no matches are found, the system will stop itself instead of abrupt behaviour. This also saves the resources of the host system. The Algorithm 1 describes the steps involved in executing the instance of the selected rule and updating the working memory.

5.2 Estimating Size of the Working Memory

Estimating the working memory size can help to minimise context loss, in particular the critical contexts. It can be achieved using the following techniques.

Distinct Working Memory(DWM) In the database analogy, the distinct returns all the results so that the duplicated values are only seen once instead of being replicated. Similarly, the working memory size can be set equal to the number of distinct consequences of the rules. Let R be the total number of rules in the rule base, R_c be the total number of consequences of the rules and R_{dc} be the total number of distinct consequences such that $R_{dc} \leq R_c$ and $R_{dc} > 0$. Then the required size of the working memory will be R_{dc}.

Average of the Preference Sets(APS) In this technique, the sets of preferences are taken into account. The primary idea is to customize user preferences so that resource-bounded context-aware applications can be personalised [30]. When preferences need to be implemented, the procedure is more complex than the prior method, but it saves space. This method considers the rules of several different groups of preferences. To begin, it calculates an average working memory size based on the overall average of rule base sizes. It then determines how many preference sets have WM requirements that are higher than the computed average. In the second stage, the technique will verify these preference sets (which have higher requirements than the average size) for distinct values in their consequent parts. If the number of distinct consequent parts in any of

```
Input: 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]
Output: Rule instance executed, consequent added to WM and
        corresponding action performed.
START
to_ fire from conflict resolution and $\mathbf{R}_{cons}$ is the consequent
if $R_g$ then
    if $\mathbf{R}_{cons}$ is a conflicting context then
        Overwrite the contradictory context with $\mathbf{R}_{cons}$
    end
    else if $|\mathbf{WM}| < MAX\_SIZE$ then
        Add $\mathbf{R}_{cons}$ to WM
    end
    else
        Overwrire an existing context with $\mathbf{R}_{cons}$
    end
    Goal Reached
    Execution Halts
end
else
    if $\mathbf{R}_{cons}$ is a conflicting context then
        Overwrite the contradictory context with $\mathbf{R}_{cons}$
        if $R_c$ then
            initiate communication module
        end
    end
    else if $|\mathbf{WM}| < MAX\_SIZE$ then
        Add $\mathbf{R}_{cons}$ to WM
        if $R_c$ then
            initiate communication module
        end
    end
    else
        Overwrite an existing context with $\mathbf{R}_{cons}$
        if $R_c$ then
            initiate communication module
        end
    end
end
END
```

Algorithm 1: Execution of a rule

these preference sets checked for WM is less than the average, the system will request WM size equal to the calculated average. However, if the value is higher than the average, the system will request a WM size that is equal to this value. To avoid context loss, the system will use the largest calculated value as the WM size in both cases. Let R be a rule base with $n = |R|$ number of rules and there are m preference sets $P_1, P_2, \ldots P_m$ with varied preference methods. Then the size of the working memory will be $(|P_1| + |P_2| + \ldots + |P_m|)/m$, where $|P_i|$ is the size of the preference set P_i for $1 \leq i \leq m$. However, the memory size may be larger than the calculated average if any preference set requires more memory as discussed above.

Smart Average of the Preference Sets (SAPS) Preference sets are another focus of the SAPS technique. This technique is similar to APS, but it has the added benefit of reducing the computation factor if certain conditions are met. Otherwise, it will function in the same way as APS does. In this approach, the standard deviation (SD) of the available preference sets is first calculated. A low standard deviation implies that the values are close to the set's mean, whereas a high standard deviation shows that the values are spread out over a larger range. Therefore, if the SD is small (less than a threshold value 2) then the number of rules in the preference sets will be close to the mean. In this case, among the available preference sets, the system will request the preference set that requires the most memory units. There will be no need to do any further calculations because the size will be enough for any other preference sets. However, if the SD is large (greater than or equal to a threshold value 2), it will function similarly to APS.

5.3 An Example

Let us consider an example system developed using a set of rules presented in Table 1 [30]. There are 18 different rules in this rule base. Some rules have the same consequence, and some of the rules have the same preference set. We have shown below how the aforementioned techniques are applied to Table 1.

Distinct Working Memory As previously explained, this technique returns all distinct results and does not display duplicated values; duplicated values are displayed just once. The following terms are used to apply the DWM approach to the table. Here, $|R| = 18$ (total number of rules), $|R_c| = 7$ (total number of consequences of the rules) and $|R_{dc}| = 4$ (total number of distinct consequences). In this case, $|R_c| \geq R_{dc}$ and the size of WM $= R_{dc}$. We get 12 distinct rules out of a total of 18 rules if we use the DWM technique to group the rules that have the same consequence and define the working memory on DWM bases. In this specific case, the DWM technique decreases the number of rules by almost 33.33%. This technique mainly depends on the number of rules having the same consequences. It will work more efficiently as the number of rules with the same consequence grows.

Table 1. Blood pressure and heart rate rules

Blood pressure category rules				
Category	m	Corresponding rule	F	CS
Low BP	1	Person(?p), hasSystolicBloodPressure(?p,?sbp), hasDiastolicBloodPressure(?p, ?dbp), lessThan(?sbp, '90), lessThan(?dbp,60) $\longrightarrow$ 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) $\longrightarrow$ 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)$\longrightarrow$ hasBPCategory(?p,PreHigh)	D	-
High	1	Person(?p), hasSystolicBloodPressure(?p,?sbp), hasDiastolicBloodPressure(?p, ?dbp), greaterThan(?sbp,140), greaterThan(?dbp,90)$\longrightarrow$ hasBPCategory(?p, HighBP)	D	-
Heart rate category rules				
Category	m	Corresponding rule	F	CS
Athlete	1	Person(?p), hasHeartRate(?p,?hrt), greaterThan(?hrt,48), lessThan(?hrt,55) $\longrightarrow$ hasHRCategory(?p, Athlete)	D	-
Excellent	1	Person(?p), hasHeartRate(?p,?hrt), greaterThan(?hrt,54), lessThan(?hrt,62) $\longrightarrow$ hasHRCategory(?p,Excellent)	D	-
Good	1	Person(?p), hasHeartRate(?p,?hrt), greaterThan(?hrt,61), lessThan(?hrt,66) $\longrightarrow$ hasHRCategory(?p,Good)	D	-
Above Average	1	Person(?p), hasHeartRate(?p,?hrt), greaterThan(?hrt,65), lessThan(?hrt,71) $\longrightarrow$ hasHRCategory(?p,AboveAverage)	D	-
Average	1	Person(?p), hasHeartRate(?p,?hrt), greaterThan(?hrt,70), lessThan(?hrt,75) $\longrightarrow$ hasHRCategory(?p,Average)	D	-
Below Average	1	Person(?p), hasHeartRate(?p,?hrt), greaterThan(?hrt,74),lessThan(?hrt,82) $\longrightarrow$ hasHRCategory(?p,BelowAverage)	D	-
Poor	1	Person(?p), hasHeartRate(?p,?hrt), greaterThan(?hrt,81) $\longrightarrow$ hasHRCategory(?p,Poor)	D	-
Some example rules to derive different situations				
Category	m	Corresponding rule	F	CS
Emergency	2	Patient(?p), hasBPCategory(?p,HighBP), hasHRCategory(?p,Poor) $\rightarrow$ hasSituation (?p,Emergency)	D	H
Emergency	2	Patient(?p), hasBPCategory(?p,PreHigh), hasHRCategory(?p,Poor) $\rightarrow$ hasSituation (?p,Emergency)	D	H
Emergency	2	Patient(?p),hasBPCategory(?p,Normal), hasHRCategory(?p,Poor) $\rightarrow$ hasSituation (?p,Emergency)	D	N
Emergency	2	Patient(?p),hasBPCategory(?p,LowBp), hasHRCategory(?p,Poor) $\rightarrow$hasSituation (?p,Emergency)	D	L
Non Emergency	1	Patient(?p),hasBPCategory(?p,Normal), hasHRCategory(?p,Average) $\rightarrow$ $\sim$hasSituation (?p,Emergency)	D	N
Non Emergency	1	Patient(?p),hasBPCategory(?p,Normal), hasHRCategory(?p,AboveAverage) $\rightarrow$ $\sim$hasSituation (?p,Emergency)	D	N
Non Emergency	1	Patient(?p),hasBPCategory(?p,Normal), hasHRCategory(?p,Good) $\rightarrow$ $\sim$hasSituation (?p,Emergency)	D	N

Average of the Preference Sets (APS) The APS approach is primarily concerned with preference sets. This technique takes into account the rules in various preference sets and calculates an average; the WM size is determined by the calculated average. The WM is the same size as the average. There can be four different preference sets based on the rule-base in Table 1, consisting of 11, 2, 1, and 4 rules, with an average $(11 + 2 + 1 + 4)/4 = 4.5$. However, the preference set $P1$ having $9(> 5)$ distinct consequences. Thus, the size of the working memory WM is 9. This method will reduce the amount of time it takes to calculate and check each rule for distinct consequences. In this example, instead of 18 rules, only 9 rules were checked, which is a 50 percent reduction. In other circumstances, depending on the number of rules in the preference sets with values higher than the average calculated, it can be reduced from 20 to 60 percent. In the general case, the required WM size for the DWM technique is 12, whereas it is for the APS technique is 9, which is roughly 25 percent less than the DWM technique. However, the results may vary from one example to another. It's also worth mentioning that this just applies to the size of the dynamic working memory.

Smart Average of the Preference Sets (SAPS) In the above example, $|P1| = 11, |P2| = 2, |P3| = 1$, and $|P4| = 4$. Therefore, the average is 4.5, while the standard deviation is 3.90. As a result, it will function similarly to APS, and the required WM size will be 9. However, for example, if we have a scenario where $|P1| = 5, |P2| = 6, |P3| = 7$, and $|P4| = 8$, the average and standard deviation would be 6.5 and 1.11, respectively. In this case, the WM size will be 8 since standard deviation is smaller than the threshold value and the memory requirement of the preference set with the most rules is 8.

6 Discussion and Future Work

In this paper we have addressed the issues with the working memory in resource constrained devices and proposed a possible solution as to how much dynamic memory size of an agent is required with a fixed set of rules. There can be more complex solutions but resources should be considered. In the future we intend to work on the reverse engineering of the preference sets. Although, in our proposed system, we have a well-explained mechanism for deriving a preference set from a set of rules. This set depends on a variety of choices by a user such as context-based, derived or live preferences. However, it is unavoidable to make a strategy that can reverse a set of rules when not required. The preference set may be able to remove rules which are not likely to fire in the future by its own. As a result, redundancy will be reduced, allowing the system's output to be maximised. Another area where further development could be made is with the rule generating strategy. Because the rules are created in software or with a tool and then processed in a typical manner, in our situation, we have several options for creating and changing rules, including a web-based interface, writing to a JSON file, and utilising the Onto-HCR tool [24]. We would also like to make changes to the generated rules for preferences and other factors. It would be more convenient to do devoted research in this regard in order to develop a standalone framework capable of automating all of the processes mentioned above.

References

1. A. Ligeza, *Logical Foundations for Rule-Based Systems*, vol. 11(2). Berlin, Heidelberg: Springer-Verlag, 2006.
2. J. C. Giarratano and G. Riley, “Expert systems, principles and programming, thomson course of technology,” *Boston, Australia*, 2005.
3. G. F. Luger, *Artificial Intelligence: Structures and Strategies for Complex Problem Solving–6th Edition*.
4. W. Tai, J. Keeney, and D. O’Sullivan, “Resource-constrained reasoning using a reasoner composition approach,” *Semantic Web*, vol. 6, pp. 35–59, 2015.
5. A. Rakib and H. M. U. Haque, “A logic for context-aware non-monotonic reasoning agents,” in *Mexican International Conference on Artificial Intelligence*, pp. 453–471, Springer, 2014.
6. A. Rakib and I. Uddin, “An efficient rule-based distributed reasoning framework for resource-bounded systems,” *Mobile Networks and Applications*, vol. 24, no. 1, pp. 82–99, 2019.
7. I. H. Sarker, “Mobile data science: Towards understanding data-driven intelligent mobile applications,” *arXiv preprint arXiv:1811.02491*, 2018.
8. I. H. Sarker, “Behavminer: Mining user behaviors from mobile phone data for personalized services,” in *2018 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, pp. 452–453, IEEE Computer Society, 2018.
9. I. Chronis, A. Madan, and A. Pentland, “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*, pp. 1–4, 2009.
10. J. J. Jung, “Contextualized mobile recommendation service based on interactive social network discovered from mobile users,” *Expert Systems with Applications*, vol. 36, no. 9, pp. 11950–11956, 2009.
11. D. O. Olguín, B. N. Waber, T. Kim, A. Mohan, K. Ara, and A. Pentland, “Sensible organizations: Technology and methodology for automatically measuring organizational behavior,” *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 39, no. 1, pp. 43–55, 2008.
12. W. M. Aly, K. A. Eskaf, and A. S. Selim, “Fuzzy mobile expert system for academic advising,” in *2017 IEEE 30th Canadian Conference on Electrical and Computer Engineering (CCECE)*, pp. 1–5, IEEE, 2017.
13. A. Ghasempour, “Optimized scalable decentralized hybrid advanced metering infrastructure for smart grid,” in *2015 IEEE International Conference on Smart Grid Communications (SmartGridComm)*, pp. 223–228, IEEE, 2015.
14. A. Ghasempour, “Optimum packet service and arrival rates in advanced metering infrastructure architecture of smart grid,” in *2016 IEEE Green Technologies Conference (GreenTech)*, pp. 1–5, IEEE, 2016.
15. A. Ghasempour, “Optimized advanced metering infrastructure architecture of smart grid based on total cost, energy, and delay,” in *2016 IEEE Power & Energy*

Society Innovative Smart Grid Technologies Conference (ISGT), pp. 1–6, IEEE, 2016.

16. A. Ghasempour, *Optimizing the advanced metering infrastructure architecture in smart grid.* Utah State University, 2016.
17. V. Sharma, F. Song, I. You, and M. Atiquzzaman, "Energy efficient device discovery for reliable communication in 5g-based iot and bsns using unmanned aerial vehicles," *Journal of Network and Computer Applications*, vol. 97, pp. 79–95, 2017.
18. V. Sharma, I. You, K. Andersson, F. Palmieri, M. H. Rehmani, and J. Lim, "Security, privacy and trust for smart mobile-internet of things (m-iot): A survey," *IEEE Access*, vol. 8, pp. 167123–167163, 2020.
19. M. F. Abulkhair and L. F. Ibrahim, "Using rule base system in mobile platform to build alert system for evacuation and guidance," *Int J Adv Comput Sci Appl*, vol. 7, no. 4, pp. 68–79, 2016.
20. C. Mukherjee, *Build Android-Based Smart Applications: Using Rules Engines, NLP and Automation Frameworks.* Apress, 2017.
21. I. Uddin, *A rule-based framework for developing context-aware systems for smart spaces.* PhD thesis, University of Nottingham, 2019.
22. M. Alirezaie, J. Renoux, U. Köckemann, A. Kristoffersson, L. Karlsson, E. Blomqvist, N. Tsiftes, T. Voigt, and A. Loutfi, "An ontology-based context-aware system for smart homes: E-care@home," *Sensors (Basel, Switzerland)*, vol. 17, 2017.
23. R. Abdur, "Smart space system interoperability," in *Proceedings of the 3rd International Workshop on (Meta)Modelling for Healthcare Systems, Bergen, Norway*, vol. 2336, pp. 16–23, CEUR Workshop Proceedings, 2018.
24. I. Uddin, A. Rakib, H. M. U. Haque, and P. C. Vinh, "Modeling and reasoning about preference-based context-aware agents over heterogeneous knowledge sources," *Mobile Networks and Applications*, vol. 23, pp. 13–26, 2018.
25. N. A. Streitz, D. Charitos, M. Kaptein, and M. Böhlen, "Grand challenges for ambient intelligence and implications for design contexts and smart societies," *J. Ambient Intell. Smart Environ.*, vol. 11, pp. 87–107, 2019.
26. P. N. Mahalle and P. S. Dhotre, *Context-Aware Pervasive Systems and Applications*, vol. 169 of *Intelligent Systems Reference Library.* Springer, 2020.
27. D. Cook and S. Das, *Smart Environments: Technology, Protocols and Applications (Wiley Series on Parallel and Distributed Computing).* USA: Wiley-Interscience, 2004.
28. N. Noy, D. McGuinness, and P. J. Hayes, "Semantic integration & interoperability using RDF and OWL." W3C Editor's Draft 3 November, 2005.
29. A. K. Dey, "Understanding and using context," *Personal Ubiquitous Comput.*, vol. 5, p. 4–7, Jan. 2001.
30. I. Uddin and A. Rakib, "A preference-based application framework for resource-bounded context-aware agents," in *International conference on mobile and wireless technology*, pp. 187–196, Springer, 2017.
31. X. H. Wang, D. Q. Zhang, T. Gu, and H. K. Pung, "Ontology based context modeling and reasoning using owl," in *IEEE Annual Conference on Pervasive Computing and Communications Workshops*, pp. 18–22, 2004.
32. I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean, "SWRL: A Semantic Web rule language combining OWL and RuleML. Acknowledged W3C submission, standards proposal research report: Version 0.6," April 2004.

33. B. Grosof, I. Horrocks, R. Volz, and S. Decker, “Description logic programs: Combining logic programs with description logics,” in *The Twelfth International World Wide Web Conference, Budapest*, pp. 48–57, ACM, 2003.
34. A. Rakib, H. M. U. Haque, and R. U. Faruqui, “A temporal description logic for resource-bounded rule-based context-aware agents,” in *Context-Aware Systems and Applications*, pp. 3–14, Springer, 2014.