# Handling Working Memory Knowledge Through a Consultant-Level Resource Management Strategy

Rafael Sousa Silva
rsousasilva@mines.edu
Colorado School of Mines
Golden, Colorado, USA

Tom Williams
twilliams@mines.edu
Colorado School of Mines
Golden, Colorado, USA

## ABSTRACT

Working memory is an important component of cognition that influences key cognitive processes, such as language. As such, working memory should play a key role in cognitive models for language-capable robots. The ways in which working memory buffers are organized within a robot's architecture can inform processes such as Referring Expression Generation. Thus, it is important to understand how information and resources within working memory may be organized to lead to human-like robotic language. Previous work on the DIARC cognitive architecture described an entity-level, feature-based working memory framework in which each known entity had its own dedicated working memory buffer. This paper expands on that framework and proposes a new resource management strategy in which sets of entities that belong to the same type share a single working memory buffer. We end the paper with a brief discussion of how this novel strategy compares to the previously implemented entity-level strategy.

## CCS CONCEPTS

• **Computing methodologies** → **Cognitive robotics**; • **Computer systems organization** → **Distributed architectures**.

## KEYWORDS

robot cognitive architectures, working memory, referring expression generation

**ACM Reference Format:**
Rafael Sousa Silva and Tom Williams. 2024. Handling Working Memory Knowledge Through a Consultant-Level Resource Management Strategy. In *Companion of the 2024 ACM/IEEE International Conference on Human-Robot Interaction (HRI '24 Companion), March 11–14, 2024, Boulder, CO, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3610978.3640634

## 1 INTRODUCTION AND MOTIVATION

Cognitive models for language-capable robots may borrow inspiration from human cognition in order to promote more natural, human-like dialogue. Working memory is a pervasive component of human cognition [3], influencing key cognitive processes, including language [cf. 1, 2, 6, 9–11, 14, 18]. Given its importance to cognition, working memory is commonly featured in robot cognitive architectures and plays a major role on the intrinsic decision-making processes that guide an agent's behavior. In this paper, we directly build upon recent work on one of these architectures, DIARC [15], which has historically implemented cognitively-inspired models of working memory to facilitate the process of Referring Expression Generation [16, 22]. For example, activated working memory contents may be prioritized by a robot speaker when generating referring expressions to facilitate hearers' understanding.

One of the main characteristics of working memory is its limited capacity, which is usually estimated to range from 4 to 9 items [5, 13]. This low storage capacity and the volatility of working memory suggest that its contents are subject to constant updates. Therefore, it is important to understand how items leave working memory in order to give space to more salient items. As such, DIARC's working memory models are centered around the mechanism of *forgetting*, which performs the maintenance of working memory through two cognitively-inspired strategies. The first strategy is *decay* [4, 8], in which items within working memory are removed from buffers over time. The second strategy is *interference* [7, 19], in which older working memory items are replaced with newer entries if not rehearsed or reiterated.

Finally, instead of storing salient entities, DIARC's working memory buffers store the *features* of such entities to prioritize the quality over the quantity of working memory representations [12]. This *feature-based* perspective introduces different structural options for how working memory buffers are distributed across the architecture. In previous work, Williams et al. [22] described an *entity-level*, feature-based framework in which each entity known by the architecture has a dedicated working memory buffer that will hold salient features. In this work, we formally define a *consultant-level* framework (see Section 2.1) that instead assigns working memory buffers to different *entity types*. These buffers are shared by all entities that belong to their corresponding type. For example, if a robot has information about three humans in a knowledge base, these three entities would share the same "people" working memory buffer. More details about DIARC's knowledge bases and working memory are provided in the next section.

After formalizing and introducing our new strategy for working memory buffer organization (Section 3), we perform a discussion of how our model compares to the entity-level framework described in previous work (Section 4). Finally, we state our concluding remarks and a few directions for future work (Section 5).

## 2 THE DIARC ARCHITECTURE

In this section, we briefly introduce how DIARC's knowledge bases and working memory component work. This information is necessary to understand how our new resource management strategy functions and informs robot decision-making.



*HRI '24 Companion, March 11–14, 2024, Boulder, CO, USA*

ACM ISBN 979-8-4007-0323-2/24/03.
https://doi.org/10.1145/3610978.3640634

## 2.1 DIARC's Knowledge Bases and Consultants

In DIARC, world knowledge is not handled by a single knowledge base, but is rather split across different Distributed Heterogeneous Knowledge Bases (DHKBs) [23], allowing each different type of knowledge to be handled in the way that suits it best. This decentralized knowledge organization creates the need for a special type of architectural component called *consultants* [20]. Consultants filter the domain-specific information from these DHKBs into first-order logic statements that can be understood and used by any other component within the architecture. Thus, when other architectural components require certain information from a DHKB, a consultant will only share the necessary domain-independent information in first-order logic format with them. For example, in a task where a robot needs to formulate a description for an object that it knows as `object_1`, the consultant responsible for storing the features of `object_1` may be queried by other DIARC components and return a list of first-order logic statements that can be used in a description (e.g., `object_1 = [blue(X), mug(X)]`).

## 2.2 DIARC's Working Memory Manager

The Working Memory (WM) Manager is a component responsible for implementing forgetting and maintaining working memory buffers across the architecture. Previous work [22] has described an entity-level, feature-based implementation for the WM Manager. In this work, we expand this concept with a formal definition of the WM Manager component that includes a *consultant-level* resource management strategy.

The WM Manager can be described as $W = (S, F, C)$, where:

- $S = \{s : s \in \{e, t\}\}$ represents the active working memory resource management strategy within the architecture. When $S = \{e\}$, an entity-level strategy is active within the architecture. Similarly, when $S = \{t\}$*, a consultant-level strategy is active within the architecture.
- $F = [\delta, \alpha]$ represents the list of parameter values for Decay and Interference that guide how Working Memory functions. The parameter $\delta$ represents the decay factor associated with Decay and the parameter $\alpha$ represents the maximum Working Memory buffer size associated with Interference;
- $C = \{c_1, \ldots, c_n\}$ represents the set of active consultants within the architecture. Each consultant $c \in C$ is described as $c = (c_{domain}, c_{constraints})$, where $c_{domain}$ represents the domain of entities within $c$ (e.g., "objects", "locations", "people") and $c_{constraints}$ represents the set of first-order logic constraints that $c$ can handle (e.g., `metallic(X)`, `green(X)`, `on(X,Y)`) [20, 21].

Our formal definition of the WM Manager introduces two new features to this component. First, the parameter $S$, which specifies the active resource management strategy within the architecture. Second, in the initial framework the interference parameter $\alpha$ specified the size limit for each entity's working memory buffer. With the introduction of the consultant-level strategy, $\alpha$ now specifies the upper bound for the number of total working memory properties per consultant. As such, when $S = \{t\}$, each consultant $c \in C$ will have a dedicated working memory buffer $B_c \subseteq c_{constraints}$ such that $|B_c| \leq \alpha$ for managing constraints that span all entities in $c_{domain}$. On the other hand, when $S = \{e\}$, each entity within a consultant will have a dedicated working memory buffer $B_e \subseteq c_{constraints}$ such that $|B_e| \leq \lceil \frac{\alpha}{|c_{domain}|} \rceil$ for managing salient constraints.

*We use $t$ instead of $c$ to avoid confusions with previously established notation.

# 3 CONSULTANT-LEVEL FRAMEWORK

In this section we describe how features are encoded and removed from Working Memory buffers at a consultant-level.

## 3.1 Encoding

In a consultant-level working memory model, features are added to a consultant's working memory buffer when they are used to describe an entity within that consultant's domain. The pseudocode for this process is listed in Algorithm 1.

**Algorithm 1** Consultant-level encoding

```
1: procedure ENCODE(R, C)
   R: Referring expression.
   C: Set of active consultants.
2:     for all c ∈ C do
3:         for all r ∈ R do
4:             if r ∈ c_constraints and r ∈ B_c then
5:                 remove(r, B_c)
6:                 push(r, B_c)
7:             else if r ∈ c_constraints and r ∉ B_c then
8:                 push(r, B_c)
```

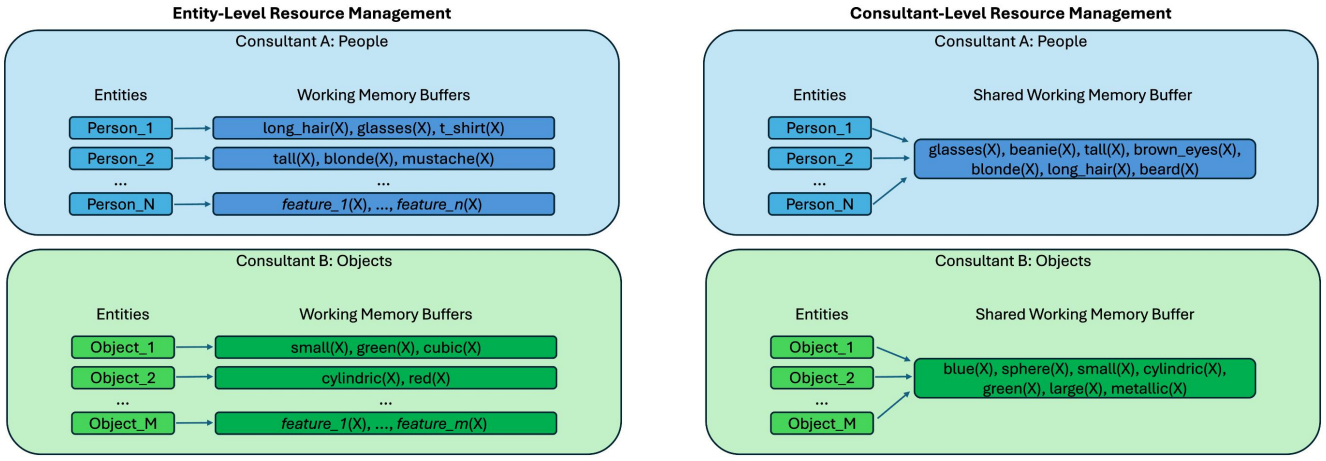
When a referring expression $R \subseteq \bigcup_{c \in C} c_{constraints}$ is generated or processed by the robot architecture (line 1), for each consultant $c \in C$ (line 2), each constraint $r \in R$ (line 3) is added to $c$'s working memory buffer $B_c$ if it is part of the list of constraints that can be handled by $c$ (i.e., $r \in c_{constraints}$). If $r$ is already in the working memory buffer, then its recency is updated (lines 4–6). Otherwise, it is added to the working memory buffer (lines 7–8).

## 3.2 Removal

The removal of constraints from working memory buffers is governed by the forgetting parameters specified in $F = [\delta, \alpha]$. When decay is in use (i.e., $\delta \neq \infty$), the least recent property from a buffer is removed every $\delta$ seconds. When interference is in use (i.e., $\alpha \neq \infty$), the least recent constraints are removed from a buffer that exceeds the maximum capacity $\alpha$. Algorithm 2 provides the pseudocode for this process.

**Algorithm 2** Consultant-level removal

```
1: procedure REMOVE(F, C)
   F = [δ, α]: List of forgetting parameters.
   C: Set of active consultants.
2:     for all c ∈ C do
3:         if δ seconds have passed and |B_c| > 0 then
4:             pop(B_c)
5:         while |B_c| > α do
6:             pop(B_c)
```




**(a) In an entity-level strategy, each entity known by the architecture will have a dedicated working memory buffer. Properties in working memory buffers are used to describe only their corresponding entity.**

**(b) In a consultant-level strategy, all entities of the same type share one single working memory buffer. Properties in a consultant's buffer may apply to different entities.**

**Figure 1: Architectural overview of the entity-level and consultant-level strategies with two example consultants.**

For each consultant $c \in C$ (line 2), if $\delta$ seconds have passed and there exist properties in $B_c$, then the least recent property is removed from $B_c$ (lines 3–4). In addition, if the size of $B_c$ exceeds the maximum working memory buffer size $\alpha$, the least recent properties are removed until buffer capacity matches $\alpha$ (lines 5–6).

## 4 DISCUSSION

We will now briefly discuss how the consultant-level strategy compares to the previously proposed entity-level strategy in terms of their spatiotemporal needs and cognitive plausibility. While the consultant-level strategy seems to carry a better computational efficiency and a structure that is closer to the psychological definition of working memory, there are a few downsides of using it instead of the entity-level strategy.

### 4.1 Spatiotemporal Needs

The spatial and temporal needs of the *consultant-level* working memory resource management strategy may be significantly lower than those of the *entity-level* strategy. This is observed because in the latter strategy one working memory buffer will be assigned to each entity known by the architecture (see Figure 1a). As such, in situations where the number of entities known by a consultant is considerably high, the processes of encoding and removal may take a while longer to be completed. Furthermore, a consultant-level strategy reduces the number of required working memory buffers according to the entity types that are known by the architecture (see Figure 1b). Depending on the level of abstraction between different entity types, the total working memory capacity for the entire architecture may be known in advance. For example, if an architectural model adopts the taxonomy for core knowledge proposed by Spelke and Kinzler [17], there will be at most five active consultants within it (i.e., people, locations, objects, actions, numbers). Thus, the total working memory capacity for the architecture will have an upper bound of $5\alpha$ elements.

### 4.2 Cognitive Plausibility

A consultant-level resource management strategy is closer to cognitive psychology models of human working memory in terms of storage than the entity-level strategy. Specifically, the entity-level resource management strategy is more likely to bring overall storage capacity above the speculated human limit even with low buffer sizes, and requires storage of some information about every known entity in working memory, which is simply not cognitively plausible. While a consultant-level strategy is not dependant on the number of known entities and significantly reduces the total working memory storage that is required, it still may overshoot the stipulated working memory capacity for humans, depending on the number of consultants, and on whether traditional human working memory limitations are assumed to hold at the entity or feature level. For example, an architectural model with five consultants will already surpass the stipulated human capacity if $\alpha = 2$. With this in mind, future work may investigate the feasibility of a global resource management strategy for working memory, in which only one working memory buffer is assigned to *all entity types*.

### 4.3 Disadvantages of Using the Consultant-Level Strategy

Using a consultant-level strategy in situations where the total number of known entities by the architecture is not high may not be ideal, as it may make referring expression generation less consistent. Since working memory buffers are shared by entities of the same type, the way in which a robot chooses to formulate descriptions of objects will not be as specific as that of an entity-level strategy. This is due to the fact that in an entity-level strategy, the buffer for each entity will hold the most recent constraints that were used to

describe it and thus will promote the repetition of the same properties in subsequent robot-formulated utterances. For example, if an object is described to a robot as "the blue mug," the properties `blue(X)` and `mug(X)` will remain in that object's working memory buffer independently of how many other objects are referenced afterwards, and the robot will reuse `blue(X)` and `mug(X)` to describe the object when necessary. In contrast, a consultant-level strategy will update the buffer for each entity type whenever an entity of that type is mentioned, and thus may not guarantee a consistent referring expression generation process, as multiple entities may share the same constraints. For instance, if an object is described to a robot as "the blue mug," `blue(X)` and `mug(X)` may be overwritten by the features of other objects that are referenced after the blue mug. This limitation is important as a less consistent process of referring expression generation may hurt human interlocutors' ability to perform reference resolution, i.e., understand which entity the robot is referring to, and may decrease the quality of interaction.

## 5 CONCLUSION

In this work, we introduced a consultant-level resource management strategy for the working memory models implemented in the DIARC cognitive architecture. We formally proposed a definition for the Working Memory Manager component that adds the need to specify which resource management strategy is guiding working memory. Furthermore, we performed a high-level discussion of the consultant-level strategy in terms of spatiotemporal needs and cognitive plausibility compared to the entity-level strategy. We observed that the consultant-level strategy likely promotes better computational efficiency both in terms of space and time needed to run the encoding and removal algorithms. In addition, the consultant-level strategy has a higher cognitive plausibility than the entity-level strategy, as less working memory buffers need to be active simultaneously.

The introduction of this model opens opportunities for subsequent work to investigate how a consultant-level resource management strategy impacts important natural language processes within DIARC. For example, future work may assess how the referring expressions of an agent that is using this strategy to guide its working memory are perceived by human interlocutors. An experiment can be designed to compare the utterances generated by an entity-level strategy, a consultant-level strategy, and a baseline framework for Referring Expression Generation that does not use working memory. The experimental results will help identify the best structural choices for robotic working memory models targeted at enabling better language-based interactions with humans. Finally, future work may also consider the implementation of a global resource management strategy that limits working memory information to a single architecture-wide buffer, an organization that will be more cognitively plausible than the currently existing strategies and that may be more appropriate for use in certain social contexts and interactions.

## ACKNOWLEDGMENTS

This work was funded in part by NSF CAREER grant IIS-2044865.

## REFERENCES

[1] Alan Baddeley. 2010. Working memory. *Current biology* 20, 4 (2010), R136–R140.
[2] Alan D Baddeley, Susan E Gathercole, and Costanza Papagno. 1998. The phonological loop as a language learning device. *Exploring Working Memory* (1998), 164–198.
[3] Paul Baxter and Tony Belpaeme. 2014. Pervasive memory: The future of long-term social hri lies in the past. In *Third international symposium on new frontiers in human-robot interaction at AISB*.
[4] John Brown. 1958. Some tests of the decay theory of immediate memory. *Quarterly journal of experimental psychology* 10, 1 (1958), 12–21.
[5] Nelson Cowan. 2001. The magical number 4 in short-term memory: A reconsideration of mental storage capacity. *Behavioral and brain sciences* 24, 1 (2001), 87–114.
[6] Nadiia Denhovska, Ludovica Serratrice, and John Payne. 2016. Acquisition of second language grammar under incidental learning conditions: The role of frequency and working memory. *Language Learning* 66, 1 (2016), 159–190.
[7] Michaela T Dewar, Nelson Cowan, and Sergio Della Sala. 2007. Forgetting due to retroactive interference: A fusion of Müller and Pilzecker's (1900) early insights into everyday forgetting and recent research on anterograde amnesia. *Cortex* 43, 5 (2007), 616–634.
[8] Hermann Ebbinghaus. 1885. Memory: A contribution to experimental psychology, trans. *HA Ruger & CE Bussenius. Teachers College.[rWvH]* (1885).
[9] Jeanette K Gundel, Nancy Hedberg, and Ron Zacharski. 1993. Cognitive status and the form of referring expressions in discourse. *Language* (1993), 274–307.
[10] Graeme S Halford, William H Wilson, and Steven Phillips. 1998. Processing capacity defined by relational complexity: Implications for comparative, developmental, and cognitive psychology. *Behavioral and brain sciences* 21, 6 (1998), 803–831.
[11] Patrick C Kyllonen and Raymond E Christal. 1990. Reasoning ability is (little more than) working-memory capacity?! *Intelligence* 14, 4 (1990), 389–433.
[12] Wei Ji Ma, Masud Husain, and Paul M Bays. 2014. Changing concepts of working memory. *Nature neuroscience* 17, 3 (2014), 347–356.
[13] George A Miller. 1956. The magical number seven, plus or minus two: Some limits on our capacity for processing information. *Psychological review* 63, 2 (1956), 81.
[14] Jerker Rönnberg, Mary Rudner, Thomas Lunner, Adriana A Zekveld, et al. 2010. When cognition kicks in: Working memory and speech understanding in noise. *Noise and Health* 12, 49 (2010), 263.
[15] Matthias Scheutz, Thomas Williams, Evan Krause, Bradley Oosterveld, Vasanth Sarathy, and Tyler Frasca. 2019. An overview of the distributed integrated cognition affect and reflection diarc architecture. *Cognitive architectures* (2019), 165–193.
[16] Rafael Sousa Silva, Michelle Lieng, and Tom Williams. 2023. Forget About It: Entity-Level Working Memory Models for Referring Expression Generation in Robot Cognitive Architectures. In *Proceedings of the Annual Meeting of the Cognitive Science Society*, Vol. 45.
[17] Elizabeth S Spelke and Katherine D Kinzler. 2007. Core knowledge. *Developmental science* 10, 1 (2007), 89–96.
[18] Heinz-Martin Süß, Klaus Oberauer, Werner W Wittmann, Oliver Wilhelm, and Ralf Schulze. 2002. Working-memory capacity explains reasoning ability—and a little bit more. *Intelligence* 30, 3 (2002), 261–288.
[19] Nancy C Waugh and Donald A Norman. 1965. Primary memory. *Psychological review* 72, 2 (1965), 89.
[20] Tom Williams. 2017. A Consultant Framework for Natural Language Processing in Integrated Robot Architectures. *IEEE Intell. Informatics Bull.* 18, 1 (2017), 10–14.
[21] Tom Williams. 2017. Situated natural language interaction in uncertain and open worlds. *AI Matters* 3, 2 (Jul 2017), 20–21. https://doi.org/10.1145/3098888.3098896
[22] Tom Williams, Torin Johnson, Will Culpepper, and Kellyn Larson. 2020. Toward forgetting-sensitive referring expression generationfor integrated robot architectures. *arXiv preprint arXiv:2007.08672* (2020).
[23] Tom Williams and Matthias Scheutz. 2016. A framework for resolving open-world referential expressions in distributed heterogeneous knowledge bases. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 30.