Pontifícia Universidade Católica do Paraná
Programa de Pós-Graduação em Informática

Cognitive Memory-based Service Directory for Performing Dynamic Pervasive Service Composition

Rafael Salazar Salazar

Supervisor

Prof. Dr. Edson Emílio Scalabrin

Supervisor

Prof. Dr. Félix Francisco Ramos Corchado

Centro de Investigacion y de Estudios Avanzados del Instituto Politecnico Nacional

Curitiba
2024

Pontifícia Universidade Católica do Paraná
Programa de Pós-Graduação em Informática

Cognitive Memory-based Service Directory for Performing Dynamic Pervasive Service Composition

Rafael Salazar Salazar

Thesis presented to the *Programa de Pós-Graduação em Informática* as a partial requirement for the degree of Doctor in Informatics.

Major Field: Computer Science

Supervisor: Prof. Dr. Edson Emílio Scalabrin

Supervisor: Prof. Dr. Félix Francisco Ramos Corchado

Curitiba
2024

Dados da Catalogação na Publicação
Pontifícia Universidade Católica do Paraná
Sistema Integrado de Bibliotecas – SIBI/PUCPR
Biblioteca Central
Luci Eduarda Wielganczuk – CRB 9/1118

Salazar, Rafael Salazar
S161c Cognitive memory-based service directory for performing dynamic pervasive
2024 service composition / Rafael Salazar Salazar ; supervisors: Edson Emílio Scalabrin, Félix Francisco Ramos Corchado. – 2024.
88 f. : il. ; 30 cm

Tese (doutorado) – Pontifícia Universidade Católica do Paraná, Curitiba, 2024
Bibliografia: f. 82-88

1. Informática. 2. Arquitetura cognitiva. 3. Composição pervasiva.
4. Composição baseada em experiência. 5. Seleção de serviços. I. Scalabrin, Edson Emílio. II. Corchado, Félix Francisco Ramos. III. Pontifícia Universidade Católica do Paraná. Programa de Pós-Graduação em Informática. IV. Título.

CDD. 21. ed. – 004

Pontifícia Universidade Católica do Paraná
Escola Politécnica
Programa de Pós-Graduação em Informática

Curitiba, 07 de novembro de 2024.

92-2024

DECLARAÇÃO

Declaro para os devidos fins, que **RAFAEL SALAZAR SALAZAR** defendeu a tese de Doutorado intitulada "**Cognitive Memory-based Service Directory for Performing Dynamic Pervasive Service Composition"**, na área de concentração Ciência da Computação no dia 21 de junho de 2024, no qual foi aprovado.

Declaro ainda, que foram feitas todas as alterações solicitadas pela Banca Examinadora, cumprindo todas as normas de formatação definidas pelo Programa.

Por ser verdade firmo a presente declaração.

gov.br Documento assinado digitalmente
EMERSON CABRERA PARAISO
Data: 18/11/2024 22:09:14-0300
Verifique em https://validar.iti.gov.br

Prof. Dr. Emerson Cabrera Paraiso
Coordenador do Programa de Pós-Graduação em Informática

Abstract

Progress in the capabilities of computing devices is relentless, leading to the rise of research fields looking to exploit devices' functionalities to improve our quality of life. Pervasive service composition studies the way to create useful services satisfying actual problems. However, service composition still has many challenges because the dynamicity of the environments, where resource availability is uncertain. Thus, two key problems to solve are: first, to know the availability of resources to compose a service, and second, the use of experience to compose a service to solve a specific problem efficiently. In this work, we propose a Biologically Inspired Cognitive Architecture for a service directory and a directory of past compositions for specific problems. Our proposal is inspired in the way the human brain keeps track of available resources it disposes to solve a problem and keeps experience of past solution for specific problems.

Key-words: Pervasive Composition, Bio-Inspired Cognitive Architecture, Experience-Based Composition, Service Selection

Resumo

O progresso nas capacidades dos dispositivos computacionais é incessante, levando ao surgimento de campos de pesquisa que buscam explorar as funcionalidades desses dispositivos para melhorar nossa qualidade de vida. A composição de serviços pervasivos estuda a forma de criar serviços úteis que resolvam problemas reais. No entanto, a composição de serviços ainda enfrenta muitos desafios devido à dinamicidade dos ambientes, onde a disponibilidade de recursos é incerta. Assim, dois problemas-chave a serem resolvidos são: primeiro, conhecer a disponibilidade de recursos para compor um serviço e, segundo, o uso da experiência para compor um serviço de forma eficiente para resolver um problema específico. Neste trabalho, propõe-se uma Arquitetura Cognitiva Inspirada Biologicamente para um diretório de serviços e um diretório de composições passadas para problemas específicos. Esta proposta é inspirada na maneira como o cérebro humano monitora os recursos disponíveis para resolver um problema e retém a experiência de soluções passadas para problemas específicos.

Key-words: Composição Pervasiva, Arquitetura Cognitiva Bio-Inspirada, Composição Baseada em Experiência, Seleção de Serviços

Acknowledgements

I would like to thank my supervisor, Dr. Félix Ramos Corchado, for his guidance his feedback, for always sticking by me, for saying the things that needed to be said in order to get things done and to keep me focused. Without his guidance, support and perseverance, this work would not have been possible.

I am very thankful to Dr. Edson Emílio Scalabrin for his guidance and help with my work, for receiving me and helping me adapt during my academic exchange at the PUCPR in Curitiba, Brazil, for helping me out whenever I required it, up to the very last day of my exchange, for being always so kind. Obrigado por tudo e até breve!

I would also like to thank all my peers and friends at both CINVESTAV Guadalajara and PUCPR: people like Axel, Carlos, Christian, César, Jorge, Pedro, Alfonso, Gustavo, Alan, Adrián Parra, Adrián Ulises, Diana, Francisco, Román, Luis Martín, Alison, David, Miguel, Rafael, Herai, Bruno, Sheila, An- gelo, all of them left a mark on me and helped me through this phase of my life, and I am eternally grateful for it.

I am also thankful to the academic staff at CINVESTAV Guadalajara for all the help they offered me through this time: Nora, Rogelio, Juan Pablo, Aracely, Mony was always there when I needed them, always ready to help.

I am thankful to the Consejo Nacional de Humanidades, Ciencias y Tecnologías (CONAHCYT) for granting me a scholarship, which allowed me yo accomplish all this. I wouldn't be here without their help.

To CINVESTAV Guadalajara and the PUCPR, for giving me the chance to attend their doctorate programs.

Estoy inmensamente agradecido con mi familia: mi mamá, mi papá, mi hermana, mi tía María Luisa, mi madrina Lupita, mis tíos y primos, por todo el apoyo que me han brindado a través de mi vida, ya que sin ellos no sería quien soy el día de hoy. Este logro es tanto de ellos como mío.

And last but not least, to God, for always protecting me, for guiding me through my life, and for gracing me with all the opportunities and people, which turned me into who I am, allowing this to happen. All of the above is but a fraction of all the blessings He has bestowed upon me.

Contents

List of Figures

List of Tables

List of Algorithms

List of Codes

1

Introduction

1.1 Problem Definition

Breakthroughs in technologies such as wireless communications, battery technologies, miniaturization of semiconductors, among others, have been key for the development and deployment of ubiquitous and mobile computing. Technologies from these research areas allow devices to interact among themselves, opening the way for them to cooperate and coordinate in order to combine their individual capabilities resulting in a complex one that can tackle sophisticated user needs. This kind of service composition is known as pervasive service composition (ZHOU et al., 2009). Pervasive service composition puts different challenges on the table, among them: selecting the devices with suitable capabilities for a specific task, ensuring the required devices availability when needed, handling communication between heterogeneous devices, dealing with missing devices and communication lines, managing devices with limited energy, to know the way to achieve a solution, assessing the fitness of a produced solution, among others.

Many of the aforementioned challenges, which are important for fulfilling a specific task, are directly related to available device directories, which hold the information of available devices (services) for their proper use. In dynamic environments, where devices come in and go out, devices can become unavailable for different reasons, which requires that these directories need to be kept updated with the information of the available services reflecting the needed information

for their use when its capabilities are required. There are different challenges to solve in order to have this directory updated, such as: how this directory keeps the information updated, which actions to take when an available service leaves the environment, how to manage the security of the information and its processing, how often does the directory needs to inspect if a service is still alive, what are the policies and protocols to follow when a new service arrives or another leaves the environment that is, it needs to be registered or deleted from the service's directory. Even more, in the literature we can find many proposals dealing with the aforementioned challenges and some other more (CERVANTES et al., 2018; SOMMER; MAHÉO; BAKLOUTI, 2020; LIU; CAO; WANG, 2017; OSTOS et al., 2015; ZHOU et al., 2018) These proposals make clear the importance of the directory in their solutions (WALDO, 1999a; VINOSKI, 1997; SABBOUH et al., 2001; DOULKERIDIS; VALAVANIS; VAZIRGIANNIS, 2003; RAYCHOUDHURY et al., 2011). However, these proposals contributions are addressed to specific kind of situations. Even more, current research in fields such as Ambient Intelligence (AmI), Smart Cities, Smart Environments and Internet of Things (IoT) are very active in proposing solutions to issues mentioned above (ALSARYRAH; MASHAL; CHUNG, 2019; KHANOUCHE et al., 2019; URBIETA et al., 2017), given that this problem is very relevant to their whole research.

Moreover, we have observed that the human being deals with similar challenges when facing similar tasks to the previously described ones in similar dynamic environments, and what is most important, it tackles such situations not in an optimal, but in a sufficiently satisfactory and useful way. Thus, our main hypotheses is that using neuroscientific evidence about how human brain manages previous experiences for solving similar to our concern problems, and how it manages the resources needed to solve these kind of problems we will be able to deal with challenges described above. Also we consider how our proposal interacts with other components of the whole system. That is, how the configuration of the system is achieved.

In order to achieve such configurations, we must delve into how resource (service) registries and plans (compositions) are stored, retrieved and managed in the memory, and which cognitive modules participate in those processes, with the goal of defining the components that would constitute a memory-based service directory. In this work, we propose a service directory based on both the dynamic memory theory and neuroscientific evidence, with the

goal of developing a directory structure capable of managing service registries according to different factors such as messages coming from service providers or other service modules, as well as service expiration mechanisms, which would allow us to better cope with environment dynamism by allowing the service directory to dispose of registries that may have become unavailable over time and whose providers didn´t have the chance to deregister them, enabling the directory to have a bigger consistency between services registered in it and services available in the environment. To validate it, we test an implementation of our proposal in a case study to evaluate the availability of services registered in it as well as how memory mechanisms effects the service selection process.

To test our approach, we proposed a process that could be used to configure the Cuāyōllōtl a Biologically Inspired Cognitive Architecture (BICA) (PARRA; Madrigal Díaz; RAMOS, 2023; MARTIN et al., 2022; DOUNCE; PARRA; RAMOS, 2022; GÓMEZ-MARTÍNEZ et al., 2023; SANDOVAL; RAMOS, 2021; HERNÁNDEZ et al., 2022) in order to perform a specific task. In that proposal presented in (SALAZAR; SCALABRIN; CORCHADO, 2022), the proposed directory (memory) stores both the registries of the available resources and the plans that dictate which resources to use as well as which steps to take to undertake a task.

1.2 Justification

Due to the naturally highly dynamic nature of pervasive systems, endowing a pervasive service composition system/model with a cognitive/human-like memory would allow it to add, update and delete service registries according to incoming information from both the environment and the system itself in a similar fashion to how the human brain manages short and mid-term information. This will allow the memory to quickly add or update information about the environment involved in pervasive service composition, allowing the whole system to achieve a higher degree of consistency between the services registered in a directory and those available in a pervasive environment, enabling us to propose a solution to different challenges previously described. Our proposal also considers this directory stores and retrieves previous compositions in order to employ them for recurrent situations, improving the system's performance. This mimics the way long-term information about task solutions is stored and

retrieved from the human brain as experience, enabling us to improve our performance achieving a task or similar tasks.

1.3 Motivation

By developing a service directory with cognitive memory-like capabilities, we hope to contribute to solve open problems like those described above, that are present in fields of knowledge such as: domotics, smart spaces, smart cities, ecosystems among many others.

1.4 Goals

The general goal of this work is to design a brain-inspired pervasive service directory based on neuroscientific evidence and psychological models/evidence, capable of improving the consistency between available services in the environment and services registered in the directory. The specific objectives to achieve our general goal are the following:

- Define the required characteristics/features for a brain-inspired pervasive service directory.
- Propose the data structures required to store information about services and compositions in a brain-inspired service directory.
- Propose algorithms for managing service and composition registries within the service directory.

1.5 Document Organization

This document is organized as follows. In Chapter 2, we review terms and related works to this research in order to highlight the contributions of this proposal. In Chapter 3, we describe our proposal for a cognitive memory-based

service directory. In Chapter 4, we present some details about our implementation. In Chapter 5 we describe a case study to validate our proposal, as well as a discussion about the results. Finally, in Chapter 6 we present some closing remarks.

2

Theoretical Framework

There has been many fields in which Cognitive Architectures (CA) have been employed for expanding and improving the capabilities of computer systems, such as assistance systems (YANG et al., 2022), rehabilitation (GONZÁLEZ; PULIDO; FERNÁNDEZ, 2017; ZHAI et al., 2014) and robotics (BURGHART et al., 2005; PINTO et al., 2021; TRAFTON et al., 2013). Also, there are works in the IoT field that propose to use cognitive computing (MODHA et al., 2011) to embed IoT systems with an AI that works on a similar way to the human mind (PLOENNIGS; BA; BARRY, 2018; PRAMANIK; PAL; CHOUDHURY, 2018). However, such works employ the cognitive approach just for processing information, without going further beyond towards AI planning or cognitive memory management. That is, the employment of CAs in the pervasive systems field remains quite limited as shown in this chapter. In this section we review the cognitive memory which we use as a foundation to build our work, from both the neuroscientific and psychological approaches. We also review the pervasive service discovery process, which is fundamental for our proposal, touching upon its definition and common approaches to perform it, as well as the fundamental problems related to such approaches.

2.1 Cognitive Memory

The interoception and proprioception among other cognitive functions allow the human brain to be conscious of its state, that is among other things to know

the resources and any change in them it has available (at any time) to face a task. In a similar way, through the sensory cognitive function (visual, aural, gustatory, propioceptive and olfactory) is able to sense external resources and their evolution to solve a task. The memory is also involved in this process of being aware of the resources a human has for solving a task. Memory is defined as the brain process tasked with encoding, storing, retrieving, and forgetting knowledge about the word (KANDEL et al., 2021). Thus, it helps not just to remember what kind of resources we use to solve a simple task, but also the way we combine (compose) different resources to solve complex tasks.

Many fields of knowledge have taken interest in explaining and modeling how memory performs all its tasks. For decades, researchers in such fields have performed many kinds of experiments in order to try to explain the inner workings of human memory in order to figure out, among many things, how memory is structured, how it performs its functions, and how it interacts with other brain functions. Perhaps the two fields most involved in answering these questions are neurosciences and psychology, which have engaged in different approaches to explain not only how memory operates, but the whole human cognition process. In the following sections, we present two different approaches to explain and model the working of human memory: the first one corresponding to neurosciences and the second one based on psychology.

2.1.1 Neuroscientific approach: BICAs

A *Cognitive Architecture* (CA) can be defined as a "structured systematization of the cognition process, expressed in a representative language" (CASTILLO, 2020). CAs have been a research field for decades, aiming to produce Artificial General Intelligence (AGI) models that behave in a similar way to the human cognitive process. CAs are based on the developments and evidence presented by the *cognitive sciences*: research fields aiming to explain and replicate the human cognitive process, either as a whole or just some aspects of it, such as neurosciences, psychology, artificial intelligence (AI) and philosophy, among others.

Among CAs, those whose origin, purpose and representative language are inspired in biologic procedures are known as *Bio-inspired Cognitive Architectures* (BICA) (CASTILLO, 2020). Commonly, BICAs are composed of cognitive functions according to the discipline or disciplines from which they take their

foundations. Thus, it is normal to find CAs for different cognitive functions such as Perception, Planning, Motor system, etc.

Among cognitive functions, the one we are concerned for our proposal is Memory. The memory cognitive function is responsible for encoding, storing and retrieving knowledge (MARTIN et al., 2022). Memory can be divided based on the time in: *Long-term Memory* (LTM) is tasked with storing large amount of data for long periods time; and *Short-term Memory* (STM), also called Working Memory (WM). This memory stores a limited amount of data (five to seven elements) for a short time (KANDEL et al., 2021). In (CASTILLO, 2020) Long-term memory can be divided into next three levels levels according to their retention time:

- Associative-short term storage (ASTS): works as a temporary buffer that performs associations between the data arriving at time t and data present up until time $t - 1$. It has a direct, fast link with STM.
- Mid-term memory (MTM): holds data that can be manipulated in a fast fashion. It stores both associated data coming from ASTS and non-associated data arriving directly at MTM. Just like ASTS, it has a fast, direct link to STM.
- Persistent storage (PS): permanently stores data coming from MTM, without having a cap on how much data it can hold. The time required for manipulating its entries is slower.

Different CA proposals have appeared through decades of research in this field, with each one following different theories from different cognitive sciences, and, as a result, have different designs and capabilities both in general and for its individual cognitive modules. Hereunder, we present some of the most prevalent CA's in the literature (SAMSONOVICH, 2010).

The *State Operator and Result* (SOAR) (LAIRD; NEWELL; ROSENBLOOM, 1987) is a CA intended for developing systems endowed with general intelligence. It was created by the researchers John Laird, Allen Newell, and Paul Rosenbloom, and is based around the problem-space computational model (PSCM) (YOST; NEWELL, 1989). Memory in SOAR divided in two modules:

long-term memory and short-term memory. In turn, long-term memory is divided in two types: declarative memory, tasked with managing long-term semantic and episodic knowledge for representing facts about the agent's world and snapshots of the agent's experience, respectively; and procedural memory, which represents the know-how and timing to perform certain tasks. Meanwhile, short-term memory, also called working memory, is a global memory used as a place to store the representation of the current state of the agent, and from which the retrieval of knowledge stored in long-term memory takes place.

Some limitations present in SOAR's memories are the lack of memory mechanisms such as reinforcement and forgetting, which are useful where certain bits of information have become either more relevant due to usage or irrelevant due to lack of use, respectively. In addition, the memory modules cannot work standalone, preventing them for being employed by modules/systems outside SOAR.

Adaptive Control of Thought-Rational (ACT-R) (ANDERSON et al., 2004) is both a human cognition theory and a CA based on such theory. The ACT-R CA consists of modules such as goal, perceptual-motor, declarative memory and procedural memory, each one based on a different cortical region, and they communicate among themselves by placing information chunks in the modules' buffers and then each module analyzing the chunks in its buffer in search for patterns, followed by using production rules to respond to detected patterns. There are two memory modules present in ACT-R: a declarative memory which manages representations of facts, and a procedural memory tasked with storing production rules, which represent the know-how of different tasks. There is a lack of a dedicated short-term memory module, due to some short-memory tasks being performed by the module buffers and by declarative memory elements that are activated if they surpass a certain threshold.

ACT-R is limited by the fact that it lacks an specific way to represent episodic information, which can be used to represent the state of an agent and the world around it, which is a very important thing to have when operating on dynamic environments and situations, given that episodic information can be used as reference to detect changes in the environment.

The *Learning Intelligent Distribution Agent* (LIDA) is a CA developed by the Cognitive Computing Research Group at the University of Memphis, as an extension of the *Intelligent Distribution Agent* (IDA) intelligent agent project. LIDA is built using the Global Workspace Theory (GWT) (BAARS, 1988; BAARS;

FRANKLIN, 2009) as a foundation, in addition to other theories coming from neurosciences and psychology. The LIDA cognitive cycle is the result of the collaborative work between a group of subsystems: sensation, perception, working memory, transient episodic memory, learning, among others. LIDA has four memory subsystems: working memory, transient episodic memory, sensory memory and long-term memory, with the latter divided into perceptual associative, spatial, attentional, declarative and procedural memories. LIDA memories have memory functions (e.g. encoding, retrieving, decay, etc.) implemented. The main drawback of LIDA's memory component is that it cannot work standalone, thus needing the whole CA in order to be used, and there is no enough information to use this architecture as base to solve problems presented in the introduction of this dissertation.

iCub (METTA et al., ; VERNON; HOFSTEN; FADIGA, 2011) is a BICA applied to a robot developed by the European *RobotCub* project with the goal of performing research about embodied cognition by creating a child-like humanoid robot. iCub is powered by a CA built using YARP (METTA; FITZPATRICK; NATALE, 2006), a robotics framework developed by the iCub team, with the goal of creating an abstraction layer for both software modules and hardware.

iCub is composed by thirteen modules: Exogenous Salience, Endogenous Salience, Egosphere, Attention Selection, Episodic Memory, Procedural Memory, Affective State, Action Selection, Gaze Control, Vergence, Reach & Grasp, Locomotion and the iCub Interface (VERNON; HOFSTEN; FADIGA, 2011).

While iCub possesses both Episodic and Procedural memories, it lacks shotterm memories, with some of its functions being present in the long-term memories. Also, its memory representations are focused on visual information, greatly limiting the type of applications it can perform. Last, it seems to lack memory mechanisms like decay, which are important part of cognitive memory, and of great help in dynamic environments.

Executive Process-Interactive Control (EPIC) (KIERAS; MEYER, 1997) is a CA developed by researchers David Kieras and David Meyer with the goal of modeling the way humans perform complex multimodal tasks. EPIC takes external stimuli as input and, based on the current task being performed, it uses production rules to generate a response in real time. EPIC is composed by processors, which are divided in three kinds: perceptual processors such as Visual and Auditory Processors, motor processors like Vocal Motor and Manual Motor processors, and Cognitive Processors like the Production Rule Interpreter and

Working Memory, which in kind is partitioned into different memories: one Working Memory partition for each non-cognitive processor.

While EPIC makes a distinction between declarative and procedural knowledge, with the former being stored in a permanent rule storage and the latter in a permanent declarative information storage, only Working Memory is formally established as a memory processor, being a temporary memory for information employed by the production rules; thus, there is no dedicated long-term memory processor. It also lacks many memory functions, like decay, showing a behavior not very similar to that of humans and making it unsuitable for applications where information expiration is desirable. Finally, its Working Memory processor cannot work independently from the rest of the processors, severely limiting its application potential.

Local, Error-driven and Associative, Biologically Realist Algorithm (Leabra) is a CA that started off as a neural network developed based on the neural mechanisms of the neocortex, which then expanded to different brain areas (O'REILLY; HAZY; HERD, 2017). Leabra is organized into multiple levels, according to the scale of the brain structures modeled: it goes from the micro-level representing individual neurons all the way to macro-level which represents brain areas such as prefrontal cortex and hippocampus. Different memory types are present, such as episodic memory being located on the hippocampus area or working memory on basal ganglia, but they rely on a connection-based approach that limits the representation of the information stored on them, as well as memory functions such as decay. This leaves Leabra with memories that greatly represent memory working on the micro level, but that lacks many macro level properties of the human memory, that would be of great utility for many applications. Cuāyōllōtl (a Nahuatl term which means brain or smart) is a CA under active development at CINVESTAV Guadalajara with the goal of endowing virtual creatures with human-like behavior. Cuāyōllōtl uses cognitive sciences such as neurosciences, psychology, philosophy and AI as a foundation for its development. Cuāyōllōtl is currently composed by cognitive modules such as Perception, Sensory, Working Memory, Declarative Memory, Motivation, Motor System, Planning, Decision-making, Attention and Emotions. In this CA, Declarative Memory is divided into Semantic and an Episodic memories, and Working Memory serving as a short-term memory endowed with memory processes such as decay and reinforcement, and tasked with processing and integrating

CA	Has dedicated LTM?	Has dedicated STM?	Has memory mechanisms?	Can memories work standalone?
SOAR	Yes	Yes	No	No
ACT-R	Yes	No	Yes	Yes
LIDA	Yes	Yes	Yes	Yes
iCub	Yes	No	No	Yes
EPIC	No	Yes	No	No
Leabra	Yes	Yes	No	Yes
Cuāyōllōtl	Yes	Yes	Yes	Yes

Table 2.1: Comparison table of the reviewed CAs

information coming from the Perception modules, creating a global snapshot of the system, and querying longer-term memories for information.

Table 2.1 shows a comparison between the CAs reviewed above. As we can notice, a problem a lot of the reviewed CAs share is the lack of inherent memory mechanisms such as decay and reinforcement, which can be very useful to deal with dynamic environments: decay helps us discard information from memory when it stops being relevant (e.g. I can forget about a tool near me if I don't need it for anything), and reinforcement can help us retain and refer to information that has proven to be useful in a task (e.g. if a tool has proven useful for a task, I might want to remember it and refer to that memory when doing the same task again in the future).

Also, some CAs don't have dedicated long-term or short-term memory modules. Long-term memory is the storage for permanent information about things like concepts, events, plans, among other things, and the lack of such memory means that a system wouldn't be capable to consolidate short-term information into long-term knowledge, thus being unable to learn. Meanwhile, short-term memory is where information from all around the brain is integrated and manipulated, so the lack of it, with all the memory functions it encompasses, not only moves a CA away from a from human-like behavior, it also limits its usage in applications dealing with dynamic environments where information coming from modules such as sensory and planning needs to be integrated and manipulated to quickly deal with changing situations.

2.1.2 Psychological approach: Dynamic Memory

Another approach to cognitive memory comes from the psychology-based *dynamic memory*. A memory is known as dynamic if it is able to alter its own organization with the processing of new information (SCHANK, 1999). In dynamic memory, information already stored in memory is used as a basis to interpret new information, either because they are similar or related: an object can remind you of another object because they are similar in some way (e.g. a basketball ball can remind you of an orange because both are round and orange-colored) or because of their relationship (e.g. an orange can remind you of a fruit shop because you can find oranges there), or it can remind you of an event it took part on (e.g. a hammer can remind you of the time you hit your hand with it trying to hammer a nail to a wooden plank). Every time the memory processes information, it results in a change of the memory structure, which is in itself the output of this process.

Every experience processed by the memory is tried to be understood in terms of the information already stored in memory, and the process of reminding can happen in three different situations:

- Processing-based reminding: happens during the processing of a new piece of information, e.g. when eating a dish for the first time, by trying to explain its taste, we might try to relate it to the taste of a dish we already know. In turn, processing-based reminding is subdivided into the following types:

  - *Event expectations-based reminding*: happens when we are experiencing an action in a known situation, and based on the knowledge of such situation, we remember and expect a following action, e.g. if I'm at a birthday party and I see that the birthday cake is being brought, I remember (and expect) that the following action is gonna be singing the "Happy birthday" song. If the expected action happens, the relation between both actions is reinforced, but if the expectation fails (e.g. the anticipated event doesn't occur) then the failure is stored and indexed under the point of failure.

- *Goal-based reminding*: it takes place when, during the process of trying to understand the motivations of an actor during an event, we are reminded of another actor's goal that can be similar, even if the contexts of both events are different, e.g. a person buying some gasoline everyday instead of filling the vehicle's tank in one sitting can remind me of another person that prefers to buy a few groceries every week instead of buying all the groceries for the month in one single shopping trip. The reminding can happen because the mind keeps track of goals, and the goal subsumption that happens from this type of reminding can be used to generalize our understanding of situations.
- *Plan-based reminding*: happens when a situation reminds us of another one where both are embodied by the same kind of plan, e.g. a criminal distracting police officers in some way to help a fellow criminal rob a passerby can remind me of a school children distracting the teacher so that another classmate can run away from the classroom, since both are embodied by the plan "distract the authority to do something against the rules". Since memories can be indexed in terms of plans, a situation that can be described in terms of a specific plan can remind us of another situation in a whole different context that can also be described in terms of such plan.
- *Morality-based reminding*: it's the result of, after processing new information and drawing conclusions from it, being reminded of other situations with the same lesson, moral point or otherwise message conveyed by it, e.g. the story of a friend who had to work on a team project for school and neglected it until the very end, resulting in him having to beg his teammates for help can remind me of the fable of the ant and the grasshopper, because in both cases, a lesson is to avoid neglecting your duties. This type of reminding implies the existence of both high-level memory structures representing to lessons, morals

and messages, as well as a high-level analysis of new situations to match possible conclusions from them to existing ones in memory.

  - *Intentional reminding*: happens when trying to recall a relevant experience or situation, e.g. if I ask a bartender friend about a particular client he attended in one of his shifts, he might not remember because of the vague inquiry, but if I mention something more precise, like the fact that such client, left a hefty tip, he might remember, because such detail is uncommon and helps narrow down the search in memory. Unlike the name may imply, intentional reminding isn't necessarily conscious, since intentional reminding can happen just by thinking about our current situation: in intentional reminding, by questioning ourselves about a specific situation, we can consciously or unconsciously narrow down the contexts and make more likely to fetch a desired memory.

- *Dictionary-based reminding*: happens when we try to recover information about a concepts we don't employ frequently in our daily lives. Unlike a standard dictionary, where concepts have a semantic definition, in our memory's dictionary a concept is defined in terms of features such as how to employ the object or term represented by the concept, associated feelings to that concept, the circumstances where such concept first appeared in our lives, etc., e.g. if I was a person unfamiliar with computers, I could remember the concept computer by defining it on the basis of the first time i used one, which tasks I can perform with it, etc. As we get accustomed to a concept, dictionary-based reminding for such concept disappears.

- *Visually-based reminding*: performed specifically during visual recognition, it happens when two distinct things look alike, one might remind us of the other one, e.g. when one meets a new person, that person's face might

remind us of the face of a person we already know, due to them sharing similar facial features.

The basic memory unit employed in Dynamic Memory is the *Memory Organization Package* (MOP), used to represent knowledge about concepts stored in memory. A MOP has *norms* that represent basic characteristics about the knowledge represented by it. MOPs are treated as both memory and processing structures: for every concept represented in memory, the corresponding MOP is tasked with both managing and processing its information.

MOPs can have both *abstractions* and *specializations*: an abstraction represents a generalized version of the knowledge of a MOP (e.g. the MOP that represents fruits is an abstraction of the MOP representing apples), while a specialization stands for a more specific version of the knowledge of a MOP (e.g. the MOP representing cars is a specialization of the MOP representing vehicles). A MOP referring to a particular occurrence of the concept of a MOP is known as *instance* (e.g. the MOP representing my car is an instance of the MOP representing cars).

MOPs are connected with each other through *links*. There are several types of links (RIESBECK; SCHANK, 1989):

- *abstraction links* connect a MOP with its abstractions, e.g. a link connecting the MOP representing apples with the MOP representing fruits.
- *scene links* connect a MOP representing a complex concept with MOPs representing elements or components from that concept, e.g. the MOP representing reunions with the MOP representing places, tagged as "place".
- *index links* connect a MOP with its specializations, and are tagged with a key-value pair, e.g. a link connecting the MOP representing figures with the MOP representing triangles, with the label sides = 3.
- *example links* connect a MOP with a prototype example of such MOP, e.g. a link connecting the MOP representing houses with the MOP representing my house, the one I know the most.

- *failure links* connect a MOP with an instance representing an expectation failure for such MOP, e.g. a link connecting the MOP representing vacation trips with the MOP representing the one where everything went wrong.

Abstraction links create an *abstraction hierarchy* that can be used to go from specific concepts towards more general knowledge. Scene links are used to create a *packaging hierarchy* to break down complex concepts into smaller components, all the way into indivisible components. Index links are used to create a *discrimination network* which helps dividing sets of MOPs into smaller, more manageable subsets.

In dynamic memory, MOPs are employed as building blocks for larger structures that represent more complex situations, like *scrips*, *scenes* and *events*.

Scripts are a set of specific sequential actions associated to constantly-repeating situations. They can be seen as a snapshot of a particular event.

Scenes group a set of actions that as a whole have a common goal and that happen within a specific time period. Specific memories grouped under scenes are indexed with regards to how they differ with the scene's general flow. A scene has a setting, a goal and actions taking place in the setting to further the goal.

Events are a set of scenes directed towards the achievement of a goal. An event has a main scene whose goal represents the essence of the event.

One of the main uses of the Dynamic Memory and its MOPs is in the field of case-based reasoners (CBR) give some references: a type of AI problem solver where new problems are tackled by employing solutions to previous problems and adapting them (RIESBECK; SCHANK, 1989; KOLODNER; SIMPSON; SYCARA-CYRANSKI, 1985). The application of this idea for planning gave rise to Case-based Planning (CBP): generating a plan to solve a new problem falling back on previous cases adapting the plans that closely matches the current predicament (HAMMOND, 1990).

Features such as multiple paths to reach specific memory items, storage and retrieval of past experiences as a foundation for creating plans for new situations, discrimination of memory elements using their attributes and the update of existing memory elements with each new element processed makes dynamic memory a powerful approach that we consider can be of great help to create a pervasive system directory, given how such features help in our task

of creating a service directory that employs past experiences to deal with new tasks, while having tools to deal with dynamic environments.

2.2 Service Composition

As described previously, the devices available in our environment make possible pervasive service composition. That is, different devices in the environment can be abstracted and exploited by third parties as *services*: abstract hardware-software entities that encapsulate a specific functionality that can be used to compose user needed complex service.

We call *pervasive services* to those software or hardware services capable of getting information from their surrounding environment, and whose functionality is performed on a limited spatial area with minimal human intervention (AZIEZ; BENHARZALLAH; BENNOUI, 2019; YANG et al., 2005). Pervasive services must comply with requirements such as *selectability* (select among different services according to their attributes), *parameterisability* (adapt a service according to user or situation needs), *inter-service dependency* (being able to link multiple services, replace any of them by other services if necessary) and *context awareness* (the capability of a service to compute context information) (FUNK; KUHMÜNCH; NIEDERMEIER, 2005). Some of the devices in a pervasive system are limited in communication and energy means (e.g. smartphones and drones having limited energy resources and wireless communication reach), and many of them constantly moving through the environment, all of which factors on their availability at a given moment

When a device requests another device to use a functionality abstracted as service, we call the device requesting the service as consumer, and the device providing the service as provider. Consumers seek services in order to employ them to contribute in the solution of a task they are undertaking. However, there may be tasks too complex to be performed by using a single service. In that case, it is needed to combine the functionalities of multiple services in order to fulfill such complex request. One approach to deal with that is *pervasive service composition*: the process of combining different services into a sequence with the goal to create a new, composite service that is able to fulfill complex requests (SALAZAR; SCALABRIN; CORCHADO, 2022).

The entire pervasive service composition process, from locating the necessary services to executing the resulting composition is a very complex process that can be divided into the next three phases (KAPITSAKI et al., 2007):

- Discovery phase: here is where the services available in the environment are found by the system, in order to be employed during the composition process.
- Composition phase: also known as planning (CLARO; ALBERS; HAO, 2006), refers to the phase where the discovered services go through a selection process and the selected ones are combined to create a new, complex service to solve a complex task.
- Invocation phase: also known as execution, it's when the selected services for a composition are called and executed according to the composition flow to solve a the task needed to solve.

The discover of services in the environment is quite important and challenging for the pervasive service composition problem. Important because the more precise the information provided by this discover service, the better are the decisions that will take place for performing compositions, resulting in a better performance for the composed service. It is quite challenging mainly because of dynamic characteristics of the environment, where services may go through the space of interest for very short of time, may fail because of energy, hardware or software problems, and also because the arrival of a better option than the previously selected service must be taken into account to improve service composition performance.

To fulfill this task of service discovery, a pervasive service composition system can follow either a directory −based model, or a directory −free model. In the former case, the system employs a data structure known as *service directory* to store and manage information related to services available. In the latter case, service providers and consumers communicate directly among themselves, which increases resiliency compared to the directory-based approach since there is no centralized point of failure, but it induces a heavier communication complexity, since every consumer must communicate with every reachable provider. It thus

requires that either providers and consumers know each other before establishing communication, which could prove challenging in a dynamic environment, or the presence of intermediary entities (e.g. brokers) in the discovery process.

Depending of how service registries are managed in the service discovery process, service registration approaches can be classified in the following three types (TJIONG; LUKKIEN, 2008): First: the *stateless* approach that is the case when service registries are not kept; second: the *hard-state* approach, service registries are kept until a deregistration process is triggered by the provider; and third: the *soft-state* approach, service registries have an expiration period, so the registries have to be renewed periodically or they are removed from the registry.

For a directory-based service discovery, only the latter two approaches are relevant. Additionally, registries where both hard-state and soft-state approaches are employed are known as *hybrid-state*. That type of states has shown to be better at maintaining consistence levels that employing either soft or hard states alone (TJIONG; LUKKIEN, 2008).

There are proposals such as (WALDO, 1999b) which is a distributed architecture based on the idea of grouping different users and the resources required by them. The goal is to turn the network into a flexible tool allowing humans and computational clients to find required services (devices). However, is has limitations, such as segmenting the services into *lookup groups*, requiring a service consumer to query each group for services, and that service providers must join as many groups as possible to maximize reach, or it being limited to just the Java ecosystem.

Pervasive service compositions must be able to handle the interactions between services, the data flow of the composite service and the way the sequence must be built. Many approaches to service composition have been developed, such as based on machine learning (KHANOUCHE et al., 2019; WANG et al., 2016), game theory (OSTOS et al., 2015; LEI; JUNXING, 2017), self-organization (CERVANTES et al., 2018; CABRI; MARTOGLIA; ZAMBONELLI, 2016; GUTIÉRREZ-GARCIA; SIM, 2013), optimization (SOMMER; MAHÉO; BAKLOUTI, 2020; LIU; CAO; WANG, 2017; ZHOU et al., 2018; ALSARYRAH; MASHAL; CHUNG, 2019) and genetic algorithms (BRISCOE; WILDE, 2008), among others. However, despite the variety of approaches, only a very limited amount of them employ hybrid-state service directories: a big number of the reviewed works don't employ a directory at all, and most of those that do only use

hard-state registries, which directly impacts the consistency of the directory's registries, given the degree of dynamism present in pervasive environments.

There have been many works that deal with service discovery, either as standalone solutions or as part of a bigger entity, such as a middleware or a service composition model. Table 2.2 presents a comparison between some representative works. From the reviewed works, we can see a fairly even split between directory-based and directory-less discovery approaches: while directory-based can introduce centralized points of failure, directory-less approaches can incur in heavy communication complexity as the number of participants, and therefore service providers, grows. Among those approaches employing directories in the discovery process, most employ hard states, which can introduce inconsistency since they require the service provider to explicitly request the deregistration of a service, and given the high dynamism inherent to pervasive environments, such request may become impossible if the provider becomes unavailable, leaving the directory with an inconsistent registry. Also, none of the reviewed works stores information from past compositions, which means that compositions must be created from scratch every time a change occurs in the environment or to the composition, even if such situation has previously been grappled with in previous cases. In other words, the proposals do not take advantage of previous experience, representing not only wasted effort, but also the lack of a capability that serves as the foundation for the learning process (SCHANK, 1999).

After reviewing the state of the art, we find only one work employing a brain inspired cognitive architecture or BICA approach for service composition, which is COPERNIC (ROMERO, 2019). The author proposes a multi-agent-based composition model in which cognitive agents, which employ cognitive functions and following the steps established in the Common Model of Cognition (LAIRD; LEBIERE; ROSENBLOOM, 2017), would be tasked with selecting, composing and invoking services in a pervasive environment. However, COPERNIC shows the same shortcomings of the non-BICA approaches, in addition to presenting other issues like the lack of filtering of data flowing from the Perception module towards the Working Memory module, which could potentially cause a bottleneck; the employed device grouping technique makes the model biased towards devices belonging to the user, potentially leading to ignore well-performing services on remote devices; the requirement of static devices such as desktops and servers, something not present in many pervasive environment scenarios; and

the lack the capacity to store and retrieve previous compositions, meaning that any composition process must always start from scratch.

This fact shows that the CA approach is not exploited in the service composition field. Features such as being able to store and retrieve previous useful task solutions (pervasive service compositions) on memory for later use, can be useful for instance to change a plan if the goal changes mid-task.

Work	**Type**	**Employs directory?**	**State type**	**Employs information from past compositions?**
Jini (WALDO, 1999a)	Distributed Middleware	Yes	Hybrid	N/A
Corba (VINOSKI, 1997)	Distributed Middleware	Yes	Hard	N/A
UDDI (SABBOUH et al.,)	Discovery Model	Yes	Hard	N/A
(DOULKERIDIS; VALAVANIS; VAZIRGIANNIS, 2003)	Discovery Model	Yes	Hard	N/A
MAPS (SHEU et al., 2009)	Composition Model	No	N/A	No
K-Directory (RAYCHOUDHURY et al., 2011)	Discovery Model	Yes	Weak	N/A
(GUTIÉRREZ-GARCIA; SIM, 2013)	Composition Model	Yes	Hard	No
(CERVANTES et al., 2018)	Composition Model	No	N/A	No
MAS-ASC (CHAIB; BOUSSEBOUGH; CHAOUI, 2017)	Composition Model	No	N/A	No
GoCoMo (CHEN; CARDOZO; CLARKE, 2018)	Composition Model	No	N/A	No
OCE (YOUNES et al., 2018)	Composition Model	No	N/A	No
(MASCITTI et al., 2018)	Composition Model	No	N/A	No
(CABRERA et al., 2018)	Discovery Model	Yes	NOS	N/A
(BAKLOUTI; Le Sommer; MAHEO, 2019)	Composition Model	No	N/A	No
COPERNIC (ROMERO, 2019)	Composition Model	Yes	NOS	No

Table 2.2: Comparative table between different service discovery approaches.

3

Memory Design

Performing efficient service compositions in a pervasive environment is an open problem. However, it is present in many fields such as IoT and Smart ecosystems (OSTOS et al., 2015), with the main requirement to solve it is to know accurately which services are available when the composite service is created or restructured. As was explained above, pervasive service composition ought to deal with the failure of one or more elements that are part of the pervasive composite service. As described previously, common approaches restart the pervasive service composition from scratch, while in this dissertation, we propose the substitution of the failing service for another equivalent, taking advantage of even new incoming services or, even better, to recompose the pervasive service using services providing better performance than the currently selected, all this taking into account the time the pervasive service is needed.

This thesis contributes to this solution by proposing a pervasive service directory whose structure is based on the Dynamic Memory MOP structures (RIESBECK; SCHANK, 1989), and employing the Cuāyōllōtl BICA Memory modules to manage the MOPs, thus creating a memory structure that can store both pervasive service registries and previous composition solutions (MARTIN et al., 2022; CASTILLO, 2020). In our proposal, pervasive services would be registered in a service directory in memory, and the registries managed according to messages coming from different Cuāyōllōtl modules.

For this chapter, we first present the requirements that our proposal must fulfill. The following section shows the types of services that we consider for this work. After that, we present the Cuāyōllōtl cognitive modules we consider

as involved in the directory management tasks. Then, we proceed to show how the directory is structured in memory based on MOPs. Finally, we proceed to show how our proposal would fit within a CBP-based service composition process employing the Cuāyōllōtl BICA.

3.1 Requirements

As a result of the literature review from the previous chapter, we have detected a set of requirements that a pervasive service directory must have in order to have the required capabilities for managing service registries in a way that allows it to keep a high degree of consistency with the status of pervasive services in the environment.

Register services at the petition of the service providers		
Precondition	**Trigger**	**Post-condition**
A new pervasive service enters the environment.	Message from service provider containing a name, an IP direction, a port, a timestamp, a set of attributes and their values.	Memory structures created for the service registry and its attributes.

Deregister services that go past their expiration times		
Precondition	**Trigger**	**Post-condition**
A service registry with an associated timestamp that crosses the expiration time threshold.	Service directory receives a message signaling a service has crossed the expiration threshold.	Memory structures related to the service and its information deleted.

Deregister a service at provider's request		
Precondition	**Trigger**	**Post-condition**
A service provider requests the deregistration of a service.	Service directory receives a message requesting the deregistration of a service, containing the service name.	Memory structures related to the service and its information deleted.

Deregister a service due to its unavailability		
Precondition	**Trigger**	**Post-condition**
A service that is part of a composition has failed to successfully execute.	Service directory receives a message requesting deregistration of a service, containing the service name.	Memory structures related to the service and its information deleted.

Renew the timestamp of a registered service		
Precondition	**Trigger**	**Post-condition**
A new pervasive service enters the environment	The service directory receives a service registration message for a service already registered.	The memory structure representing the service's old timestamp is replaced by one representing the new timestamp.

Store feedback for both service compositions and its individual components after execution.		
Precondition	**Trigger**	**Post-condition**
A service composition has successfully finished its execution.	The service directory receives a message containing numeric values representing feedback from the composition execution.	Memory structures containing feedback history for the compositions and individual services are updated.

3.2 Types of Services

In order to better manage the pervasive services registered in the directory, as well as to facilitate adaptability for service composition models employing our directory, we classify services into the following types:

- concrete services (CS): pervasive services available in the environment through devices.
- abstract services (AS): represent a specific functionality and group together a set of services. There are two types of ASs:
  - Atomic abstract services (AAS): these group CSs together according to the functionality they offer: e.g. all pervasive services offering a visual feed are grouped under the AAS "camera", all pervasive services offering an audio feed under the AAS "microphone", etc. Every AAS holds pointers to registered CSs that belong to the category it represents.
  - Composite abstract services (CAS): they group other ASs that take part in a specific service composition: they encompass all the component services that are used by specific compositions, e.g. the CAS

representing a service composition for visual monitoring in an assisted living system would group the ASs required for that task, such as cameras and devices that process the cameras' video feeds.

As an example, let's assume the scenario including a room with two cameras, a speaker and an embedded computing device, all connected through a network. This pervasive environment has four CSs: the camera service `cam.01` offered by the first camera, camera service `cam.02` offered by the second camera, the speaker service `speaker.01` offered by a device with speakers, and the processing service `comp.01` offered by the embedded computer device. These CSs would be grouped under three AASs: the two camera CSs would be grouped under a `camera` AAS, the speaker CS would fall under the `speaker` AAS and the processing CS would be placed under the `processing` AAS. If we created a service composition tasked with greeting any person entering the room, we could perform this task using the `camera`, `speaker` and `processing` AASs as components, which would be grouped as such under a `greeting` CAS representing the composition.

The reasoning behind this distinction is the following: we believe that by differentiating among ASs and CSs, we allow a service composition model employing our proposal to decouple the more abstract functionalities that services provide from the more technical capabilities provided by each service's attributes, thus allowing a service composition model to carry out two different kinds of adaptations: adapting to a service unavailability by swapping the unavailable CS to a new one with the same functionality, and altering a composition by swapping a type of AS by another one with different capabilities. This enables us to swap CSs in a composition without altering the composition's structure.

Another way this could be useful is to opportunistically allow a composition to swap a CS from a composition should a new CS from the same AAS but with a better performance be registered before the composition is executed, e.g. if a composition employs the CS `cam.02` corresponding to the `camera` AAS, but before executing it a new CS named `cam.03` is registered in the directory under the `camera` AAS and has a better definition attribute, a composition model

could take this opportunity to swap `cam.02` for `cam.03` in order to get an overall better-performing composition

Also, by treating the AS sets representing service composition's components as another type of ASs, we enable service composition models to employ existing compositions as components for a new composition, allowing service composition models to use existing compositions as building blocks for even more complex compositions, instead of limiting them to only employing atomic services as components, which would limit the complexity for possible compositions and force the composition model to rebuild compositions from scratch just to employ them as part of a new composition, thus going against the spirit of employing existing compositions to tackle new tasks. This also mimics how we, as humans, formulate plans: we can employ basic plans to create complex plans, which in turn can be used to create even more complex plans, e.g. the plan to prepare a specific meal can be used as a component for the more complex task that can be to figure out a plan for a romantic dinner at home, which in turn can be the key component of a grand plan that can be winning over someone's heart.

We propose two different data structures to store the different types of services. The first one is a Concrete Service Directory (CSD), where the CS registries would be stored and managed. In the second one, both AASs and CASs are to be stored in an Abstract Service Directory (ASD), where they are managed.

Figure 3.1 shows a graph representing the relations among the different types of services. The blue ellipse represents the root of the structure: the starting point for any service query. The red ellipses represent the CASs. The green ellipses represent the AASs. Finally, the purple circles represent the CSs. The solid lines represent indexing links, the dashed lines represent links between a composition and its components, and the dotted lines represent links between AASs and CSs categorized by them.

3.3 Involved Cognitive Modules

As mentioned before, our proposal is built on top of the Cuāyōllōtl BICA, given that is the one with the most suitable memory modules, as well as different modules feeding the system with both external and internal information. For

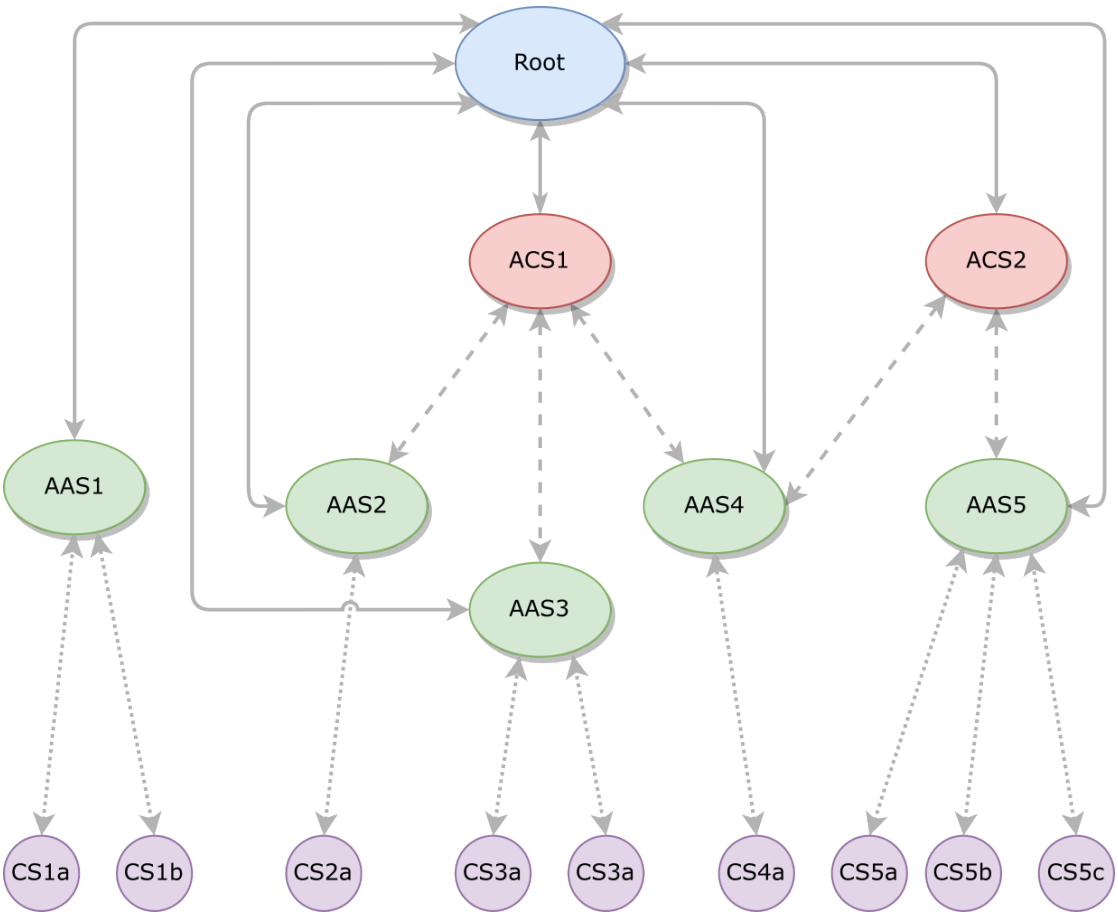


Figure 3.1: Graph representing the relationships among different service types. The blue ellipse represents the root of the structure. The red ellipses represent the CASs. The green ellipses represent the AASs. The purple circles represent the CSs. The solid lines represent indexing links, the dashed lines represent composition links and the dotted lines represent categorization links.

the functioning of our proposal, we consider the participation of the following Cuāyōllōtl cognitive modules:

- Mid-term Memory (MTM): this is where the CSD is to be located. MTM has a limited retention time, and elements stored on it are constantly decaying until they are forgotten. Our proposal allows us to implement a similar mechanism, but instead of making memory entries harder to retrieve, it can be used to compute their *relevancy* to the composition process: the older the entry is, the less relevant it becomes, enabling the composition model the option to use this relevancy value to ponder which service to select in a composition. MTM is also the end-point for messages used to manage service registries.

- Associative Short-term Storage (ASTS): it's tasked with performing the association between a newfound CS and a corresponding AAS. Every time a new CS is registered in the CSD, ASTS uses the new service's information to classify it under the corresponding AAS according to the CS's functionality.

- Persistent Storage (PS): in our proposal it's here where the ADS is located. This is the endpoint for queries looking for services for a composition and for stored composition solutions, and it's where the storing of new CASs and updating of AAs takes place.

- Percepto-Attentional Process (PAP): through its Proprioception function, it is tasked with detecting available pervasive services in the system's environment and forward their information to the CSD. It also gathers information about services that failed to execute.

- Planning: where the service composition is performed. This module queries PS to look for available services to use in compositions and for existing compositions to employ or adapt. It also provides feedback about employed CSs according to their performance.

Our work focuses on the work of the MTM, PS and ASTS modules in managing CSD and ASD registries, so we make the following assumptions regarding the process related to service information being retrieved by PAP modules and that information reaching MTM:

- When a new information about a service becomes present in the environment, the Sensory modules pick it up, and then that information is interpreted by the PAP, which then generates a message with the interpreted information and sends it to Working Memory (WM).
- Once in WM, service information stays there decaying until it falls below a threshold, after which the service data is deleted from WM and sent to MTM (CASTILLO, 2020). That time is considered to be around two seconds (BADDELEY; HITCH, 1974; COWAN, 2001).

Figure 3.2 illustrates the current architecture of the Cuāyōllōtl BICA, showcasing the cognitive functions represented as modules and their interconnections, demonstrating the relationships between them.

For this work, we define five types of messages in order to manage service registries in the CSD:

- up messages: sent by the PAP to add a CS registry every time it locates a new pervasive service in the environment, or to renew a CS registry if its provider signals that it will continue to be available.
- down messages: sent by the PAP to delete a CS registry at the provider's request. A provider could ask to take down a service for different reasons, such as low energy remaining, device soon to be out of system's range, etc.
- timeout messages: sent by the MTM to delete an expired service registry. When a CS registry goes past a predefined time threshold without being renewed, MTM alerts the CSD with this type of message.
- unavailable messages: sent by the PAP to delete the registry of an unavailable CS. If the system tries to invoke a CS during the execution of a

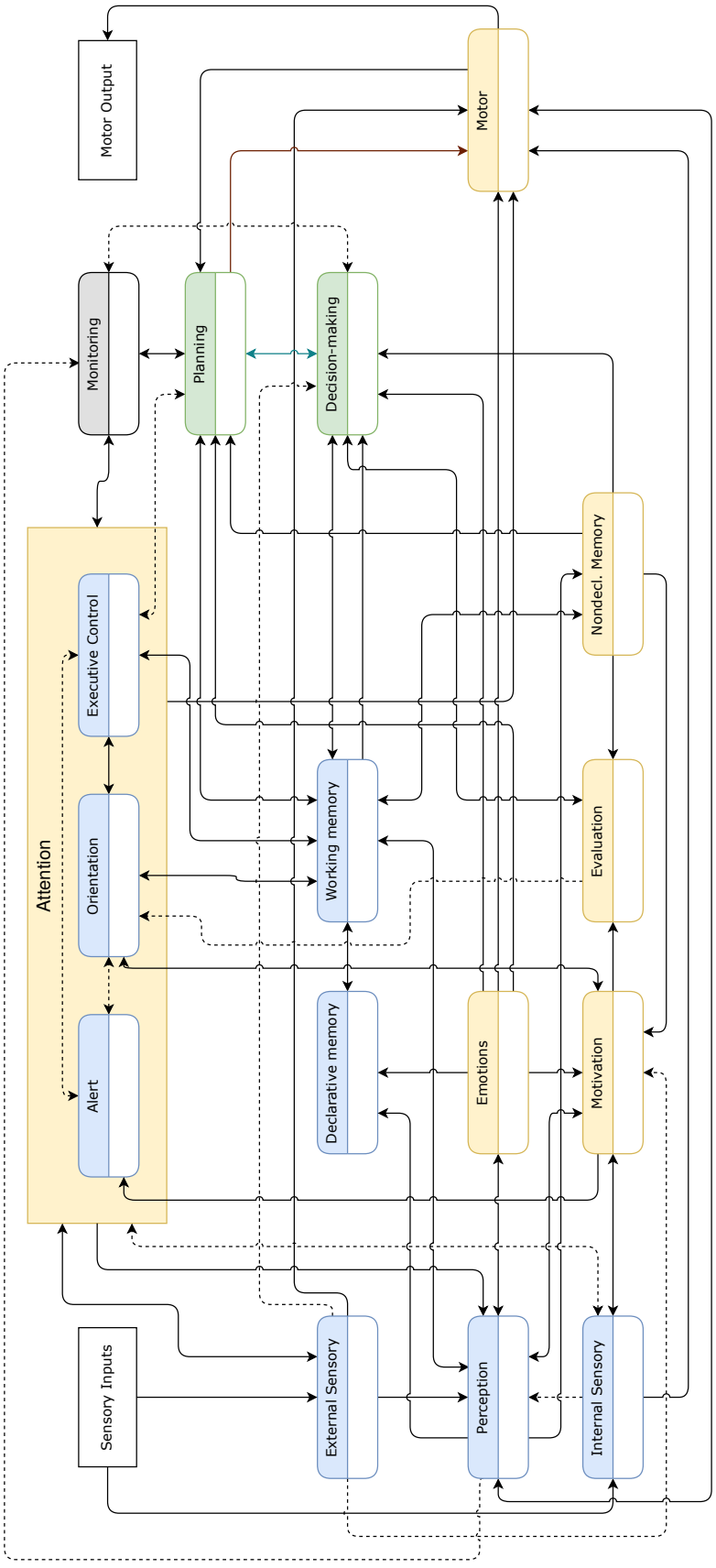


Figure 3.2: General architecture of the Cuāyōllōtl BICA. The starting point of the cognition process are the sensory inputs feeding the Sensory modules. The ending point is the Output produced by the Motor module that generally allows to interact with the environment.

composition and fails (e.g. the CS has a failure or doesn't respond), the PAP emits a message of this type.

- feedback messages: sent by Planning after the execution of a composition. It contains feedback values both for the individual CSs that took part in the composition and for the composition in general.

The ability to delete service entries from the CSD both at the system's request once a service has been deemed as unavailable (either because the provider requested it or because the service failed during the composition), and by expiration when the MTM's decay process determines that a registry has crossed the time threshold allows us to implement a service directory that employs hybrid states, which can be seen as analogous to how entries in memory decay over time or can be updated to show they are no longer available after failing to further sense them, which allows humans to quickly update the representation of our surroundings, a crucial capability since humans dwell in extremely dynamic environments.

We can see this as a similar process to how we construct episodes in memory: when we sense all the elements in our surroundings, we can build a mental structure that contains all the objects we sensed in it, and with every change we perceive, we alter that structure to reflect the sensed changes for it to be accurate with the state of the environment.

Also, information in our memory decays: the shorter the term the memory is, the faster it decays. The difference is that the memory doesn't delete information: as time passes, information that is not reinforced simply becomes harder to recover. Meanwhile, in our case, once an item becomes old enough without being renewed, we choose to delete it. It is not in our interest to hinder the retrieval of old service registries, but to simply offer that information to a composition model for it to use that information as it best suits it, and delete it after it becomes old enough to still be useful.

3.4 Memory Structure

In order to store and manage the information employed by both the CSD and the ASD, we designed a MOP-based memory structure: starting from a root MOP, known as M-Root, we derive the following specializations:

- M-Attribute: represents the values that a CS's attributes can take, e.g. quantity of remaining battery life, duration of the response time, etc. We consider five instances for M-Attribute:
  - M-Quantity: represents values associated with quantities, used to represent quantitative attributes. For this work, we only represent the values high (with I-M-High) and low (with I-M-Low).
  - M-Duration: stands for values representing a time duration. We consider the instances of short time period (with I-M-Short) and a long time period (with I-M-Long).
  - M-Distance: represents the physical distance between two places. We only represent (with I-M-Near) and (with I-M-Far).
  - M-Timestamp: stands for the timestamps for when a service registration message is emitted. Each instance corresponds to a codified timestamp sent within an up message associated to a CS.
  - M-Location: represents a place in the system's physical environment. Every instance represents a specific place in the environment.
  - M-IP: represents the service's IP address. Every instance is a specific IP associated to a registered CS.
  - M-Status: represents the status of a service, which can be either busy performing a task (represented by the instance I-M-Busy) or free to be invoked (represented by the MOP I-M-Free).

- M-Act: depicts actions that can be carried out by specific devices, e.g. watch for cameras, listen for microphones. Instances of M-Act are actions related to the capabilities offered by pervasive services.
- M-Event: stands for events, e.g. the steps of a plan. This MOP has only one specialization:
  - M-Step: represents a step belonging to a service composition. Every instance of M-Step stands for one step that takes part of one or more service compositions.
- M-Group: represents groups of elements, e.g. groups of services. It has the following specializations:
  - M-Step-Group: represents groups of steps employed by at least one service composition.
  - M-Concrete-Group: stands for groups of CSs.
  - M-Abstract-Group: standing for groups of ASs, both AASs and CASs.
  - M-Feedback-Group: represents groups of quantitative evaluations produced as feedback after the execution of a composition.
- M-Concrete-S: stands for CSs. This MOP serves as the root of the CSD, with each instance corresponding to a registered pervasive service.
- M-Abstract-S: represents ASs. It has two specializations:
  - M-Atomic: represents AASs and serves as the root of the ASD and under which all AASs are grouped.
  - M-Composition: stands for and groups all CASs.

The aforementioned MOPs are also connected via scene links in the following way:

- M-Concrete-S has one scene link pointing to the corresponding specialization of M-Attribute for each attribute, so that every instance of M-Concrete-S can be linked to the corresponding instance of M-Attribute in order to represent attributes and their values, e.g. a CS MOP I-M-Foobar-Service having a link named response-time towards the attribute MOP I-M-Short to depict that the service has a short response time.
- M-Step is linked via a packaging link named action to M-Act, to depict that every composition step is constituted by an action, e.g. the step MOP I-M-Composition has a link going towards the action MOP I-M-Composition.
- M-Concrete-Group and M-Abstract-Group have links tagged as 1 to M-Concrete-S and M-Abstract-S respectively, in order to allow instances of M-Concrete-Group and M-Abstract-Group to have numbered links towards the MOPs representing the members of such groups, e.g. the MOP I-M-Concrete-Group.1 having a link tagged 1 towards the MOP I-M-Concrete-S.1, another link tagged 2 to the MOP I-M-Concrete-S.2, etc.
- M-Atomic has a link tagged available to M-Concrete-Group, which is used to link every AAS to a CS group standing for all available CSs of that category, e.g. the AAS MOP I-M-Foobar-Service having a link to the CS group MOP I-M-Available-Foobar.
- M-Composition is linked through a packaging link tagged components to M-Abstract-Group, used to link every CAS to an abstract service group representing all the components services of a composition, which could be both atomic services or other compositions, e.g. a CAS MOP I-M-Foobar-Composition having a link to an AS group MOP I-M-Foobar-Components.

Figure 3.3 presents a graphical representation of the aforementioned memory structure. Each box represents a specific MOP, with links in Figure 3.3a

representing abstraction links going from each MOP towards its abstractions, and links in Figure 3.3b representing packaging links going from MOPs representing more complex concepts towards MOPs representing its constituent parts, with the accompanying text being the link's name. Listing 3.1 shows a pseudo-code representing the general structure of a CS, including slots (the outgoing packaging links of a MOP), while Listing 3.2 shows the pseudo-code for the general structure of a CAS, whose slots point towards the MOPs grouping a composition's component ASs, composition steps and the feedback history.

```
MOP I-M-Service
  type: instance
  abstractions: M-Concrete-S
  slots:
    attribute1: I-M-Value1
    attribute2: I-M-Value2
    ...
    timestamp: I-M-MyTimestamp
    status: I-M-MyStatus
    feedback: I-M-ServiceFeedback
```

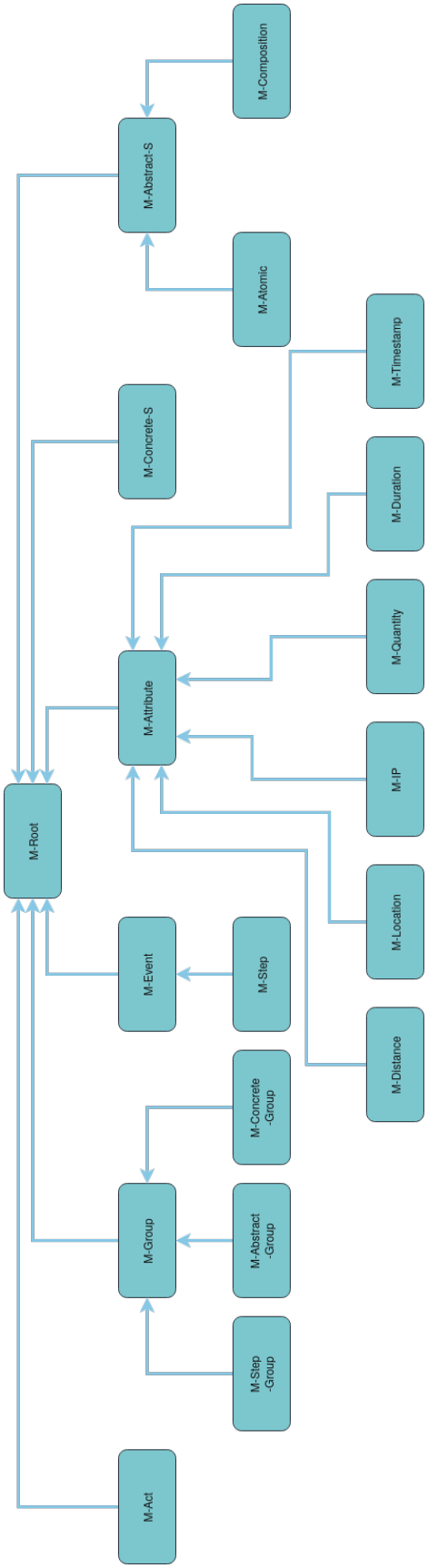
Code 3.1: pseudo-code for the MOP structure of a CS registry

```
MOP: I-M-Composition
  type: instance
  abstractions: M-Composition
  slots:
    components: M-Abstract-Group
      1: I-M-ComponentService1
      2: I-M-ComponentService2
      ...
    steps: M-Step-Group
      1: I-M-Step1
      2: I-M-Step2
      ...
    feedback: I-M-CompositionFeedbacks
```

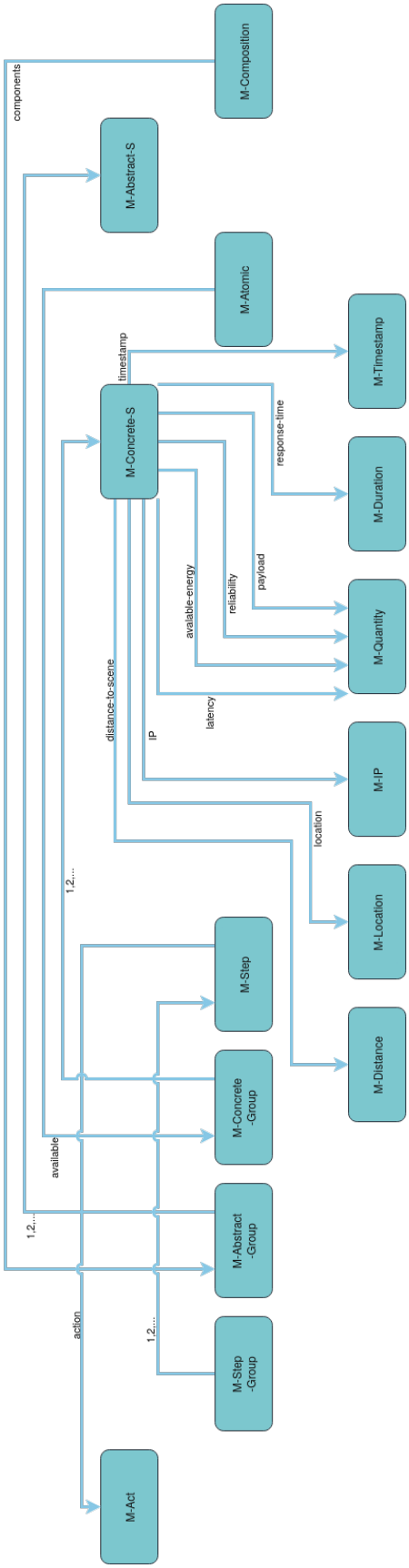
Code 3.2: Pseudo-code for the MOP structure of a CAS registry

The general/overall functioning of the directories can be described by the following processes:

- Service registration: when the system receives information about a new service, the PAP sends an up message to the CSD with the information



(a) Diagram representing the abstraction hierarchy for the proposal's MOPs.



(b) Diagram representing the packaging hierarchy for the proposal's MOPs.

Figure 3.3: Diagrams showing the proposal's MOP-based directory structure.

about the new service. The CSD uses the message information to create a new instance under M-Concrete-S, and links it with the appropriate instances of M-Attribute according to the new service's attributes. Then the new service's MOP is sent to ASTS to match it with an AAS according to its functionality. A scene link is created from the instance of M-Abstract-Group corresponding the matching AAS of the the new service's MOP.

- Service renewal: if the CSD receives an up message corresponding to an already registered CS, it resets the CS registry's timestamp by creating a new MOP under M-Timestamp with the new timestamp and linking the CS's MOP to the new timestamp.

- Service de-registration due to petition from provider: when the system receives a request to remove a service from the directory (due to the provider having low resources or being about to exit the environment, etc.), the PAP sends a down message to the CSD to delete the MOP for such service. First, ASD deletes the link going towards the MOP to be removed from the instance of M-Concrete-Group corresponding to the service's AAS, and after that, the CS MOP is deleted.

- Service de-registration due to registry expiration: every CS registry has a timestamp represented by an instance of M-Timestamp, and a predefined expiry period. MTM constantly monitors the CSD and, in case a registry passes such period without its provider sending an up message to renew its registry, it sends a timeout message to inform of such event to the CSD, which then proceeds to delete the corresponding CS's MOP in an identical fashion that when a down message is received.

- Service de-registration due to unavailability: if a concrete service is unsuccessfully tried to be used during the execution of a composition, by reasons such as service failures or the service provider becoming unavailable before notifying the system, the PAP notifies the CSD by the means of

a unavailable message. The registry removal for such service happens the same way that the two previous cases.

- Composition registration: when a new composition is created, PS adds a new instance under M-Composition, which is then linked to two new MOPs: a new instance of M-Abstract-Group representing the group of component services, which is itself linked to the corresponding AAS MOPs; and a new instance of M-Step-Group, which represents the composition steps, and is linked to the instances of M-Step representing the individual steps.

- Composition feedback: after a CAS has finished executing, feedback must be provided for both the composition and its components. Planning sends a feedback message that contains a numeric feedback value for the CAS and for each CS used in the composition. The feedback is stored in the corresponding instances of M-Feedback-Group for each MOP representing either the CAD or the component CSs.

Figures 3.4, 3.5, 3.6, 3.7, 3.8 and 3.9 show the UML sequence diagrams depicting the each one of the aforementioned processes.

3.5 CBP-based Process

We envision our directory as a part of a CBP-based service composition process, based on the modules proposed on (HAMMOND, 1990):

- RETRIEVER is tasked with retrieving both compositions (CASs) and their components (AASs): when it receives a set of goals that the system seeks to achieve through a service composition, RETRIEVER would ponder the goals to determine their importance to the task, and then find an indexed CAS capable of satisfying the highest number of goals, giving preference to the ones with higher priority.

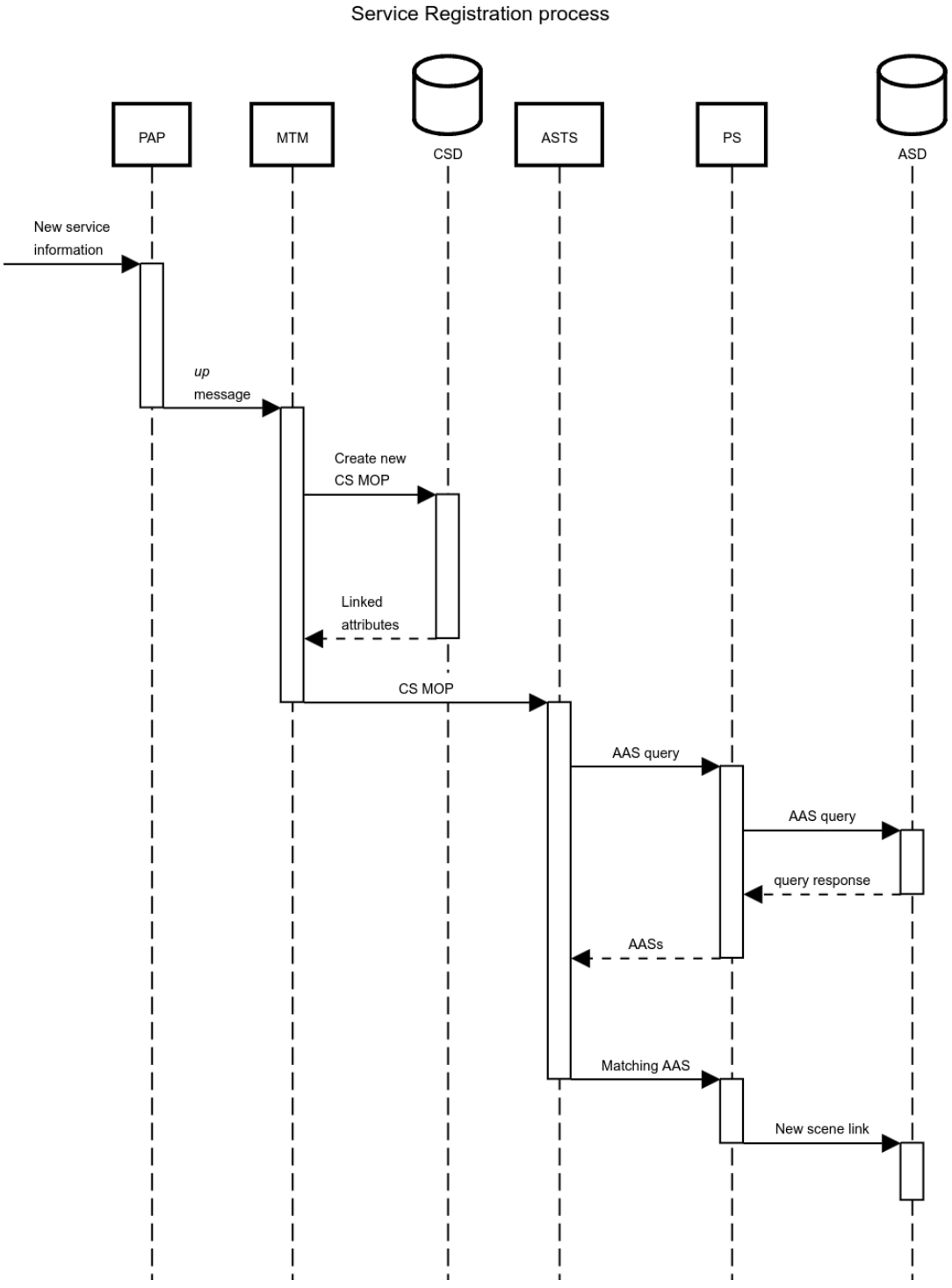


Figure 3.4: Sequence diagram depicting the service registration process started by a provider request.

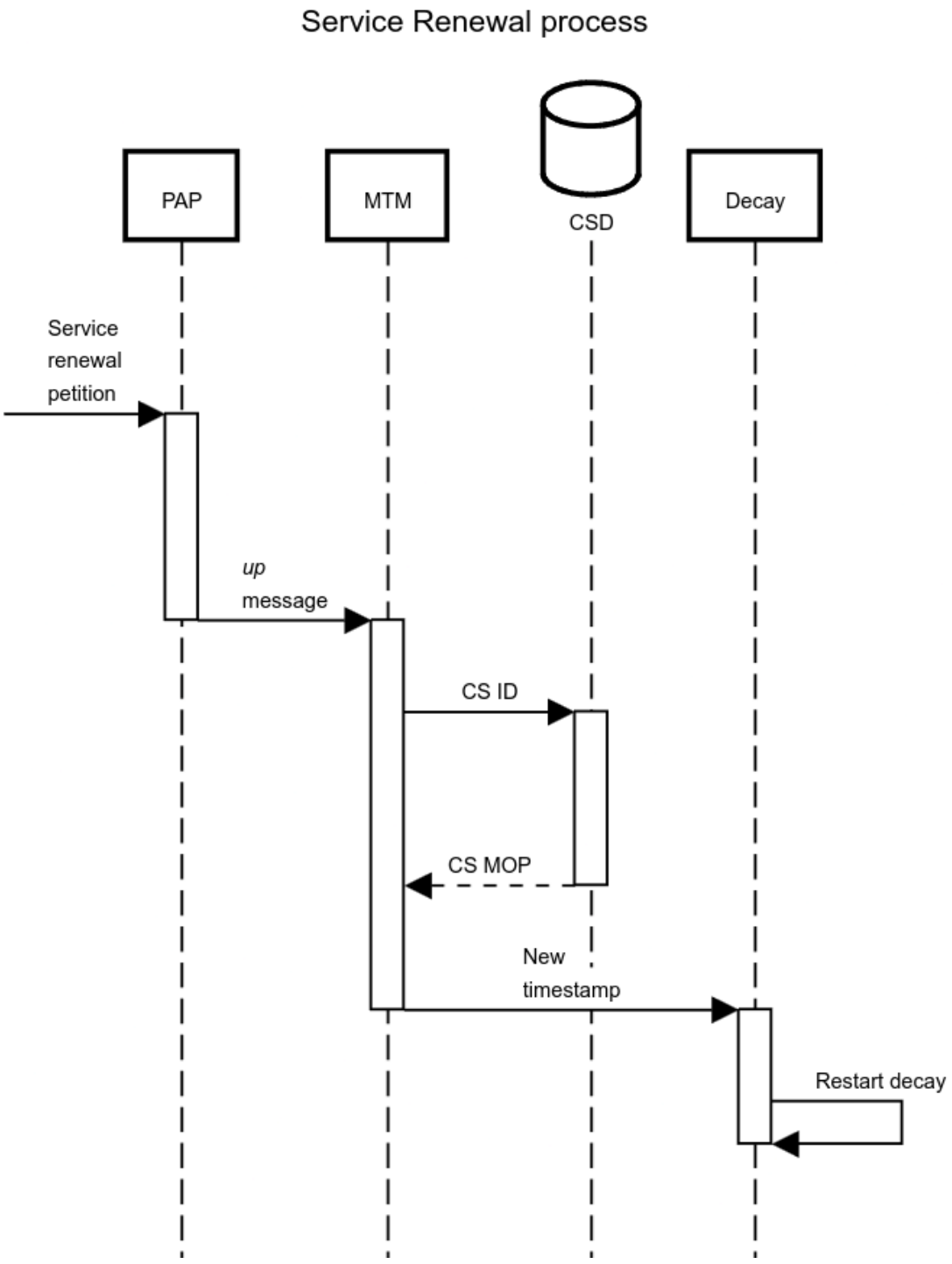


Figure 3.5: Sequence diagram depicting the renewal of a service registry triggered by a provider message.

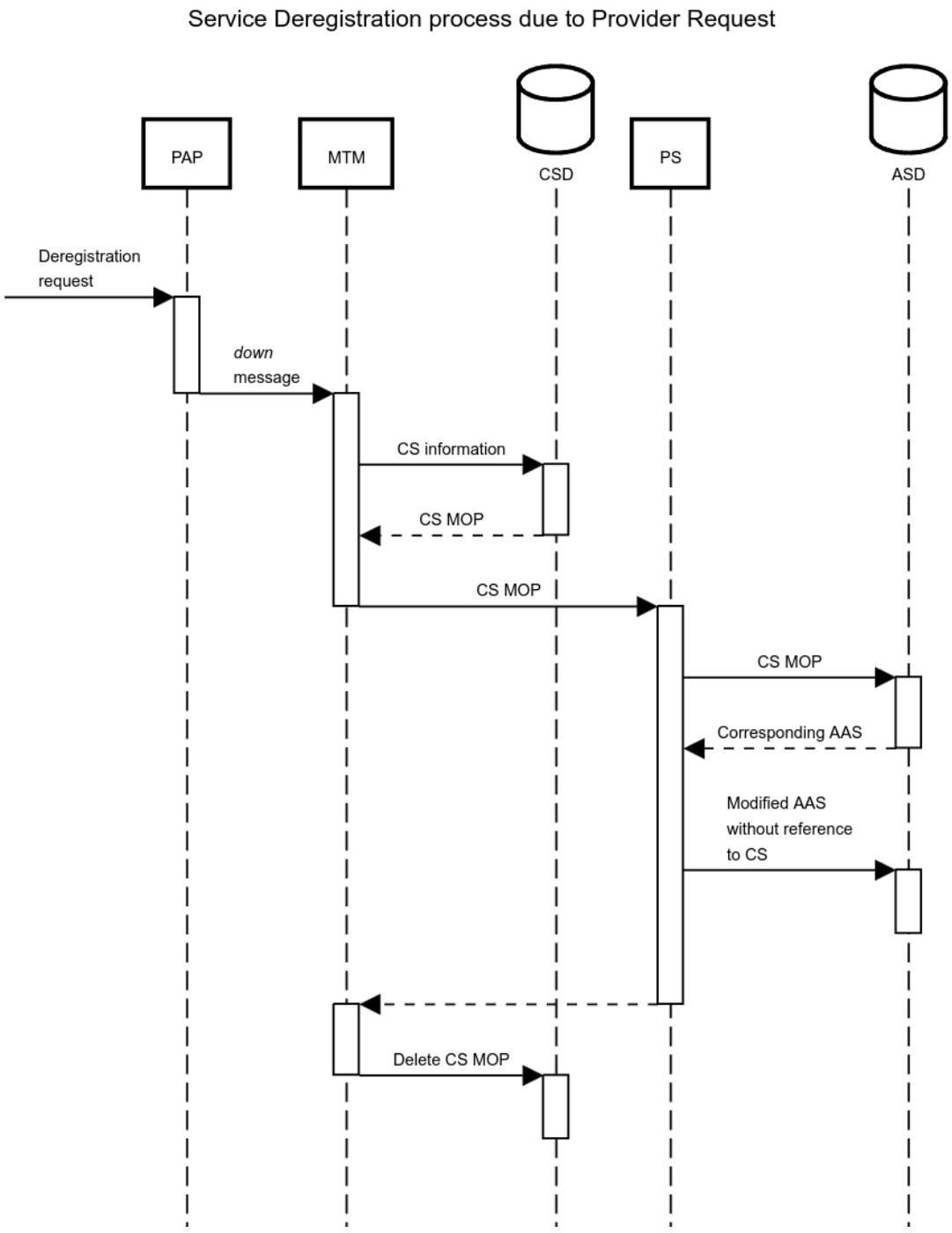


Figure 3.6: Sequence diagram depicting the service deregistration process due to the service provider requesting it.

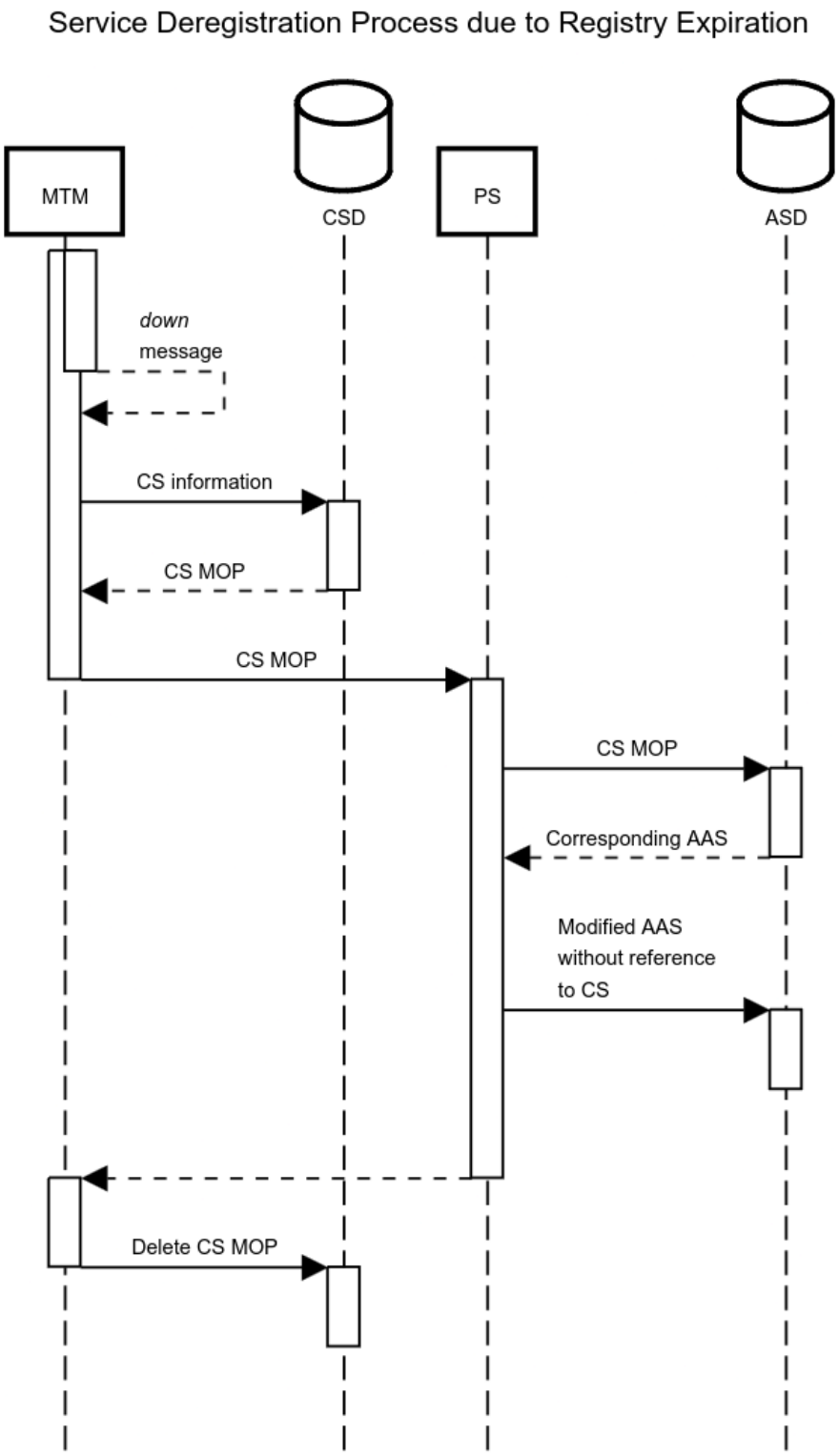


Figure 3.7: Sequence diagram depicting the service deregistration process due to the service registry crossing the expiration threshold.

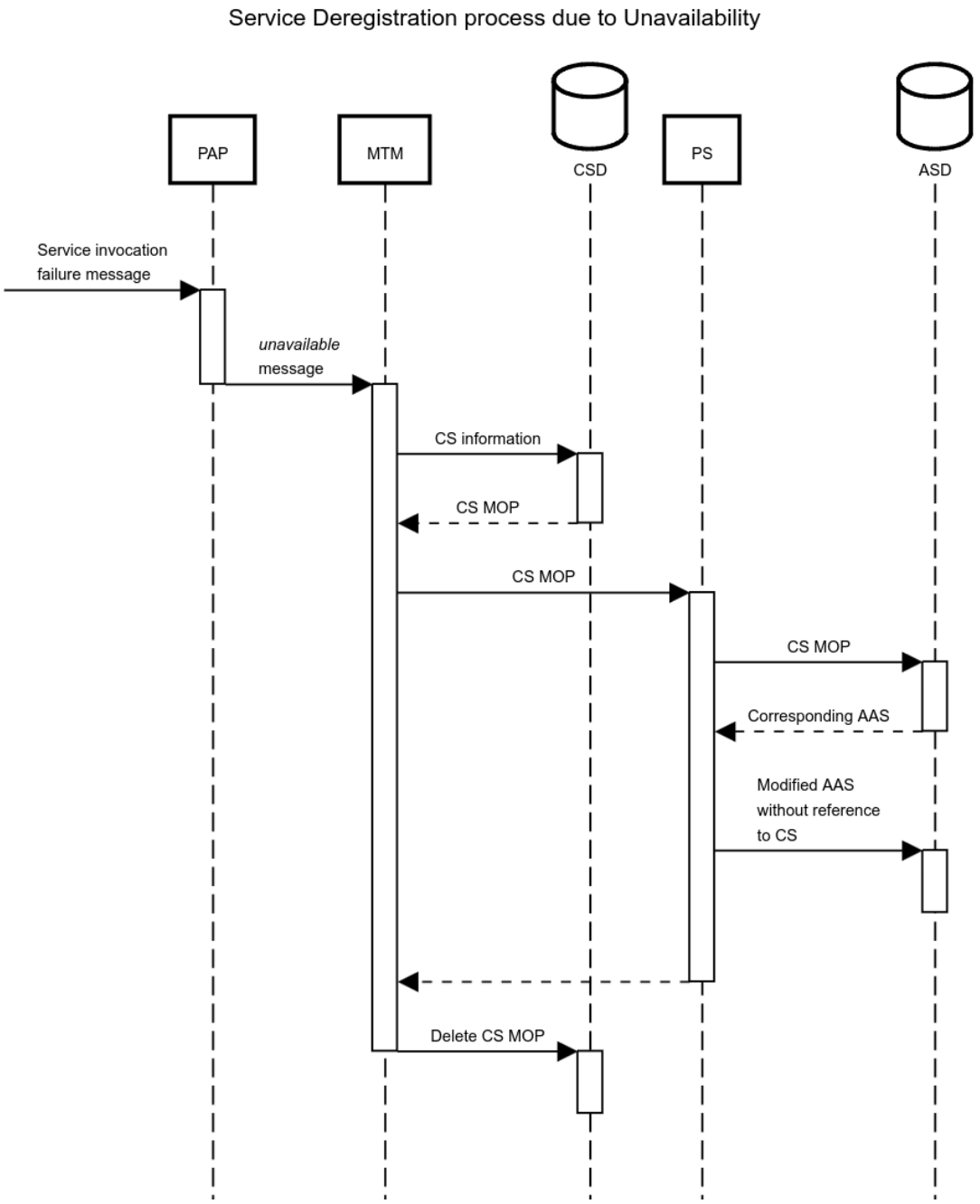


Figure 3.8: Sequence diagram depicting the service deregistration process due to the service being unable to be successfully executed during a service composition.

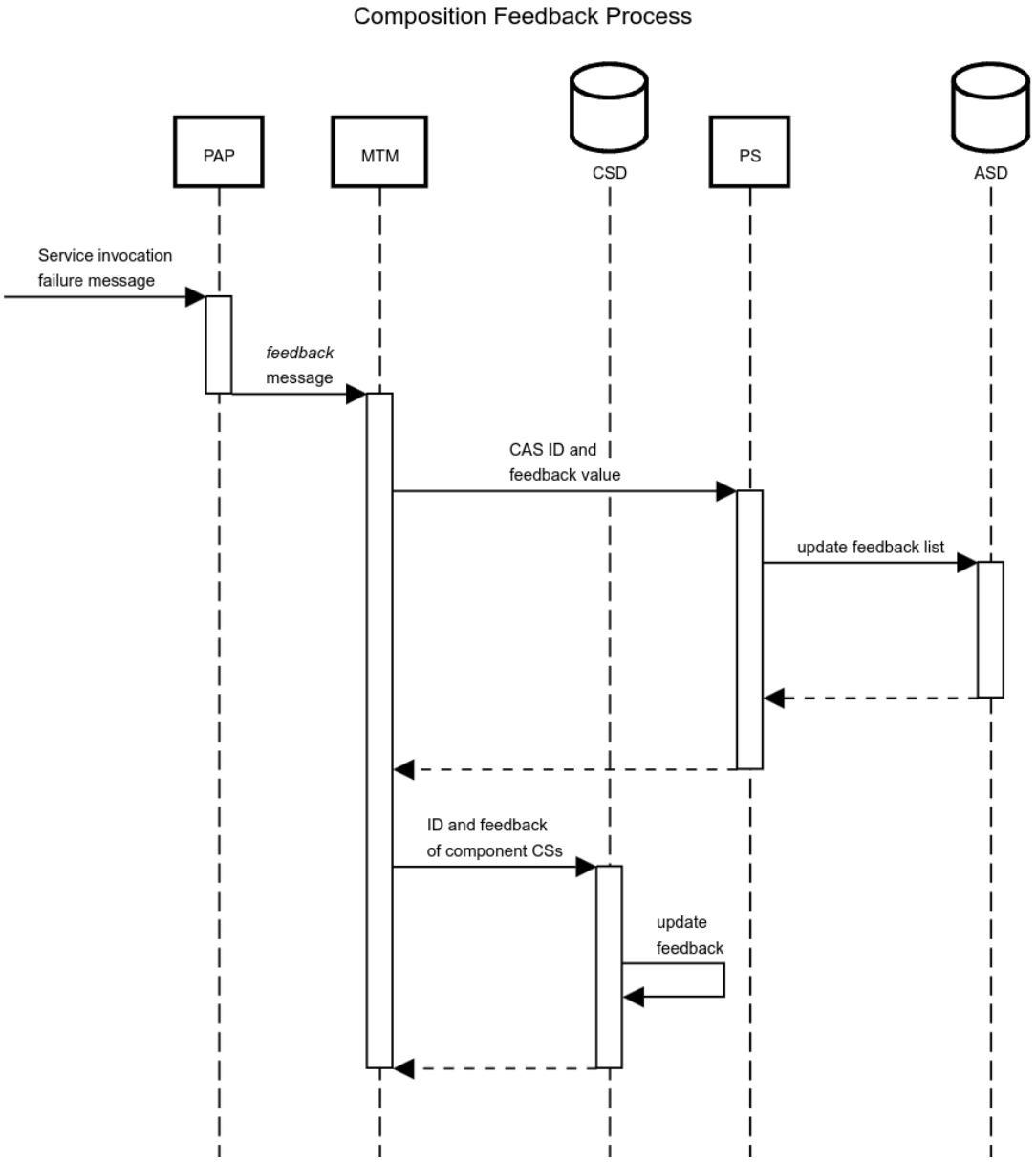


Figure 3.9: Sequence diagram depicting the service feedback process for a composition and its components after a successful execution.

- MODIFIER takes the CAS selected by the RETRIEVER as input, and modifies it to satisfy all goals in case it doesn't satisfy them all: based on a set of predefined rules, MODIFIER would add, swap or delete AASs and steps in a CAS in order for it to become more fitting to the input set of goals.

- STORER indexes and stores both new and modified CASs in the ASD: if the system produces a CAS that didn't exist before, as a result of either the modification of a CAS or as the output of the reparation process of a failed CAS, STORER takes the new composition, indexes it and stores it in the ASD. Indexes for such CASs can be either the goals they achieve and the failures they avoid.

- REPAIRER is responsible for fixing failed compositions: if a CAS is executed successfully but the result isn't satisfactory, REPAIRER can employ a failure vocabulary and a set of predefined repair strategies to modify the composition and produce a new CAS that deals with the detected failure point(s). The new CAS is sent to the STORER, while the failure points are fed as input to the ASSIGNER.

- ASSIGNER is tasked with analyzing the CAS failure(s) and, after performing a reasoning process, which establishes which inputs and steps were responsible for the failure of the composition. In other words, it receives the step(s) where the composition failed, and reasons its way up to the steps and inputs that originated such failure(s). This information is then fed to the ANTICIPATOR.

- ANTICIPATOR uses information about previous composition failures to analyze incoming sets of goals to try to find situations that can result in a failure: if a previous composition failure was pinned on a situation produced by two specific goals, ANTICIPATOR can alert the system if two goals of the same kind are present in a new input, in order to take measures early to avoid the same failure again.

In our case, each CBP module is located in a specific Cuāyōllōtl cognitive module according to its functionality. The RETRIEVER module is located on MTM and tasked with retrieving ASs, both atomic and composite. The STORER module is to be located on PS, in charge of storing new compositions produced during the CBP process. The ASSIGNER would be located on ASTS and perform the matching of composition failures with their cause. The rest of the modules, mainly involved in composition adaptation and repair, are located on the Planning module, where according to neuroscientific evidence such tasks take place. The only CBP modules directly involved in the management and retrieval of service and composition MOPs are the RETRIEVER and the STORER, located in memory areas, with most remaining modules are mainly involved in planning, and therefore beyond the scope of our work. Figure 4 shows a diagram of the CBP-based composition process, showing in which cognitive module each process takes place.

As an addition to the preexisting steps in the CBP process, we added a "plan execution" step: after MODIFIER outputs a composition, the CBP selects CS corresponding to the specified components in the CAS, and then executes the CSs according to the composition steps. Unlike other planning cases, where resources can be assumed to be always present and never fail, pervasive services can fail to execute or become unavailable, so executing a composition and checking if all selected services worked allows us to replace unavailable/failed services and try again.

After successfully executing a CAS, we can then proceed to evaluate if the results are satisfactory: a service composition may execute flawlessly, but that doesn't mean the result accomplishes the input goals. After a composition finishes the execution, the Planning module should evaluate if the goals it set were fulfilled, and then send feedback back to the CBP process. Should the feedback be unsatisfactory, the CBP can start the plan-repairing process by sending the used composition to REPAIRER.

The difference between the plan adaptations performed by REPAIRER and simply swapping component services if a CAS plans to execute is simple: the swapping of a component just means that an individual component is not usable, and adapting to such a situation requires to either swap the failed CS by another CS of the same category (AAS) or to alter/change the composition to one that replaces specific AASs, in case there is no more available CSs under an AAS. Meanwhile, the composition repair happens when all components in

a CAS executed successfully but the composition result is not satisfactory, thus meaning that the failure wasn't caused by a failed component service, but on an unexpected interaction between goals and steps/components, requiring an entirely different type of adaptation.

It is during the execution of the CBP process that many of the service-managing and service retrieval processes happen:

- CSs are retrieved during the composition execution phase: once MODIFIER outputs a composition, Planning will select a CS from the available ones corresponding to a AAS, for every AAS used as component, and then query the CSD in order to get its information, like IP, in order to invoke it during the execution.

- If a CS fails to be invoked during the composition execution, then the Planning module sends an `unavailable` signal to MTM to de-register service from the CSD.

- After a CAS is executed successfully, a `feedback` will be sent to the MTM containing the feedback for the CAS and its corresponding components.

In this chapter, we defined which Cuāyōllōtl cognitive modules are involved in our proposal, the messages used to manage service registries in our directory, which processes are triggered by those messages and how they are carried our, and the memory structure that we use to store information in our directory, and also laid out a vision of a service composition and adaptation process of which our directory would be part of. With our proposal now defined, we now move towards defining the implementation details of our proposal.

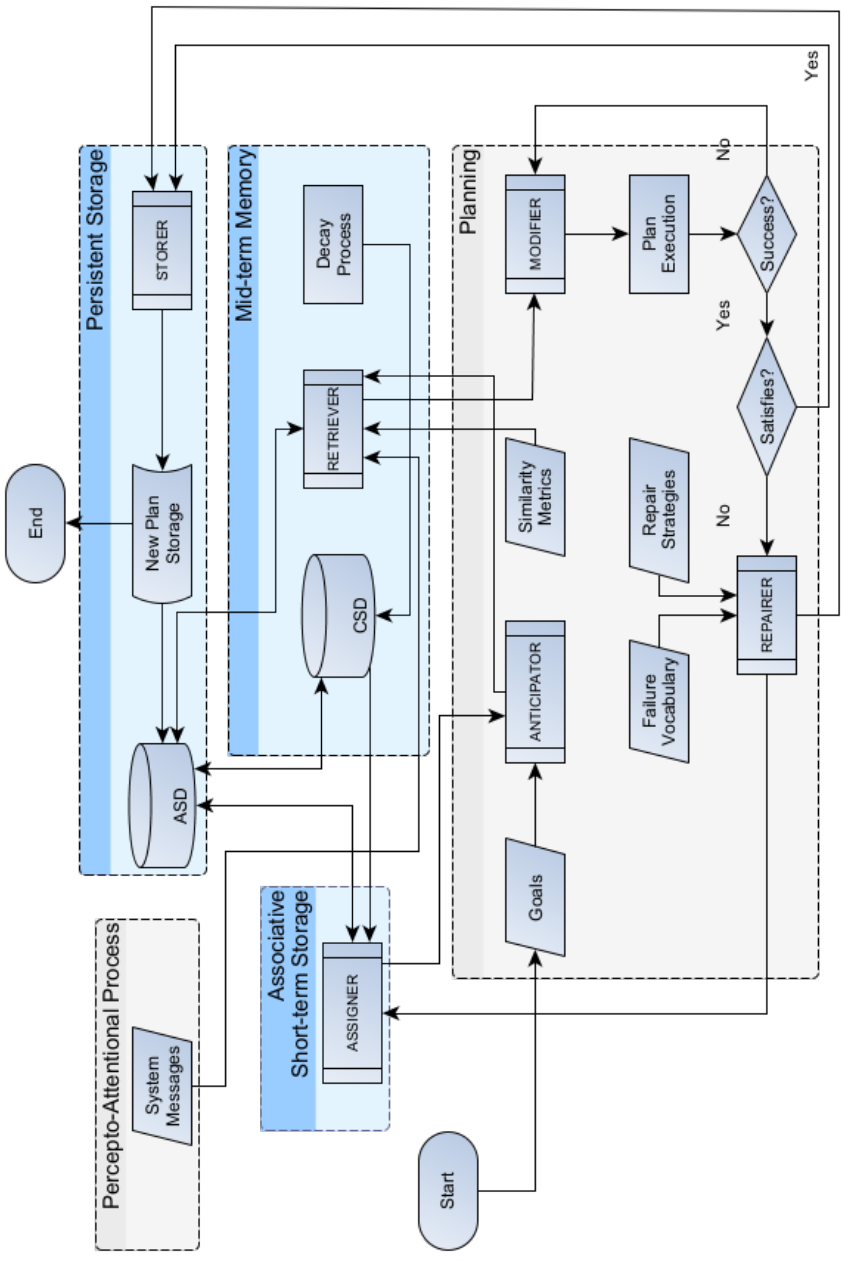


Figure 3.10: Diagram showing the Cuāyōllōtl-based CBP-based service composition process. Blue blocks represent the cognitive modules where directory tasks take place, and gray blocks stand for cognitive modules apart from the directories.

4

Implementation

In order to test that our proposal in a more meaningful way, we developed an implementation of it on Common Lisp, based on the Micro MOP implementation of Riesbeck (RIESBECK; SCHANK, 1989). W chose to implement our proposal in Common Lisp because the language has a good performance, a big set of libraries to quickly and easily add functionalities like socket communication and XML parsing, and because it has the most development and documentation for MOP-based tools and projects.

In this chapter, we proceed to delve into the specifics of the implementation of our proposal, like the structure of the proposed messages and the algorithms employed in registry management.

4.1 Message and MOP structure

We implemented our service directory such that all communications are performed through TCP messages formatted on XML. Our implementation takes as input the IP direction and socket it will listen to for messages, as well as, optionally, the size in seconds of the threshold for the expiration of service registries in the CSR: if not specified, the CSR uses the default value. Listings 4.1, 4.2, 4.3, 4.4 and 4.5 show the format for the up, timeout, down, unavailable and feedback messages, respectively.

```
<message name="up">
  <name>Foobar</name>
  <dir>127.0.0.1</dir>
  <port>2222</port>
  <attributes>
    <latency>value</latency>
    <payload>value</payload>
    <available-energy>value</available-energy>
    <response-time>value</response-time>
    <timestamp>value</timestamp>
  </attributes>
</message>
```

Code 4.1: XML structure for an up message

```
<message name="down">
  <service>
    <name>I-M-Foobar</name>
  </service>
</message>
```

Code 4.2: XML structure for a down message

```
<message name="timeout">
  <service>
    <name>I-M-Foobar</name>
  </service>
</message>
```

Code 4.3: XML structure for a timeout message

```
<message name="unavailable">
  <service>
    <name>I-M-Foobar</name>
  </service>
</message>
```

Code 4.4: XML structure for an unavailable message

```
<message name="feedback">
  <composition>
    <name>I-M-Foobar-Composition</name>
    <feedback>value</feedback>
  </composition>
  <component>
    <name>I-M-Foobar-Component-1</name>
    <feedback>value</feedback>
  </component>
  <component>
    <name>I-M-Foobar-Component-2</name>
    <feedback>value</feedback>
  </component>
  ...
</message>
```

Code 4.5: XML structure for a feedback message

The expression in Common Lisp for MOPs is shown is Listing 4.6. Every MOP has a type, which can be either instance for MOPs that have no specializations or mop for every other MOP. Also, every MOP has an unique name, with names for mop-type MOPs starting with M- (e.g. M-composition) while names for instance-type MOPs starting with I-M- (e.g. I-M-Foobar-Composition). The last necessary element for a MOP are its abstractions, represented as a list with the names of all abstractions of a MOP. Optionally, a MOP can have a list of slots, represented by a list of key-value pairs (slot filler), where slot stands for the name of the slot, and filler can be either the MOP it points at or nil (null).

```
(type name (abstractions)
  ((slot1 filler1) (slot2 filler2) ... ))
```

Code 4.6: Expression for a MOP structure in Common Lisp

The system uses three tables to store information about outgoing links for every MOP in the memory structure:

- A table named mop-absts holds a list of every abstraction of a MOP, from direct abstraction all the way to M-Root, e.g. the entry for the MOP M-Atomic would be (M-Abstract-S, M-Root). This table effectively holds the outgoing abstraction links of every MOP.

- A table named mop-specs that has the links of every direct specialization of a MOP, e.g. the entry for M-Abstract-S would be (M-Atomic, M-Composition). This table basically holds simplified versions of outgoing indexing links of every MOP.
- A table named mop-slots which holds a list of slots of a MOP, e.g. the entry for M-Composition would be ((steps M-Step-Group) (components M-Abstract-Group))

4.2 Directory Functions

Algorithm 1 depicts the simple process of adding a new CS registry to the directory: simply creating a MOP standing for the new CS and then creating slots for that MOP corresponding to the MOPs representing the values of the attributes.

Algorithm 1 Adding a new CS to the directory

attribute list A
CS MOP s
$s \leftarrow$ create new MOP with A as slots
a in A such that $a \neq$ nil
create slot x for attribute $a.key$ on MOP s
$s.x.value \leftarrow a.value$

Algorithm 2 takes as input an AAS MOP, and optionally a list of (attribute, value) pairs, and produces a list of available CS MOPs of the same type of the AAS, and matching the same attributes that are provided. The algorithm first goes and collects all services linked to the AAS MOP, and then checks if a list of attributes was provided: if it was, it collects all services that match the attributes in the list and are available into a list; otherwise it collects all available services into the new list.

Algorithm 3 shows the process of removing a CS registry from the directory. First, it searches for any reference of the CS MOP on the corresponding instance of M-Concrete-Group and deletes it, then it proceeds to delete the MOP for the CS's timestamp, and finally it deletes the CS MOP from the CSD.

Algorithm 2 CS search algorithm

AAS MOP a, attribute list A
CS MOP set L
$g \leftarrow$ filler for the available slot in a
$G \leftarrow$ slot list of g
$L \leftarrow \emptyset$
$A = \emptyset$
s in G

if s.status=free **then**
$L \leftarrow L \cup s$
s in G
x in A
$f \leftarrow$ true

if s doesn't have attribute x or $x.value \neq s.x.value$ **then**
$f \leftarrow$ false

if f = true and s.status=free **then**
$L \leftarrow L \cup s$

Algorithm 3 CS removal algorithm

CS MOP s
$g \leftarrow$ AAS corresponding to s
$G \leftarrow$ slot list of g
$G \leftarrow G$ without the slot containing s
$t \leftarrow$filler for the timestamp slot in s
$f \leftarrow$ expiration function for s
unschedule (f)
remove MOP t
remove MOP s

Algorithm 4 depicts process for searching compositions that achieve a specified goal and, optionally, employ a specific set of ASs.

Algorithm 4 CAS search algorithm

goal g, component list C
CAS MOP set R
$X \leftarrow$ specializations of M − Composition
$R \leftarrow \emptyset$
$C = \emptyset$
x in X

if $index.goal$ (M − Composition, x) $= g$ **then**
$R \leftarrow R \cup x$
x in X

if $index.goal$ (M − Composition, x) $= g$ **then**
$S \leftarrow$ components of x
$f \leftarrow$ true
$c \in C$

    if $c \notin S$ **then**
$f \leftarrow$ false
break

    if $f =$ true **then**
$R \leftarrow R \cup x$

In order to implement the service relevance mechanism, we used the memory decay function employed by Cuāyōllōtl's MTM (CASTILLO, 2020), which was proposed by Altmann and Schunn (ALTMANN; SCHUNN, 2012):

$$A(x) = -0.5 \ln(x) \tag{4.1}$$

where $A(x)$ stands for the degree of activation of an item in MTM at a time x, where in our case x stands for the difference between the timestamp of a service registry and the current time. For our work, $A(x)$ stands for the relevance of the corresponding service registry at time x.

The time threshold e used to signal that a service registry has expired is received as an optional parameter at the start of the execution. If the user didn't enter a value for the expiration threshold, the default value used is 900 seconds (15 minutes), one of the lower time bounds MTM can retain items (**??**). When a CS is registered and its corresponding timestamp MOP is created, a directory function is timed to be executed after e seconds have passed and sends a timeout

signal about the related CS. But if a CS registry is renewed through an up signal, the timer is restarted and the current CS timestamp is replaced with another timestamp with the system time at the moment of the renewal. Algorithm 5 shows the process of renewing a CS MOP registry by resetting the expiration function and replacing its timestamp.

Algorithm 5 Service registry renewal process

service MOP s, expiration threshold e
$t \leftarrow$ MOP in the timestamp slot of s
$f \leftarrow$ expiration function for s
unschedule (f)
create timestamp MOP t_n for the current system time
schedule (f, e)
swap t with t_n in the timestamp slot of s
delete t

5

Case Study

With a service directory fully implemented, we chose a predatory crime deterrence case study to test the behavior of our proposal, which is a common issue in urban areas. We chose this case study given not only its prevalence in urban areas, but also because, given the environment the a crime deterrence system may work in, a way to manage devices and services offered by them has to be able to deal with a great degree of dynamism regarding the availability of services in it, as well as a great variety of service attributes.

5.1 Introduction

We call predatory crimes to those that are premeditated, in which a victim's property is taken, and in which the criminal stalks the victim until an opportunity to attack arises, behaving similarly to a predator hunting a prey (MCELLISTREM, 2004). Currently, there are some documented cases of drones being employed to deter criminals, as well as for assisting law enforcement agencies to capture criminals. As an example, in (**??**) it was reported about how in the Mexican city of Ensenada, a single drone employed by the city police department helped reduce the general crime rate by 10%, and assisted in the arrest of over 500 individuals. This leads us to believe that a pervasive system composed by adequate kinds of devices can successfully be employed to deter potential criminals and capture those detected committing crimes, and a ser-

vice directory with the capability to quickly adapt to changes in the state of the system's devices would be of great help.

For the task of deterring potential criminals or assisting on their capture by law enforcement officials, we consider the system operation to be composed by three phases:

- Monitoring: it consists in sensing the environment, looking for either crimes being committed in this moment or for suspicious situations that can be considered the preamble of a crime. In this phase, sensor devices like cameras or microphones are to be employed to monitor the environment, and processing devices would analyze the sensor information in search of unusual situations.

- Deterrence: it consists in driving a drone to the potential crime scene. If during the monitoring phase a suspicious situation is detected, e.g. a passerby closely following another person, the system transits to a deterrence phase, in which a drone is dispatched to the place where the unusual situation is taking place, with the goal of deterring and driving the potential criminal away.

- Alarm: it consists of activating actuators near a crime scene in order to draw the attention of nearby law enforcement agents towards it. If during either the monitoring or deterrence phases a crime is detected to be taking place, the system would switch to this phase, with the goal of using any close device capable of using actuators to draw attention towards it, e.g. speakers or alarms. The intention behind this is to draw the attention of any law enforcement unit close enough to the crime scene, or even prevent any further action by the criminal and causing the criminal to flee.

For this case study, we assume the following:

- for the sake of simplicity, every CAS is composed by just three steps each: one for each one of the aforementioned phases.

- no new compositions will be stored: we test the retrieval of existing compositions, but not the storage of new ones.
- the set of service categories is predefined as instances of M-Atomic and doesn't change during the system execution: no new categories are created nor deleted.
- the service classification process is a black box: once a new service is registered, ASTS randomly indexes it under one of the instances of M-Atomic.
- for the sake of simplicity, all ASs employed by compositions are AASs: CASs using other CASs as components is not allowed.

5.2 Additional Memory Structures

In order to adapt our memory structure for its employment in this study case by adding the following MOPs:

- M-Public-Sec-Act: specialization of M-Act. It stands for actions related to public security, e.g.
- M-Monitoring-Act: specialization of M-Public-Sec-Act. It represents actions related to the monitoring phase of the system, e.g. watching the environment through cameras.
- M-Deterrence-Act: specialization of M-Public-Sec-Act. It stands for actions related to the system's deterrence phase, e.g. the drone propelling itself to a new location.
- M-Alarm-Act: specialization of M-Public-Sec-Act that represents actions carried out during the system's alarm phase, e.g. an alarm activating itself.

- M-Public-Sec-Step: specialization of M-Step that represents steps of a service composition for a public security task, e.g. watching over an area or ringing an alarm. It inherits the action packaging link from M-Step, which links it to M-Public-Sec-Act, to represent that every public security step is performed by a public security action.

- M-Monitoring-Step: specialization of M-Public-Sec-Step that stands for service composition steps performed during the system's monitoring phase, e.g. visually monitoring the environment. It's linked to M-Monitoring-Act though the packaging link action which represents which action performs the step.

- M-Deterrence-Step: specialization of M-Public-Sec-Step representing service composition steps performed during the system's deterrence phase, e.g. translating the drone to a specific place. It's linked to M-Deterrence-Act though the packaging link action, representing which action performs the step.

- M-Alarm-Step: specialization of M-Public-Sec-Step that stands for service composition steps performed during the system's alarm phase, e.g. drawing attention to the crime scene. It's linked to M-Alarm-Act though the packaging link action which represents which action performs the step.

- M-Public-Sec-Steps: specialization of M-Root representing the set of monitoring, deterrence and alarm steps of a composition. It has three packaging links: one named monitoring-steps, other named deterrence-steps, and another one named alarm-steps, with all three linked to M-Step-Group: monitoring-steps represents the link to the group of steps to be performed during the monitoring phase of a composition, deterrence-steps stands for for the link with the steps performed during the deterrence phase, and alarm-steps represents the step group for the alarm phase. It follows that, instead of being linked directly to M-Step-Group, M-Composition is linked

to this MOP through the packaging link steps, representing that every composition has three different sets of steps, with each set corresponding to a system phase.

Listing 5.1 shows the Lisp code for the structure of a CAS MOP, while Listing 5.2 shows the structure for M-Public-Sec-Steps, whose instances holds the step groups for every CASs.

```
(mop m-composition (m-abstract-s)
  (monitoring-service m-atomic)
  (deterrence-service m-atomic)
  (alarm-service m-atomic)
  (steps m-public-sec-steps))
```

Code 5.1: Lisp expression for the MOP structure of a CAS registry in the case study

```
(mop m-public-sec-steps (m-root)
  (monitoring-steps m-step-group)
  (deterrence-steps m-step-group)
  (alarm-steps m-step-group))
```

Code 5.2: Lisp expression for the MOP holding the different steps in a CAS in a composition

5.3 Experiments

With our case study fully specified, we proceed to test the implementation of our directory under the aforementioned case study. In this scenario, we chose six service types (AASs) to be used: alarm services, microphone services, speaker services, camera services, infrared camera services and drone services.

For our testing, we decided to generate random services to be registered in the directory: a client node generates Common Lisp expressions with the information necessary to create a CS MOP: a name, a timestamp and a random number of attributes, with such attributes take a random value, e.g. a service foobar can be generated with attributes response-time with value short and available-energy with value low. Then, the client parses the expression to XML and sends it to the directory node, which parses the XML message and uses the expression to

generate a new instance of M-Concrete-S. Such process takes place every in a random interval between 2 to 5 minutes.

After registering the first service, every 4 to 7 minutes, we also simulate the search and retrieval of a CAS: the directory receives a request to find a CAS MOP that fulfills certain requirements, e.g. accomplishes certain goal or employs some specific component AASs. After a set of CASs has been retrieved, we simulate the retrieval of the CSs for the execution: for every component AAS, we simulate a query for a corresponding CS: that query can include specific attributes to simulate the search of services that fulfill a certain attribute.

In order to simulate unavailability rates for services, we compute a probability in a way that linearly increases over time in order to simulate the growth over time of the probability for a device to become unavailable: this is because, as time goes, a device traversing the environment like a smartphone or a drone may move away from the environment, and devices with limited energy resources (e.g. devices that employ battery as power sources) may become unavailable due to low energy. This probability is computed every time a CS is fetched as a component for a CAS.

To review the performance of our proposal, we conduct two tests:

- in the first test, we test the effect different expiration thresholds and service renewal probabilities in the availability of registered CSs, with the goal of testing how different service availability times and service expiration thresholds affect the number of failures our proposal has during its operation.

- for the second test, we check how the employment the relevance mechanism as a criterion to select component CSs affects the selection behavior.

In the first test, we test expiration threshold of 15, 30 and 60 minutes, after which CS registries are deleted. We also employ three different probabilities that a service provider sends a message to renew the registry for the service it hosts: 33%, 50% and 67%. This leads us to having 9 different scenarios, with each one being tested 3 times each. In this test, a CAS is retrieved from the ASD, and after that, for every component AAS from that CAS, a CS is fetched. Based on the time lapsed since its registration or last renewal, a probability is computed

Run	Expiration threshold	Number of CS retrievals	Number of unavailabilities	Failure rate
1	15 min.	25	10	40%
2	15 min.	26	12	46.2%
3	15 min.	25	11	44%
4	15 min.	22	8	36.4%
5	30 min.	25	18	72%
6	30 min.	22	11	50%
7	30 min.	25	15	60%
8	30 min.	26	19	73.1%
9	60 min.	24	19	79.2%
10	60 min.	25	20	80%
11	60 min.	22	16	72.7%
12	60 min.	25	20	80%
13	N/A	25	20	80%
14	N/A	25	21	84%
15	N/A	25	21	84%
16	N/A	25	14	56%

Table 5.1: Results of the first experiment with an availability of 15 minutes and a service renewal chance of 50%

for every component CS using a Bernoulli trial where the probability of success (a CS being invoked without failures) is $p = 1 - \frac{t_c - t_r}{a}$, where t_c is the system's current time, t_r is the timestamp in the CS's registry, and a is the availability time of device in the environment, meaning that $p = 1$ at the moment of a CS registration or renewal and $p = 0$ upon reaching the maximum time a device may be available. The values a can take are 15, 30 and 60 minutes, used because those are close to the availability times for drones as watched in empirical evidence. If a service is deemed unavailable, an unavailable message is sent to deregister that service, and that invocation is counted as a failure. After 180 minutes, we stop the execution and get the rate of failure for CSs. This process is repeated for each one of the scenarios, with the goal of seeing the directory's behavior and the failure rates. Figure 5.1 shows the comparison between the failure rates under the different scenarios.

For the second test, we test the behavior of CS selection under two different scenarios: selection based purely on feedback stored in the CSD and selection based on a rating computed from both stored feedback and the relevance, com-

Run	Expiration threshold	Number of CS retrievals	Number of unavailabilities	Failure rate
1	15 min.	24	7	29.2%
2	15 min.	25	10	40%
3	15 min.	25	9	36%
4	15 min.	25	10	40%
5	30 min.	25	16	64%
6	30 min.	24	11	45.8%
7	30 min.	25	13	52%
8	30 min.	24	17	70.8%
9	60 min.	25	17	68%
10	60 min.	17	13	76.5%
11	60 min.	25	17	68%
12	60 min.	25	16	64%
13	N/A	20	12	60%
14	N/A	25	20	80%
15	N/A	26	22	84.6%
16	N/A	23	12	52.2%

Table 5.2: Results of the first experiment with an availability of 15 minutes and a service renewal chance of 67%

Run	Expiration threshold	Number of CS retrievals	Number of unavailabilities	Failure rate
1	15 min.	25	13	52%
2	15 min.	25	10	40%
3	15 min.	25	18	72%
4	15 min.	25	11	44%
5	30 min.	26	13	50%
6	30 min.	25	18	72%
7	30 min.	25	13	52%
8	30 min.	25	18	72%
9	60 min.	26	22	84.6%
10	60 min.	25	20	80%
11	60 min.	25	16	64%
12	60 min.	25	15	60%
13	N/A	26	17	65.4%
14	N/A	25	20	80%
15	N/A	25	18	72%
16	N/A	25	20	80%

Table 5.3: Results of the first experiment with an availability of 15 minutes and a service renewal chance of 33%

Run	Expiration threshold	Number of CS retrievals	Number of unavailabilities	Failure rate
1	15 min.	25	7	28%
2	15 min.	25	5	20%
3	15 min.	25	7	28%
4	15 min.	25	5	20%
5	30 min.	25	11	44%
6	30 min.	25	10	40%
7	30 min.	26	8	30.8%
8	30 min.	25	12	48%
9	60 min.	25	15	60%
10	60 min.	25	13	52%
11	60 min.	26	17	65.4%
12	60 min.	25	16	64%
13	N/A	25	19	76%
14	N/A	25	16	64%
15	N/A	25	19	76%
16	N/A	25	17	68%

Table 5.4: Results of the first experiment with an availability of 30 minutes and a service renewal chance of 50%

Run	Expiration threshold	Number of CS retrievals	Number of unavailabilities	Failure rate
1	15 min.	26	4	15.4%
2	15 min.	25	7	28%
3	15 min.	26	1	3.8%
4	15 min.	25	4	16%
5	30 min.	24	6	25%
6	30 min.	25	10	40%
7	30 min.	25	12	48%
8	30 min.	24	5	20.8%
9	60 min.	21	11	52.4%
10	60 min.	25	10	40%
11	60 min.	25	12	48%
12	60 min.	25	17	68%
13	N/A	25	9	36%
14	N/A	25	16	64%
15	N/A	26	14	53.8%
16	N/A	26	17	65.4%

Table 5.5: Results of the first experiment with an availability of 30 minutes and a service renewal chance of 67%

Run	Expiration threshold	Number of CS retrievals	Number of unavailabilities	Failure rate
1	15 min.	25	6	24%
2	15 min.	24	6	25%
3	15 min.	25	4	16%
4	15 min.	26	7	26.9%
5	30 min.	22	11	50%
6	30 min.	25	14	56%
7	30 min.	25	8	32%
8	30 min.	25	13	52%
9	60 min.	25	13	52%
10	60 min.	25	12	48%
11	60 min.	25	15	60%
12	60 min.	25	16	64%
13	N/A	25	21	84%
14	N/A	25	19	76%
15	N/A	25	18	72%
16	N/A	25	16	64%

Table 5.6: Results of the first experiment with an availability of 30 minutes and a service renewal chance of 33%

Run	Expiration threshold	Number of CS retrievals	Number of unavailabilities	Failure rate
1	15 min.	25	5	20%
2	15 min.	26	2	7.7%
3	15 min.	25	4	16%
4	15 min.	25	0	0%
5	30 min.	23	5	21.7%
6	30 min.	25	8	32%
7	30 min.	25	4	16%
8	30 min.	24	6	25%
9	60 min.	25	7	28%
10	60 min.	25	8	32%
11	60 min.	25	9	36%
12	60 min.	25	7	28%
13	N/A	25	13	52%
14	N/A	25	10	40%
15	N/A	25	8	32%
16	N/A	25	13	52%

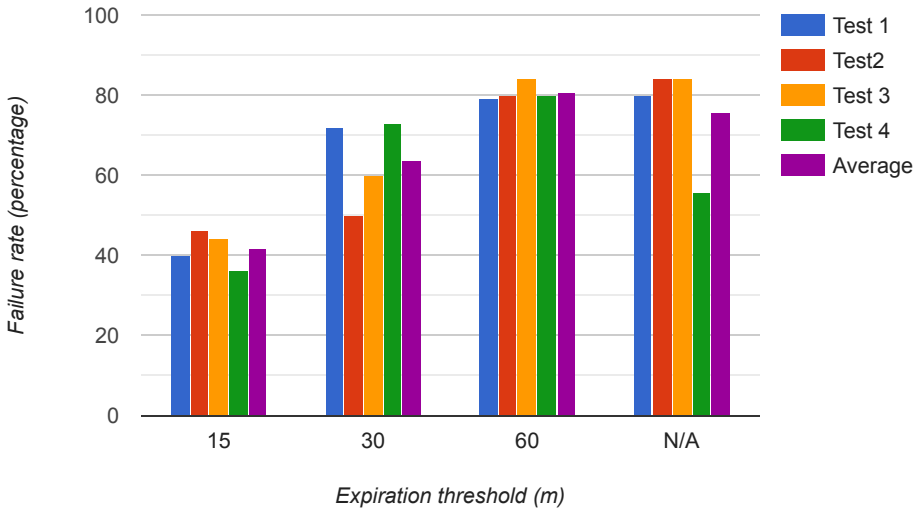
Table 5.7: Results of the first experiment with an availability of 60 minutes and a service renewal chance of 50%

Run	Expiration threshold	Number of CS retrievals	Number of unavailabilities	Failure rate
1	15 min.	25	3	12%
2	15 min.	25	2	8%
3	15 min.	23	4	17.4%
4	15 min.	25	4	16%
5	30 min.	25	7	28%
6	30 min.	25	4	16%
7	30 min.	22	7	31.8%
8	30 min.	26	4	15.4%
9	60 min.	21	9	42.9%
10	60 min.	22	8	36.4%
11	60 min.	25	9	36%
12	60 min.	24	11	45.8%
13	N/A	25	14	56%
14	N/A	25	16	64%
15	N/A	25	7	28%
16	N/A	23	10	43.5%

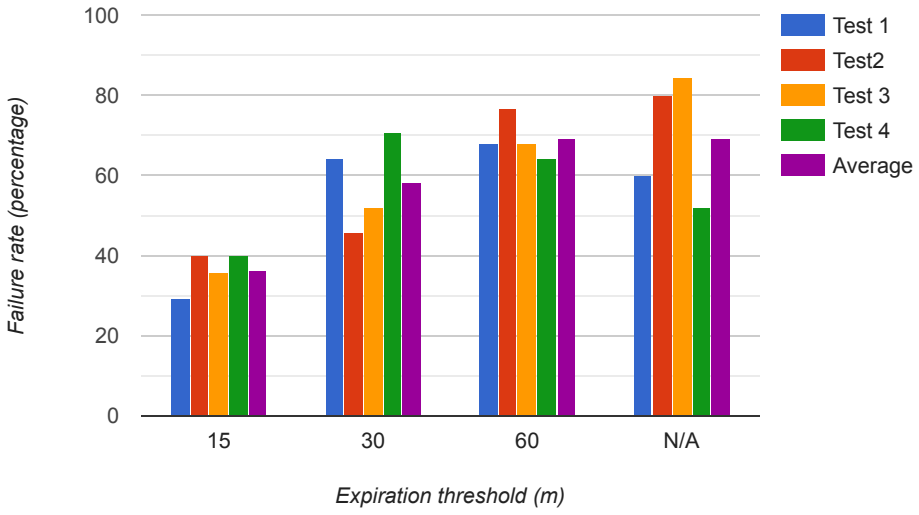
Table 5.8: Results of the first experiment with an availability of 60 minutes and a service renewal chance of 67%

Run	Expiration threshold	Number of CS retrievals	Number of unavailabilities	Failure rate
1	15 min.	25	2	8%
2	15 min.	25	4	16%
3	15 min.	25	3	12%
4	15 min.	25	4	16%
5	30 min.	24	2	8.3%
6	30 min.	26	6	23.1%
7	30 min.	25	7	28%
8	30 min.	24	7	29.2%
9	60 min.	25	8	32%
10	60 min.	25	9	36%
11	60 min.	26	11	42.3%
12	60 min.	25	14	56%
13	N/A	25	12	48%
14	N/A	25	9	36%
15	N/A	25	14	56%
16	N/A	25	15	60%

Table 5.9: Results of the first experiment with an availability of 60 minutes and a service renewal chance of 33%

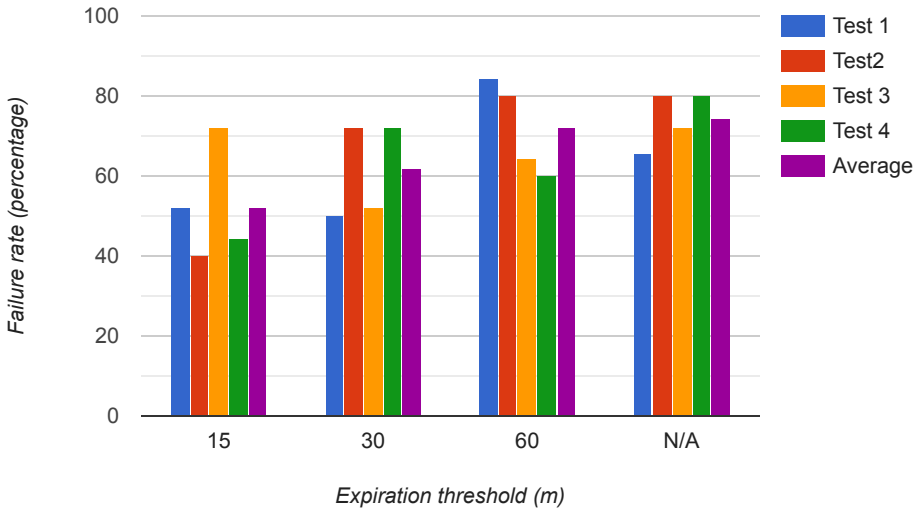


(a) Comparison of results for the scenario with a renewal probability of 50%



(b) Comparison of results for the scenario with a renewal probability of 67%

Figure 5.1: Comparison of the failure rates for the scenarios of the first experiment where availability time of devices is set to 15 minutes



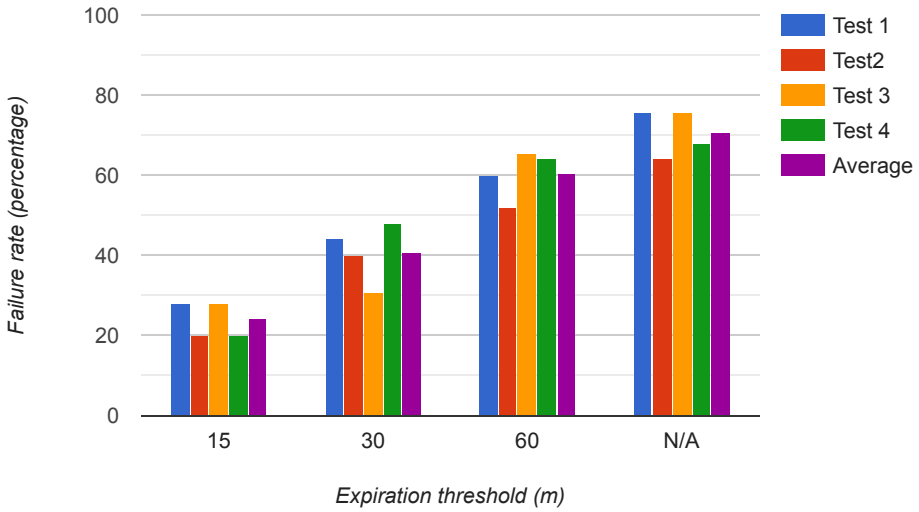
(c) Comparison of results for the scenario with a renewal probability of 33%

Figure 5.1: Comparison of the failure rates for the scenarios of the first experiment where availability time of devices is set to 15 minutes (cont)

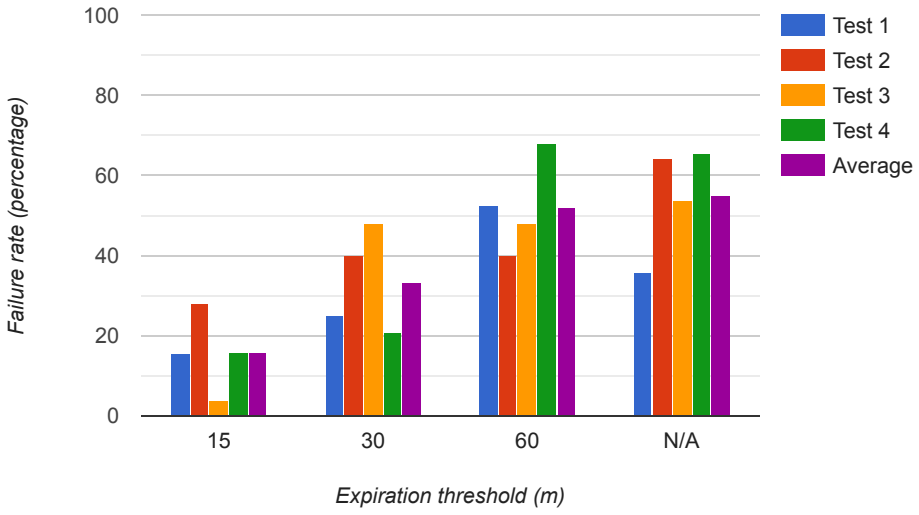
puted with the following equation:

$$w = xf + yr \tag{5.1}$$

where w is the final value of a service, f is the average of feedback values of a CS, r is the relevance value of a CS registry, and $x, y \in [0, 1]$ are weights for the two features to evaluate, with $x + y = 1$. For this, we run the directory and generate service registration/renewal messages and CAS queries at random intervals of four to six minutes, and then fetch the corresponding component CSs, after which the selected CSs receive a random feedback value between $\frac{0}{100}$ and $\frac{100}{100}$, picked using an uniform distribution. For the first scenario, every time we query the available CSs for a specific AAS, we select the one with the highest feedback average, and in the second case, the selection is based in the weighting between feedback average and the computed relevance. We test four cases: one where the the selection is based purely on feedback, one with a weighting of 50% for the feedback and 50% for the relevance, one with a weighting of 67% for the feedback and 33% for the relevance, and one with a weighting of 67% for the relevance and 33% for the feedback. In this test, we measure the average service

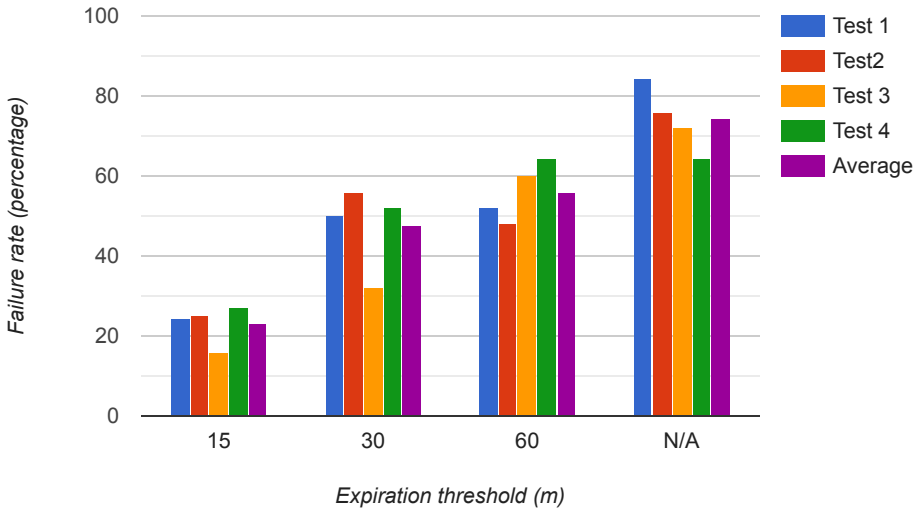


(a) Comparison of results for the scenario with a renewal probability of 50%



(b) Comparison of results for the scenario with a renewal probability of 67%

Figure 5.2: Comparison of the failure rates for the scenarios of the first experiment where availability time of devices is set to 30 minutes



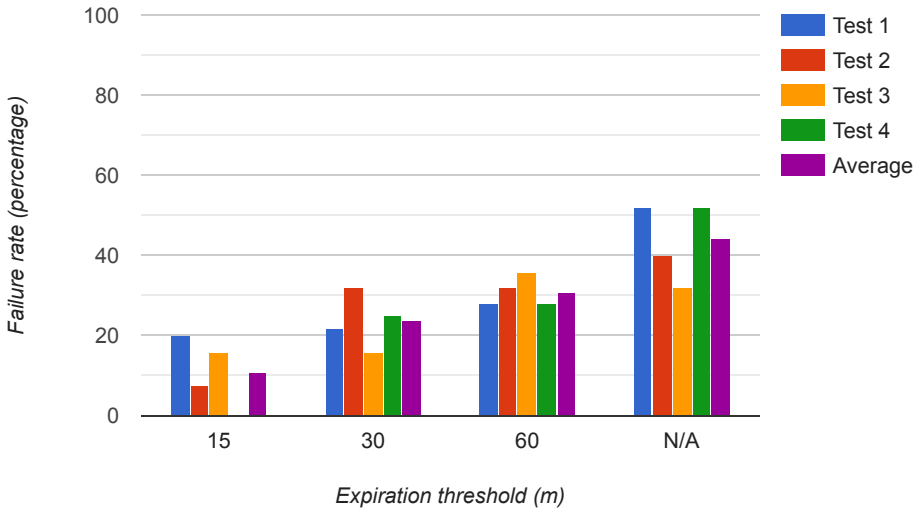
(c) Comparison of results for the scenario with a renewal probability of 33%

Figure 5.2: Comparison of the failure rates for the scenarios of the first experiment where availability time of devices is set to 30 minutes (cont)

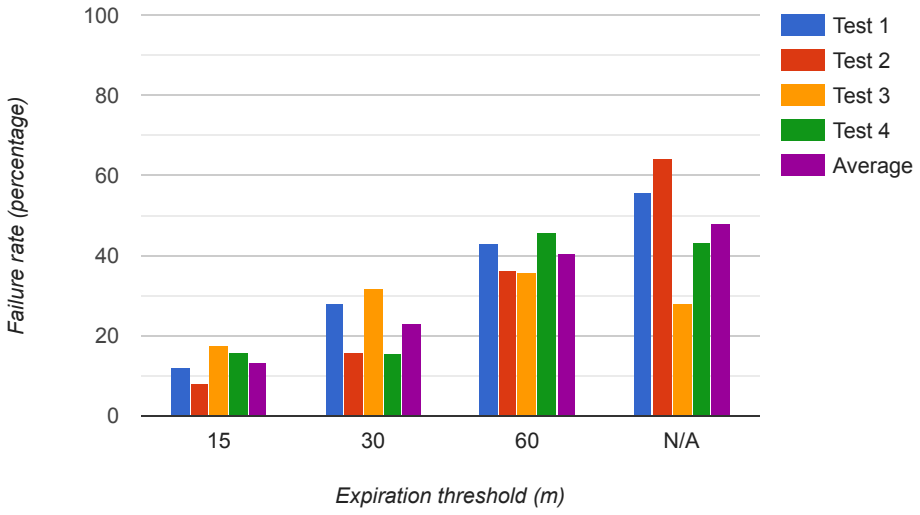
age in minutes and the service feedback average of each scenario, in order to see the impact the selection weighting has in those two factors. Tables 5.10, 5.11, 5.12 and 5.13 show the results for the scenarios where feedback has a 100% weight, feedback and relevance have a 50% weight each, feedback has a 67% weight and relevance has a 33% weight, and feedback has a 33% weight and relevance has a 67% weight, respectively, and Figures 5.4 and 5.5 show the graphic comparison with regards to average feedback value and average service age (how much time has passed for a service since the last renewal of its timestamp) for the four scenarios.

Run	**Number of fetched services**	**Average rating**	**Average feedback value**	**Average service age (m)**
1	90	0.66	0.66	49.51
2	93	0.75	0.75	51.64
3	90	0.7	0.7	50.31
4	87	0.72	0.72	44.82

Table 5.10: Results of the second experiment for the scenario with a feedback weight of 1.

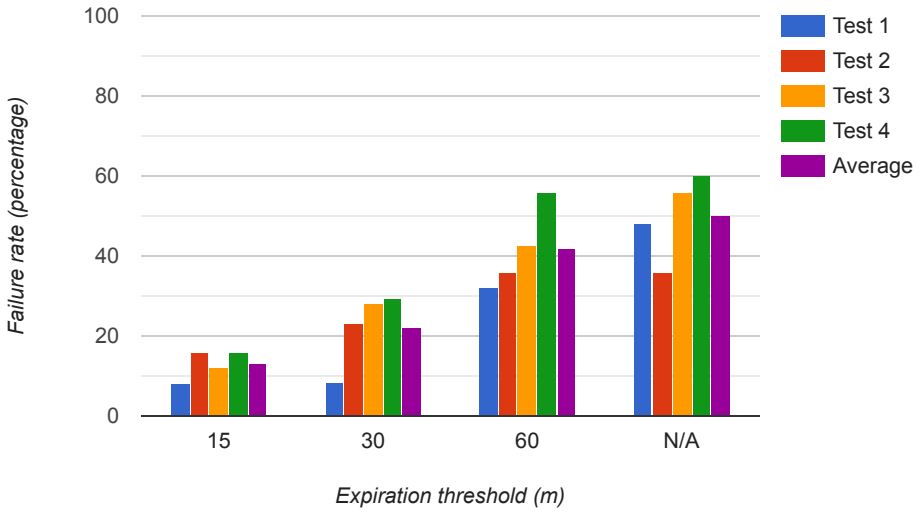


(a) Comparison of results for the scenario with a renewal probability of 50%



(b) Comparison of results for the scenario with a renewal probability of 67%

Figure 5.3: Comparison of the failure rates for the scenarios of the first experiment where availability time of devices is set to 60 minutes



(c) Comparison of results for the scenario with a renewal probability of 33%

Figure 5.3: Comparison of the failure rates for the scenarios of the first experiment where availability time of devices is set to 60 minutes (cont)

Run	Number of fetched services	Average rating	Average feedback value	Average service age (m)
1	90	-0.27	0.55	17.88
2	72	-0.23	0.49	11.38
3	93	-0.28	0.61	18.83
4	90	-0.24	0.58	15.51

Table 5.11: Results of the second experiment for the scenario with feedback and relevance weights of 0.5 each.

Run	Number of fetched services	Average rating	Average feedback value	Average service age (m)
1	90	0.03	0.67	22.49
2	87	0.01	0.62	21.96
3	90	0.02	0.58	18.07
4	93	-0.07	0.57	24.93

Table 5.12: Results of the second experiment for the scenario with a feedback weight of 0.67 and a relevance weight of 0.33.

Run	Number of fetched services	Average rating	Average feedback value	Average service age (m)
1	90	-0.57	0.52	15.84
2	87	-0.51	0.53	13.81
3	93	-0.55	0.57	13.85
4	87	-0.4	0.54	11.11

Table 5.13: Results of the second experiment for the scenario with a feedback weight of 0.33 and a relevance weight of 0.67.

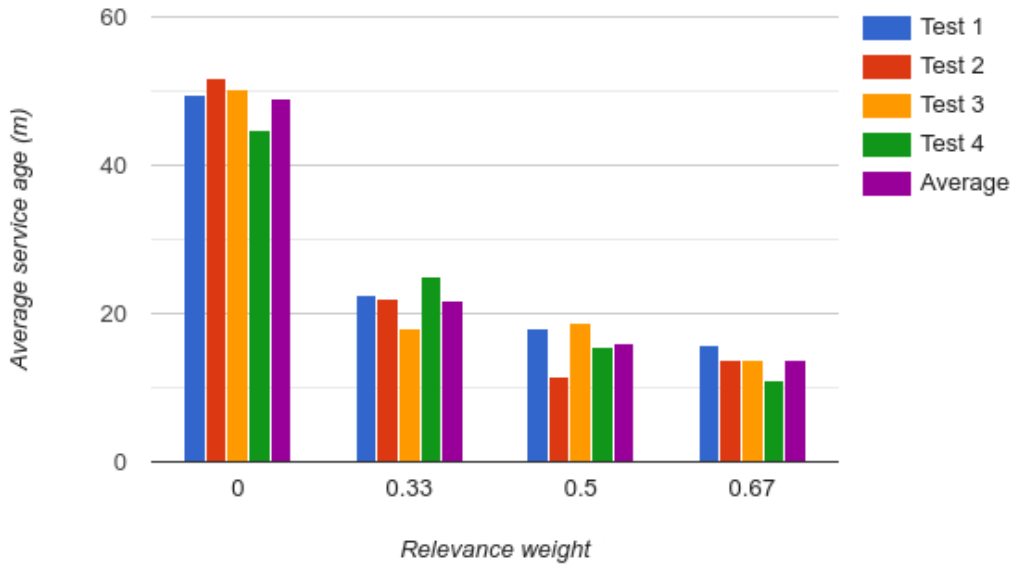


Figure 5.4: Comparison between the weight of relevance in service ratings and service age in the second experiment

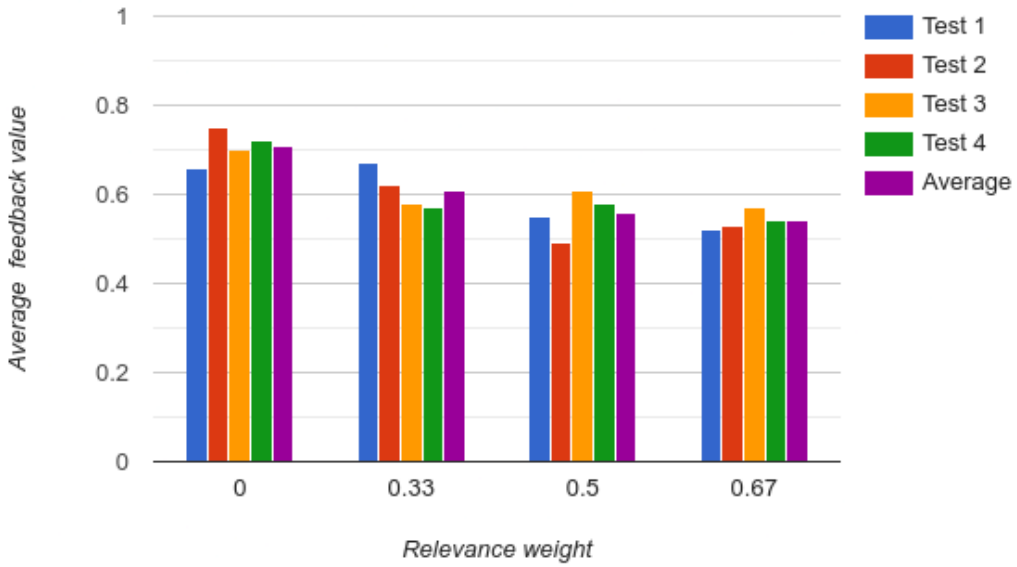


Figure 5.5: Comparison between the weight of relevance in service ratings and average service feedback value in the second experiment

5.4 Discussion

After carrying out experiments and analyzing the results, we can proceed to draw some observations from the gathered data. Regarding the first experiment, the data shows that , if we can estimate an accurate average for device availability time, having a service directory with the capability to remove service registries whose time registered in the directory crosses a threshold lower that the aforementioned average notably decreases the failure rate for services registered in the directory. As we see form the data, the expiration threshold being lower than the availability average is vital: if the threshold is higher, or even the same, than the availability average, the impact in failure rates is significantly lower. However, if that threshold is too low, we get another problem: in a directory with a very low threshold for deleting services, the failure rate is consistently smaller, but at the cost of occasionally having compositions fail due to lack of services to employ. In our experiments with low CS expiration thresholds (e.g. 15 minutes), sometimes compositions would fail due to lack of services of specific type, e.g. needing a service of type I-M-Camera and finding none. This tells

us that expiration thresholds should be carefully picked to be on a sweet spot that is lower than the estimated average availability time for services in a given environment, but high enough to avoid "service starvation": the situation when compositions cannot execute due to the lack of needed components.

The times used for computing the availability of devices in the first experiment were based on empiric evidence of commercial drone's average battery life, bu other devices like smartphones and embedded sensors may have different battery life averages, and also, devices in the environment may by affected by mobility, meaning that the permanence time of such kind of devices is another factor to have in mind. This leads us to believe that, to more effectively retrieve relevant services, we would require a heuristic that weights factors like expected battery life, expected permanence time in the environment and feedback in a specific way for every kind of device, to have a more precise assessment of its relevance. Finding such an heuristic could could enhance the performance of our proposal, and it is definitely an area of opportunity for future work.

Regarding the second experiment, while it is obvious that the introduction of the relevance parameter impacts both the average feedback value and service age of selected services, that impact is dramatically higher for the latter: the sole introduction of relevance into the evaluation process, even if with a low weight (e.g. 0.33), approximately halves the average service age for selected services, while the feedback value takes a lesser hit. While this result seems promising, we think we need to perform experiments with other probability distributions that produce statistics that resemble more closely the feedback real life services may receive, in order to see if the behavior in our experiments holds.

If we analyze the results of both experiments as a whole, we can see that if our directory were to be used by a service composition model that employs our proposed relevance metric, it would give preference to "young" services (services that have a short time span since their registry or last renewal) when selecting component services, thus leaving services that don't renew so frequently to be less frequently used, and perhaps even having a higher chance of being deleted due to the expiration threshold; all this could compel service providers to increase the rate at which they send messages to the directory to renew their services, potentially leading to flooding the pervasive system's network. A study analyzing this possibility would certainly be insightful.

The main idea behind implementing an expiration threshold is that, after a certain time, there is a chance that a pervasive service registered in a service

directory may become unable to be deregistered since its provider may become unavailable due to the dynamic nature of pervasive environments. However, since we completely delete a service registry from the CSD after expiring, what happens with services that may be recurrent in a specific environment? For example, if there was a pervasive service directory registering services that our personal devices (e.g. our smartphones) in an environment we frequent (e.g. our workplace), it would be preferable to maintain registries of them and just flag them as unavailable after crossing the expiration threshold instead of outright deleting their registry, but then, how do we decide when a service is a recurrent member of the environment versus a one-time visitor? How do we establish when a new service stops being a "visitor" to become a recurrent member, or, on the contrary, a recurrent service stops being recurrent?

6

Conclusions and Future Work

In this work, we propose a pervasive service directory for performing service composition, based on both the dynamic memory theory from psychology and cognitive architecture's memory from neurosciences. This service directory is capable of indexing and retrieving services and compositions according to attributes (e.g. name, type of service, if it belongs, the values of its attributes, etc.) and is able to delete services both on the provider's request as well as based on an expiration threshold. It also provides a relevance value that service composition models can use to select services that have spent the least amount of time without being renewed. This led us to believe that our proposal could help us keep a higher degree of consistency between the services present in a pervasive environment and those registered in the directory.

After testing our proposal with the case study, we have seen that implementing service expiration thresholds that are lower than the average expected time services are assumed to stay in a pervasive environment positively affects the failure rate in our experiments, which means the threshold helps weed out the registries of services that may no longer be there. The experiments also showed us that just by taking the relevance value into account for fetching services from the directory, the average service age (the time a service has spent since its last timestamp renewal or since registration if it has never been renewed) falls dramatically, helping select "younger" services and thus reducing the probability of fetching a service that may be unavailable in the environment.

However, as we have discussed above, different devices can be expected to be available for different time spans, e.g. drones and smartphones have different

average battery lives, they move through the environment at different speeds, etc. In order to more precisely asses how much time a device may stay in a pervasive environment, we need more robust heuristic that takes into account the aforementioned factors, and that is specific for every kind of device that can be present at a pervasive environment: smartphones, drones, wireless sensors, etc. This is an area of opportunity for this work that we may tackle in the future.

In addition, while we have a service directory that implements memory functions such as encoding, retrieval and decay, it still lacks the capability to reinforce memory elements: reinforcement is, after all, how human memory prevents itself from forgetting information that is of recurrent use to us. This feature could be the answer to the question of how we prevent the directory to delete services that are recurrent in an environment.

In top of that, the dynamic memory capability to index failed plans according to the causes of such failure can be particularly useful in service composition, where failed compositions can be indexed according to the elements or steps that induced its failure, in order to prevent the system to commit such failure yet again. Such failure can be very valuable, and we intend to to work in the future to achieve it.

References

ALSARYRAH, O. et al. A fast iot service composition scheme for energy efficient qos services. In: *Proceedings of the 2019 7th International Conference on Computer and Communications Management*. New York, NY, USA: Association for Computing Machinery, 2019. (ICCCM 2019), p. 231–237. ISBN 9781450371957. Disponível em: <https://doi.org/10.1145/3348445.3348469>.

ALTMANN, E. M.; SCHUNN, C. D. Decay versus interference: A new look at an old interaction. *Psychological Science*, v. 23, n. 11, p. 1435–1437, 2012. PMID: 23012268. Disponível em: <https://doi.org/10.1177/0956797612446027>.

ANDERSON, J. R. et al. An integrated theory of the mind. *Psychological Review*, American Psychological Association (APA), v. 111, n. 4, p. 1036–1060, 2004. ISSN 0033-295X. Disponível em: <http://dx.doi.org/10.1037/0033-295X.111.4.1036>.

AZIEZ, M. et al. A full comparison study of service discovery approaches for internet of things. *Int. J. Pervasive Comput. Commun.*, Emerald, v. 15, n. 1, p. 30–56, abr. 2019.

BAARS, B. J. *A Cognitive Theory of Consciousness*. New York: Cambridge University Press, 1988.

BAARS, B. J.; FRANKLIN, S. Consciousness is computational: The lida model of global workspace theory. *International Journal of Machine Consciousness*, v. 01, n. 01, p. 23–32, 2009. Disponível em: <https://doi.org/10.1142/S1793843009000050>.

BADDELEY, A. D.; HITCH, G. Working memory. In: BOWER, G. H. (Ed.). Academic Press, 1974, (Psychology of Learning and Motivation, v. 8). p. 47–89. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0079742108604521>.

BAKLOUTI, F. et al. Performing Service Composition in Opportunistic Networks. In: *2019 Wireless Days (WD)*. Manchester, United Kingdom, United Kingdom: IEEE, 2019. p. 1–4. ISBN 9781728101170. ISSN 2156972X.

BRISCOE, G.; WILDE, P. D. Digital ecosystems: Optimisation by a distributed intelligence. In: *2008 2nd IEEE International Conference on Digital Ecosystems and Technologies*. [S.l.: s.n.], 2008. p. 192–197.

BURGHART, C. et al. A cognitive architecture for a humanoid robot: a first approach. In: *5th IEEE-RAS International Conference on Humanoid Robots, 2005.* [S.l.: s.n.], 2005. p. 357–362.

CABRERA, C. et al. The right service at the right place: A service model for smart cities. In: *2018 IEEE International Conference on Pervasive Computing and Communications (PerCom)*. [S.l.: s.n.], 2018. p. 1–10.

CABRI, G. et al. Designing a collaborative middleware for semantic and user-aware service composition. In: *2016 IEEE 25th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*. [S.l.: s.n.], 2016. p. 223–228.

CASTILLO, L. d. J. M. *Diseño de una arquitectura cognitiva bio-inspirada de memoria de trabajo declarativa para entidades informáticas*. Tese (Doutorado) — CINVESTAV del IPN Unidad Guadalajara, 10 2020.

CERVANTES, F. et al. A new approach for the composition of adaptive pervasive systems. *IEEE Systems Journal*, v. 12, n. 2, p. 1709–1721, 2018.

CHAIB, A. et al. Adaptive service composition in an ambient environment with a multi-agent system. *Journal of Ambient Intelligence and Humanized Computing*, Springer Berlin Heidelberg, v. 9, n. 2, p. 367–380, 2017. ISSN 18685145.

CHEN, N. et al. Goal-Driven Service Composition in Mobile and Pervasive Computing. *IEEE Transactions on Services Computing*, v. 11, n. 1, p. 49–62, 2018. ISSN 19391374.

CLARO, D. B. et al. Web services composition. In: ____. *Semantic Web Services, Processes and Applications*. Boston, MA: Springer US, 2006. p. 195–225. ISBN 978-0-387-34685-4. Disponível em: <https://doi.org/10.1007/978-0-387-34685-4_8>.

COWAN, N. The magical number 4 in short-term memory: A reconsideration of mental storage capacity. *Behavioral and Brain Sciences*, v. 24, n. 1, p. 87–114, 2001.

DOULKERIDIS, C. et al. Towards a context-aware service directory. In: BENATALLAH, B.; SHAN, M.-C. (Ed.). *Technologies for E-Services*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003. p. 54–65. ISBN 978-3-540-39406-8.

DOUNCE, I. A. et al. Bio-inspired computational object classification model for object recognition. *Cognitive Systems Research*, v. 73, p. 36–50, 2022. ISSN 1389-0417. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1389041721000802>.

FUNK, C. et al. A model of pervasive services for service composition. In: MEERSMAN, R. et al. (Ed.). *On the Move to Meaningful Internet Systems 2005: OTM 2005 Workshops*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005. p. 215–224. ISBN 978-3-540-32132-3.

GÓMEZ-MARTÍNEZ, D. G. et al. A bioinspired model for the generation of a motivational state from energy homeostasis. *Cognitive Systems Research*, v. 77, p. 125–141, 2023. ISSN 1389-0417. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1389041722000638>.

GONZÁLEZ, J. C. et al. A three-layer planning architecture for the autonomous control of rehabilitation therapies based on social robots. *Cognitive Systems Research*, v. 43, p. 232–249, 2017. ISSN 1389-0417. Disponível em: <https://www.sciencedirect.com/science/article/pii/S138904171630064X>.

GUTIÉRREZ-GARCIA, J. O.; SIM, K. M. Agent-based cloud service composition. *Applied Intelligence*, Springer Science and Business Media LLC, v. 38, n. 3, p. 436–464, abr. 2013.

HAMMOND, K. J. Case-based planning: A framework for planning from experience. *Cognitive Science*, v. 14, n. 3, p. 385–443, 1990. Disponível em: <https://onlinelibrary.wiley.com/doi/abs/10.1207/s15516709cog1403_3>.

HERNÁNDEZ, O. et al. Bio-inspired task-rule retrieval model with auditory sorting test. *Cognitive Systems Research*, v. 72, p. 1–13, 2022. ISSN 1389-0417. Disponível em: <https://www.sciencedirect.com/science/article/pii/S138904172100084X>.

KANDEL, E. R. et al. Learning and memory. In: ____. *Principles of Neural Science*. New York, NY: McGraw Hill, 2021. Disponível em: <accessbiomedicalscience.mhmedical.com/content.aspx?aid=1192999623>.

KAPITSAKI, G. et al. Service composition: State of the art and future challenges. In: *2007 16th IST Mobile and Wireless Communications Summit*. [S.l.: s.n.], 2007. p. 1–5.

KHANOUCHE, M. E. et al. Clustering-based and qos-aware services composition algorithm for ambient intelligence. *Information Sciences*, v. 482, p. 419–439, 2019. ISSN 0020-0255. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0020025519300155>.

KIERAS, D. E.; MEYER, D. E. An overview of the epic architecture for cognition and performance with application to human-computer interaction. *Human-Computer Interaction*, Taylor & Francis, v. 12, n. 4, p. 391–438, 1997. Disponível em: <https://doi.org/10.1207/s15327051hci1204_4>.

KOLODNER, J. L. et al. A process model of cased-based reasoning in problem solving. In: *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*. [S.l.: s.n.], 1985. p. 284–290.

LAIRD, J. E. et al. A standard model of the mind: Toward a common computational framework across artificial intelligence, cognitive science, neuroscience, and robotics. *AI Magazine*, v. 38, n. 4, p. 13–26, Dec. 2017. Disponível em: <https://ojs.aaai.org/index.php/aimagazine/article/view/2744>.

LAIRD, J. E. et al. Soar: An architecture for general intelligence. *Artificial Intelligence*, v. 33, n. 1, p. 1–64, 1987. ISSN 0004-3702. Disponível em: <https://www.sciencedirect.com/science/article/pii/0004370287900506>.

LEI, Y.; JUNXING, Z. Service composition based on multi-agent in the cooperative game. *Future Generation Computer Systems*, v. 68, p. 128–135, 2017. ISSN 0167-739X. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0167739X16302783>.

LIU, C. et al. A reliable and efficient distributed service composition approach in pervasive environments. *IEEE Transactions on Mobile Computing*, v. 16, n. 5, p. 1231–1245, 2017.

MARTIN, L. et al. Bio-inspired cognitive architecture of episodic memory. *Cognitive Systems Research*, v. 76, p. 26–45, 2022. ISSN 1389-0417. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1389041722000390>.

MASCITTI, D. et al. Service provisioning in mobile environments through opportunistic computing. *IEEE Transactions on Mobile Computing*, v. 17, n. 12, p. 2898–2911, 2018.

MCELLISTREM, J. E. Affective and predatory violence: A bimodal classification system of human aggression and violence. *Aggression and Violent Behavior*, v. 10, n. 1, p. 1–30, 2004. ISSN 1359-1789. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1359178903000521>.

METTA, G. et al. Yarp: Yet another robot platform. *International Journal of Advanced Robotic Systems*, v. 3, n. 1, p. 8, 2006. Disponível em: <https://doi.org/10.5772/5761>.

METTA, G. et al. icub: the open humanoid robot designed for learning and developing complex cognitive tasks.

MODHA, D. S. et al. Cognitive computing. *Commun. ACM*, Association for Computing Machinery, New York, NY, USA, v. 54, n. 8, p. 62–71, aug 2011. ISSN 0001-0782. Disponível em: <https://doi.org/10.1145/1978542.1978559>.

O'REILLY, R. C. et al. 91The Leabra Cognitive Architecture: How to Play 20 Principles with Nature and Win! In: *The Oxford Handbook of Cognitive Science*.

Oxford University Press, 2017. ISBN 9780199842193. Disponível em: <https://doi.org/10.1093/oxfordhb/9780199842193.013.8>.

OSTOS, R. et al. Selection of coordination mechanisms in intelligent environments. *IEEE Latin America Transactions*, v. 13, n. 9, p. 3120–3126, 2015.

PARRA, L. A. et al. Computational framework of the visual sensory system based on neuroscientific evidence of the ventral pathway. *Cognitive Systems Research*, v. 77, p. 62–87, 2023. ISSN 1389-0417. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1389041722000481>.

PINTO, M. F. et al. Arcog: An aerial robotics cognitive architecture. *Robotica*, Cambridge University Press, v. 39, n. 3, p. 483–502, 2021.

PLOENNIGS, J. et al. Materializing the promises of cognitive iot: How cognitive buildings are shaping the way. *IEEE Internet of Things Journal*, v. 5, n. 4, p. 2367–2374, 2018.

PRAMANIK, P. K. D. et al. Beyond automation: The cognitive iot. artificial intelligence brings sense to the internet of things. In: SANGAIAH, A. K. et al. (Ed.). *Cognitive Computing for Big Data Systems Over IoT: Frameworks, Tools and Applications*. Cham: Springer International Publishing, 2018. p. 1–37. ISBN 978-3-319-70688-7. Disponível em: <https://doi.org/10.1007/978-3-319-70688-7_1>.

RAYCHOUDHURY, V. et al. K-directory community: Reliable service discovery in manet. *Pervasive and Mobile Computing*, v. 7, n. 1, p. 140–158, 2011. ISSN 1574-1192. Disponível em: <https://www.sciencedirect.com/science/article/pii/S157411921000115X>.

RIESBECK, C. K.; SCHANK, R. C. *Inside Case-Based Reasoning*. Psychology Press, 1989. ISBN 9781134930029. Disponível em: <http://dx.doi.org/10.4324/9780203781821>.

ROMERO, O. J. Cognitively-inspired agent-based service composition for mobile and pervasive computing. In: WANG, D.; ZHANG, L.-J. (Ed.). *Artificial Intelligence and Mobile Services – AIMS 2019*. Cham: Springer International Publishing, 2019. p. 101–117. ISBN 978-3-030-23367-9.

SABBOUH, M. et al. *Workshop on Web services*. World Wide Web Consortium. Disponível em: <https://www.w3.org/2001/03/WSWS-popa/paper08.html>.

SABBOUH, M. et al. Interoperability. In: *Workshop on Web services*. [s.n.], 2001. Disponível em: <https://www.w3.org/2001/03/WSWS-popa/paper08>.

SALAZAR, R. S. et al. Cognitive architecture configuration model for performing dynamic pervasive service composition. *Procedia Computer Science*, v. 213, p. 728–737, 2022. ISSN 1877-0509. 2022 Annual International Conference on Brain-Inspired Cognitive Architectures for Artificial Intelligence: The 13th Annual

Meeting of the BICA Society. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1877050922018269>.

SAMSONOVICH, A. V. Toward a unified catalog of implemented cognitive architectures. In: *Proceedings of the 2010 Conference on Biologically Inspired Cognitive Architectures 2010: Proceedings of the First Annual Meeting of the BICA Society*. NLD: IOS Press, 2010. p. 195–244. ISBN 9781607506607.

SANDOVAL, C. J.; RAMOS, F. F. A proposal of bioinspired motor-system cognitive architecture focused on feed-forward-control movements. *Cognitive Systems Research*, v. 67, p. 50–59, 2021. ISSN 1389-0417. Disponível em: <https://www.sciencedirect.com/science/article/pii/S138904172030098X>.

SCHANK, R. C. *Dynamic Memory Revisited*. [S.l.]: Cambridge University Press, 1999.

SHEU, R.-Y. et al. Multiagent-based adaptive pervasive service architecture (maps). In: *Proceedings of the 3rd Workshop on Agent-Oriented Software Engineering Challenges for Ubiquitous and Pervasive Computing*. New York, NY, USA: Association for Computing Machinery, 2009. (AUPC 09), p. 3–8. ISBN 9781605586472. Disponível em: <https://doi.org/10.1145/1568181.1568185>.

SOMMER, N. L. et al. Multi-strategy dynamic service composition in opportunistic networks. *Information*, v. 11, n. 4, 2020. ISSN 2078–2489. Disponível em: <https://www.mdpi.com/2078-2489/11/4/180>.

Wired2018 STEWART, J. *A Single Drone Helped Mexican Police Drop Crime 10 Percent*. 2018 [Online]. Disponível em: <https://www.wired.com/story/ensenada-mexico-police-drone/>.

TJIONG, M.; LUKKIEN, J. On the consistency of soft-state based service registration. In: *2008 IEEE Globecom Workshops*. [S.l.: s.n.], 2008. p. 1–6.

TRAFTON, J. G. et al. Act-r/e: An embodied cognitive architecture for human-robot interaction. *J. Hum.-Robot Interact.*, Journal of Human-Robot Interaction Steering Committee, v. 2, n. 1, p. 30–55, feb 2013. Disponível em: <https://doi.org/10.5898/JHRI.2.1.Trafton>.

URBIETA, A. et al. Adaptive and context-aware service composition for iot-based smart cities. *Future Generation Computer Systems*, v. 76, p. 262–274, 2017. ISSN 0167-739X. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0167739X16308688>.

VERNON, D. et al. The icub cognitive architecture. In: ____. *A Roadmap for Cognitive Development in Humanoid Robots*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011. p. 121–153. ISBN 978-3-642-16904-5. Disponível em: <https://doi.org/10.1007/978-3-642-16904-5_7>.

VINOSKI, S. Corba: integrating diverse applications within distributed heterogeneous environments. *IEEE Communications Magazine*, v. 35, n. 2, p. 46–55, 1997.

WALDO, J. The jini architecture for network-centric computing. *Commun. ACM*, Association for Computing Machinery, New York, NY, USA, v. 42, n. 7, p. 76–82, jul 1999. ISSN 0001-0782. Disponível em: <https://doi.org/10.1145/306549.306582>.

WALDO, J. The jini architecture for network-centric computing. In: *Commun. ACM*. [S.l.: s.n.], 1999. v. 42, p. 76–82.

WANG, H. et al. A multi-agent reinforcement learning approach to dynamic service composition. *Information Sciences*, v. 363, p. 96–119, 2016. ISSN 0020-0255. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0020025516303085>.

YANG, C.-Y. et al. A brain-inspired self-organizing episodic memory model for a memory assistance robot. *IEEE Transactions on Cognitive and Developmental Systems*, v. 14, n. 2, p. 617–628, 2022.

YANG, K. et al. Policy-based model-driven engineering of pervasive services and the associated OSS. *BT Technol. J.*, Springer Science and Business Media LLC, v. 23, n. 3, p. 162–174, jul. 2005.

YOST, G. R.; NEWELL, A. A problem space approach to expert system specification. In: *Proceedings of the 11th International Joint Conference on Artificial Intelligence - Volume 1*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1989. (IJCAI'89), p. 621–627.

YOUNES, W. et al. Towards an intelligent user-oriented middleware for opportunistic composition of services in ambient spaces. In: *M4IOT 2018 - Proceedings of the 2018 Workshop on Middleware and Applications for the Internet of Things, Part of Middleware 2018 Conference*. Rennes, France: ACM, 2018. p. 25–30. ISBN 9781450361187.

ZHAI, C. et al. A novel cognitive architecture for a human-like virtual player in the mirror game. In: *2014 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. [S.l.: s.n.], 2014. p. 754–759.

ZHOU, J. et al. Psc-rm: Reference model for pervasive service composition. In: *2009 Fourth International Conference on Frontier of Computer Science and Technology*. [S.l.: s.n.], 2009. p. 705–709.

ZHOU, Z. et al. Energy-aware composition for wireless sensor networks as a service. *Future Generation Computer Systems*, v. 80, p. 299–310, 2018. ISSN 0167-739X. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0167739X17303266>.