Cognitively-Inspired Agent-Based Service Composition for Mobile and Pervasive Computing

Oscar J. Romero(✉)

Carnegie Mellon University, Pittsburgh, OA 15213, USA
oscarr@andrew.cmu.edu
http://www.cs.cmu.edu/∼oscarr/

Abstract. Automatic service composition in mobile and pervasive computing faces many challenges due to the complex and highly dynamic nature of the environment. Common approaches consider service composition as a decision problem whose solution is usually addressed from optimization perspectives which are not feasible in practice due to the intractability of the problem, limited computational resources of smart devices, service host's mobility, and time constraints to tailor composition plans. Thus, our main contribution is the development of a cognitively-inspired agent-based service composition model focused on bounded rationality rather than optimality, which allows the system to compensate for limited resources by selectively filtering out continuous streams of data. Our approach exhibits features such as distributedness, modularity, emergent global functionality, and robustness, which endow it with capabilities to perform decentralized service composition by orchestrating manifold service providers and conflicting goals from multiple users. The evaluation of our approach shows promising results when compared against state-of-the-art service composition models.

Keywords: Service composition · Pervasive middleware · Cognition

1 Introduction and Motivation

In Mobile and Pervasive Computing (MPC), a *Service* can be defined as any hardware or software functionality (resources, data or computation) of a smart device that can be requested by other devices for usage [18]. *Service composition* refers to the technique of creating composite services by the aggregation of atomic, simpler and easily executable services. Despite the existence of MPC middleware for automatic service composition [9,10,18,24], there are still some challenges that need to be tackled, as we illustrate in the next example:

Alice and Bob are planning to have a theme party at their home next weekend (high-level goal), so they need to coordinate some tasks among them. To achieve this goal, they interact with a user application (e.g., a personal assistant [25]*, a chatbot* [28]*, etc.) connected to a MPC middleware installed on their mobile* *and wearable devices (e.g., smartphones, smartwatches, tablets, etc.), which act as Service Providers. The high-level goal, which will lead to the creation of a composite service, may be decomposed into 3 sub-goals: buy-food, buy-beer, and buy-home-decoration. There is also a set of atomic services that are hosted by service providers: get-location, find-place, calculate-distance, who-is-nearer, share-shopping-list, go-to-place. Now, each sub-goal is accomplished by the composition of a sequence of services, e.g.: buy-food =* {*get-location(user)* → *find-place(supermarket)* → *calculate-distance(user, supermarket)* → *who-is-nearer(market, users)* → *share-shopping-list(users)* → *go-to-place(user, market)*}.

Given the previous example we focus on five challenges, so service composition should: (1) consider preferences from multiple users; (2) coordinate the interaction between services hosted by different service providers; (3) consider resource scarcity in smart devices [21]; (4) perform dynamic adaptation to unpredictable changes occurring in the environment; (5) deal with both short-term and long-term user's goals. Performing service composition while taking into account a myriad of variable factors as described above (e.g., users, services, service providers, QoS values, context, etc.), makes the problem become intractable even for approaches that use dynamic composition. The main issue with these approaches is that they propose solutions focused on optimality (e.g, graph-, rule-, and workflow-based solutions), which do no consider limitations imposed by the decision-making process (specially on smart devices), revealing their inability to process and compute the expected utility of every alternative action when variable factors grow in size (combinatorial explosion). Therefore, we propose a cognitively-inspired approach based on bounded rationality, which centers on the fact that perfectly rational decisions are often not feasible in practice because of the intractability of the problem, the limited computational resources, and time constraints; instead, our approach seeks satisfactory solutions rather than optimal ones. Thus, our main contributions are twofold: (1) We propose *COPERNIC*, a cognitively-inspired agent-based service composition middleware, as a first approach to addressing the five challenges described above (i.e., multiple users, decentralized coordination, and inexpensive, dynamic, and long-short term composition) using a bounded rationality approach; and (2) We develop a prototype of *COPERNIC* and evaluate its performance against to state-of-the-art service composition models. This paper is organized as follows: Sect. 2 outlines the system architecture. Section 3 details our approach, and Sect. 4 reports the experimental results. Sections 5 and 6 presents the related work and the conclusions, respectively.

2 Overview

2.1 Preliminaries

As considered in the literature [3], we distinguish two types of services: *abstract* and *concrete* services. Formally, a concrete service cs_i is a tuple $\langle cs_i^{in}, cs_i^{out}, cs_i^{prec}, cs_i^{postc}, cs_i^{QoS}, cs_i^{ctx} \rangle$ that performs a functionality by acting on input data (cs_i^{in}) to produce output data (cs_i^{out}), with pre-conditions (cs_i^{prec}), post-conditions (cs_i^{postc}), Quality of Service requirements (cs_i^{QoS}), and contextual information. An abstract service as_i is a tuple $\langle as_i^{pre}, as_i^{post}, as_i^{cs} \rangle$ realized by several concrete services that offer the same functionality ($as_i^{cs} \in \{cs_{(i,1)}, cs_{(i,2)}, ..., cs_{(i,n)}\}$) with preconditions ($as_i^{pre}$) and postconditions ($as_i^{post}$) such that $\forall cs_{(i,j)}, cs_{(i,k)} \in as_i^{cs} / (as_i^{pre} = cs_{(i,j)}^{pre} \cap cs_{(i,k)}^{pre}) \wedge (as_i^{post} = cs_{(i,j)}^{post} \cap cs_{(i,k)}^{post})$.

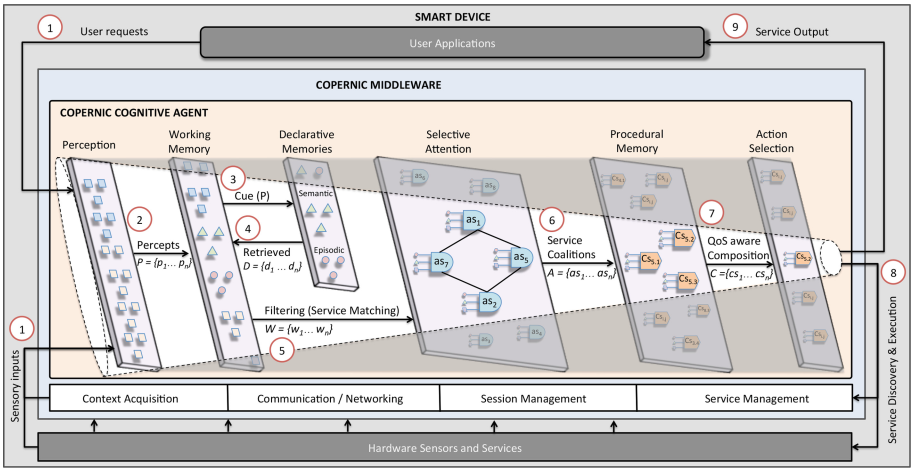


Fig. 1. *COPERNIC*'s overall architecture (single device). The white cone illustrates how a continuous stream of data is filtered out so only the most relevant elements are retained for the composition while the others are either discarded or put on hold until they receive more activation to become participants.

2.2 System Architecture

Figure 1 depicts the overall architecture of *COPERNIC*, though it is worth noting that it does not reflect yet the distributed nature of our model. The *COPERNIC* Agent is a cognitive module inspired by architectural principles defined by the Common Model of Cognition (CMC) [13,20], a computational model that captures a consensus about the structures and processes that are similar to those found in human cognition. Next, we briefly describe *COPERNIC*'s pipeline (Fig. 1) and its realization on the CMC model. In *Step 1*, the *Perception* module makes sense of the agent's current state by processing both external (e.g., user requests) and internal (i.e., signals from other modules) sensory inputs, categorizing that information, and recognizing situations where a set of abstract services may be triggered. In *Step 2*, the *Perception* module outputs a set of symbolic structures (percepts) that are stored in a Working Memory (WM) for further processing as abstract service inputs. In *Step 3*, the WM cues the declarative memories (i.e., *Episodic Memory* that retrieves information about historic performance of services, context, etc., and *Semantic Memory* that retrieves service definitions, user preferences, etc.) and stores local associations in *Step 4*. In *Step 5*, the content of the WM is filtered out by the attention mechanism so the agent only focuses on the most relevant information needed for matching abstract services. In *Step 6*, goals are decomposed and abstract services compete and cooperate (creating coalitions) among them in order to get the focus of attention. In *Step 7*, the *Procedural Memory* executes a set of heuristics to dynamically bind abstract services to concrete services by validating the QoS requirements. In *Step 8*, the *Action Selection* chooses the most appropriate action to execute a concrete service using discovery protocols adapted to the heterogeneous nature of the environment (the process is repeated until all subgoals are satisfied). In *Step 9*, service's output is returned to the application. Unlike traditional approaches that create uPFRont composition plans which are prone to inadaptability, in our approach, plans emerge from the interaction of cascading sequences of **cognitive cycles** corresponding to perception-action loops (steps 1–8) where compositional conditions are validated in every cycle. This contribution allows service composition to be more reactive, robust, and adaptive to dynamic changes while composition plans are generated on-the-fly by using minimal resources as a result of filtering out a continuous stream of data.

3 Approach

A. Perception
The perception module defines a set of feature detectors in charge of detecting and classifying the sensory inputs, i.e., there are different feature detectors to identify external stimuli (i.e., user requests) and internal stimuli (i.e., user's context, physical context – sensor readings, and service QoS). Feature detectors create *percepts*, units of perceived information with a symbolic representation (key-value pairs, e.g., `<location, home>`) and an activation level. Percepts activation rapidly decay over time (when not re-stimulated) according to the following inverse sigmoid function: $p_{act_i} = \frac{sal_i}{\log_2 cc_i}$, where p_{act_i} is the activation for percept i, sal_i is the salience of the stimulus (the quality by which a percept stands out from its neighbors), and cc_i the number of cognitive cycles since the last time the percept received activation. Salience is a numeric value between 1–10 (being 10 the most salient stimulus) and serves for designers to add some relevance to perceived information, e.g, a "user request" percept may have a higher salience (so it should last longer) whereas "Temporary WiFi disconnection" may have a lower salience. This module outputs a set of percepts $P = \{p_1 ... p_n\} \mid \forall p_i \in PR$, where PR is a set of premises such that $as^{pre} \cup cs^{pre} \subseteq PR$.

B. Short-term Working Memory (WM)
WM holds previous percepts not yet decayed away, and local associations from declarative memories that are combined with the percepts to understand the current state of the composition. Information written in the WM may reappear in different cognitive cycles until it fades away. To that purpose, WM defines a limited storage capacity (default: 7 units [16]) and a recency-based decay function that keeps active a limited number of units, expressed as a base-level activation

function [2]: $B_i^w = iB_i^w + \sum_{l=1}^{n} t_l^{-d}$, where i is a WM unit (w_i), l is the lth setting of w_i, t_l is the time since lth unit was presented, iB_i^w is the initial value of activation, and d is a decay parameter that reflects differences in WM units volatility, e.g., dynamic changes on *user context* happen more often than changes on *user preferences*, so the former should have a higher decay value, whereas the latter should have a lower one, which makes user context obsolete quicker than user preferences. If B_i^w is above a threshold ($B_i^w > t_w$) then w_i will be used as an input for service matching, i.e., $w_i \subseteq (as^{pre} \cup cs^{pre}) \subseteq PR$.

C. Long-term Episodic Memory (EM)
EM is a content-address-able associative memory that records temporal sequences of user and system events. *COPERNIC* defines 3 types of EM: (1) *User EM* that stores user past actions (e.g., Bob searched for nearby beer shops after doing the shopping); (2) *Service Provide EM* that stores historic data of service and service provider performance (efficiency, reliability, QoS, reputation, failures, etc.); and (3) *Network EM* that stores neighboring updates, network hops, etc. Any unit written to the WM cues a retrieval from EM, returning prior activity associated with the current entry. We used a Sparse Distributed Memory (SDM) [11], a high-dimensional space that mimics a human neural network. SDM is lightweight, random (it retrieves service associations in equal time from any location), content-addressable (it finds complete contents using content portions), and associative (it finds contents similar to a cue).

D. Long-term Semantic Memory (SM)
SM is intended to capture the meaning of concepts and facts about: (1) service descriptions; (2) the world (e.g., `<Home><is-a><place>`); and (3) the user (e.g., preferences, goals, etc.). SM is implemented using a Slipnet, an activation passing semantic network, where each concept is represented by a node, and each conceptual relationship by a link having a numerical length, representing the "conceptual distance" between the two nodes involved, which is adjusted dynamically. The shorter the distance between two concepts is, the more easily pressures can induce a slippage (connection) between them. Nodes acquire varying levels of activation (i.e., measure of relevance to the current situation in the WM) and spread varying amounts of activation to neighbors. Let $B_i^s = B_{i-1}^s + \sum_{j=0}^{n}(k - L_{i,j})$ such that B_i^s and B_{i-1}^s are the current and previous activations of node i, respectively; k is a constant for regulation of spreading activation; and $L_{i,j}$ is the conceptual length between nodes i and j. Traditional semantic-driven approaches for service composition use ontology-based description languages, however, these static representations do not account for the dynamicity of the environment, require the use of semantic reasoners and ontologies, and lack a mechanism to represent conceptual distance. The declarative module outputs a set of premises $D = \{d_1...d_n\} \mid \forall d_i \in PR$.

E. Selective Attention (SA)
Based on Posner's theory of attention [15], our SA filters out a continuous stream of content from WM while carrying out three attentional functions: (1) maintaining an alert state (e.g., SA gives priority to salient information like context and

QoS); (2) focusing agent's senses on the required information (e.g., to discover *get-location* service and bring it into composition, SA needs to focus on changes on GPS readings); and (3) the ability to manage attention towards goals and planning (e.g., SA focuses on the high-level goal "plan a party at home" and its corresponding sub-goals). SA uses a Behavior Network (BN) [14] , a hybrid system that integrates both a connectionist computational model and a symbolic, structured representation. BN defines a collection of behaviors (nodes) that compete and cooperate among them (through spreading activation dynamics) in order to get the focus of attention. In *COPERNIC*, each behavior maps to a single abstract service as, and *"service discovering/matching"* is modeled as an emergent property of activation/inhibition dynamics among all abstract services. Revisiting the formal definition of an abstract service as_i, we have the tuple: $\langle as_i^{pre}, as_i^{add}, as_i^{del}, as_i^{\alpha} \rangle$, where as_i^{pre} is a list of preconditions that have to be true before the service becomes active, as_i^{add} and as_i^{del} represent the expected (positive and negative) postconditions of the service in terms of an "add" and a "delete" lists, and as_i^{α} is the level of activation. If a WM unit w_i is in as_i^{pre} then there is an active link from w_i to as_i. If the goal g (i.e., a user request or any sub-goal stored in WM) is in as_i^{add} then there is an active link from g to as_i. There is a successor link from service as_i to service as_j for every WM unit such that $w_i \in as_i^{add} \cap as_j^{pre}$. A predecessor link from as_j to as_i exists for every successor link from as_i to as_j. There is a conflicter link from as_i to as_j for every WM unit such that $w_i \in as_j^{del} \cap as_i^{pre}$. Additionally, the model defines five global parameters that can be used to tune the global behavior of the network: π is the mean level of activation, θ is the threshold for becoming active, ϕ is the amount of activation energy a WM unit injects into the network, γ is the amount of energy a goal injects into the network, and δ is the amount of activation energy a protected goal takes away from the network. These global parameters make it possible to mediate smoothly between service selection criteria, such as trading off goal-orientedness for situation-orientedness, adaptivity for inertia, and deliberation for reactivity (see Listing 1.1).

```
input: a set of WM units $W$, a set of goals $G$, cognitive cycle $t$
output: selected abstract service $AS$
$A$ := set of registered abstract services //$A = \{as_1 ... as_n\}$
$M_j$ := nil //$\forall as \in A, j \in W \mid M_j = \sum_{i=1}^{n} \#(as_i^{pre} \cap j)$
$X_j$ := nil //$\forall as \in A, j \in W \mid X_j = \sum_{i=1}^{n} \#(as_i^{add} \cap j)$
$U_j$ := nill //$\forall as \in A, j \in W \mid U_j = \sum_{i=1}^{n} \#(as_i^{del} \cap j)$
for each abstract service $as_i$ in $A$ do:
  $AW_{(i,t)}$ := $\sum_j \phi \cdot (1/M_j) \cdot (1/\#(as_i^{pre}))$ //compute activation from current WM state ($AW$)
  $AG_{(i,t)}$ := $\sum_j \gamma \cdot (1/X_j) \cdot (1/\#(as_i^{add}))$ //compute activation from goals ($AG$).
  $TG_{(i,t)}$ := $\sum_j \delta \cdot (1/U_j) \cdot (1/\#(as_i^{del}))$ //take activation away from achieved goals ($TG$)
  $BW_{(i,t)}$ := $\sum_j as_i^{\alpha(t-1)} \cdot (1/X_j) \cdot (1/\#(as_i^{add}))$ //spread activation energy backward ($BW$)
  $FW_{(i,t)}$ := $\sum_j as_i^{\alpha(t-1)} \cdot (\phi/\gamma) \cdot (1/X_j) \cdot (1/\#(as_i^{add})))$ //spread activation energy forward
  $as_{(i,t)}$.act := $EW_{(i,t)} + EG_{(i,t)} - TG_{(i,t)} + BW_{(i,t)}) + FW_{(i,t)}$ //total activation for $as_i$
end for
return $AS$ := $\max_{act}(A)$
```

Listing 1.1. Pseudocode for the Spreading Activation Dynamics of *COPERNIC*'s Attentional Mechanism (see Section F.)

F. Procedural Memory (PM)
PM defines a set of heuristics (in the form of productions) to: (1) discover and match concrete services based on contextual information and QoS attributes; and (2) adjust the BN parameters in order to make the global behavior be more adaptive (i.e., deliberative vs. reactive, goal-oriented vs. situation-oriented, etc.) depending on the task requirements. Suppose there are two concrete services associated to "get-location" abstract service (e.g., Bob's phone hosts cs_p and Bob's smartwatch hosts cs_w), and each concrete service has 2 QoS features: accuracy and latency. The accuracy of cs_p is better since it uses fused location algorithms and its GPS sensor provides more precise readings, but its latency is higher than cs_w, so at a given moment, a heuristic production might prefer to match cs_w, even if this is not as accurate as cs_p, just because it can deliver a faster response during time-sensitive compositions. Regarding tuning BN parameters, PM applies the following heuristics [19] to keep the balance between: (1) goal-orientedness vs. situation-orientedness, $\gamma > \phi$; (2) deliberation vs. reactivity, $\phi > \gamma \wedge \phi > \theta$; (3) bias towards ongoing plan vs. adaptivity, $\phi > \pi > \gamma$; and (4) to preserve sensitivity to goal conflict, $\delta > \gamma$. The corresponding values are dynamically adapted over time and using a *reinforcement learning mechanism* based on the heuristic utility. Utility learning for a heuristic i after its nth usage is: $U_i(n) = U_i(n-1) + \alpha[R_i(n) - U_i(n-1)] + \epsilon$, where α is the learning rate (default [2]: 0.2), $R_i(n)$ is the effective reward value given to heuristic i for its nth usage, ϵ is a temperature (level of randomness) that is decreased over time, i.e., $\epsilon = 1/e^{(n/k)}, k = 0.35$ (determined empirically [2]).

G. Action Selection (AS)
AS processes different kind of actions: (1) internal actions such as goal setting (it adds new goals to both the WM and SA modules); and (2) external actions such as triggering a device's effector/actuator, and invoking the discovery mechanism to look up a concrete service and then execute it. AS uses a scheduler mechanism to sort (by priority) and execute actions in the future.

H. Cognitive Cycle
In mapping to human behavior [13], *COPERNIC*'s cognitive cycles operate at roughly 50 ms, although the actions that they trigger can take significantly longer to execute. A cognitive cycle starts with sensing and usually ends with selection of an internal or external action. The cognitive cycle is conceived as an active process that allows interactions between the different components of the architecture. Deliberation, reasoning, and generation of plans in *COPERNIC* take place over multiple cascading cognitive cycles in the current situation (i.e., multiple overlapping cycles iterating at an asynchronous rate, see Listing 1.2).

```
input: set of sensory inputs $SI$, set of user goals $G$
output: selected concrete service $CS$
$P$ := nil //set of salient percepts ($P = \{p_1..p_n\}$)
$W$ := nil //set of active units in the WM ($W = \{w_1..w_n\}$)
$D$ := nil //set of retrieved declarative units ($D = \{d_1..d_n\}$)
$A$ := set of registered abstract services ($A = \{as_1..as_n\}$)
$C$ := set of registered concrete services ($C = \{cs_1..cs_n\}$)
$R$ := nil //set of resulting actions
```

```
while remaining goals $G > 0$ do: //cognitive cycle $i$
  $P_i$ := detect, classify and temporarily store $SI$ and $G$
  $W_i$ := add salient percepts to WM ($W_i = W_{(i-1)} \cup P_i$)
  $D_i$ := cue and retrieve declarative memories using the content of WM
  $W_i$ := add declarative units to WM ($W_i = W_i \cup D_i$)
  $W_i$ := decay and filter WM units ($W_i = W_d$, where $W_d \subseteq W_i$)
  $A_i$ := focus attention and do service matching ($A_i = W_i \cap A^{pre}$)
  $C_i$ := apply heuristics (PM) and select concrete service candidates ($Ci = A_i \cap QoS$)
  $CS$ := select a concrete service
  $R_i$ := wrap $CS$ execution as an action and add it to the set of actions
  $a$ := prioritize actions $R_i$ and pick the most relevant
  if $a$ is of type ''service-execution'' then execute concrete service's action $a.CS$
  else execute internal action
end while
```

Listing 1.2. Pseudocode for COPERNIC's Cognitive Cycle (see Fig. 1)

I. Session Management
This module manages shared sessions across multiple user's *Devices* (service providers). A running instance of *COPERNIC* is hosted by each device. Using proximity discovery protocols, devices are grouped by physical nearness into *Groups*. For instance, on Fig. 2, Group *"bob-with-me"* represents the devices that Bob carries with him whereas *"bob-home"* represents a Group of devices at Bob's home. Groups belonging to the same user are logically grouped into *Sessions*, where each Session guarantees that multiple ubiquitous devices can share information and collaborate in a distributed, inexpensive and robust fashion.

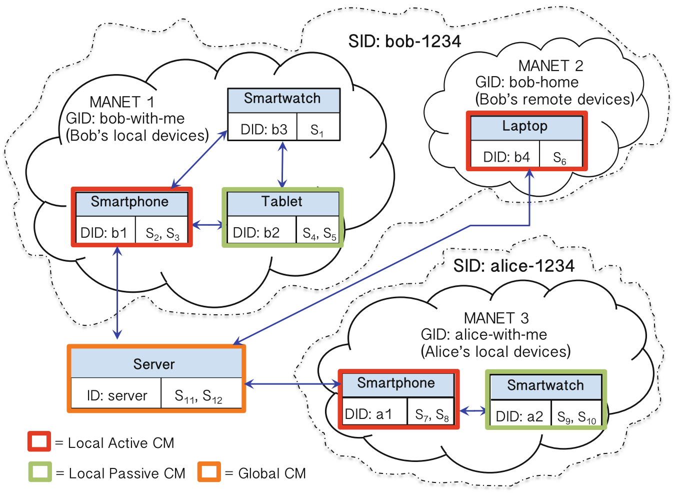


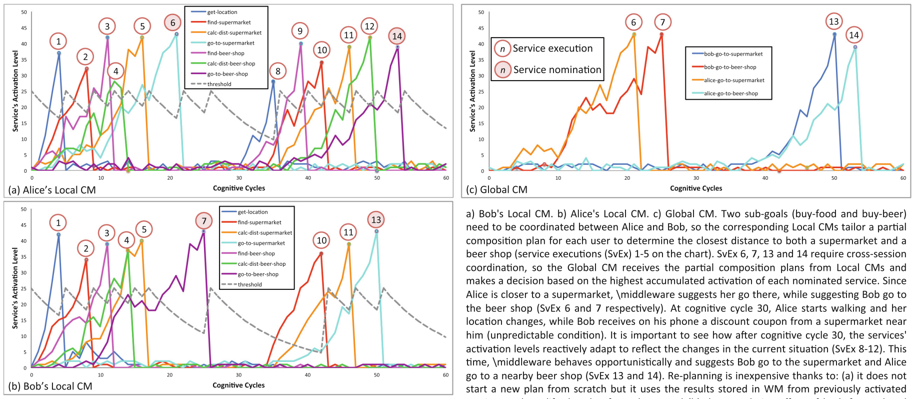
Fig. 2. Infrastructure-less MPC environment. *COPERNIC* agents are distributed across different local networks (MANET) and collaborate during composition. DID: Device ID, GID: Group ID, SID: Session ID, $S_1..S_n$: Services.

J. Service Management
Service management (i.e., service discovery, service coordination, and management of the information flow between service components) is performed by a *Composition Manager* (CM). Service management tasks are distributed on an as-needed basis among multiple *Composition Managers* (CMs) in both infrastructure-based and infrastructure-less networks. Unlike traditional approaches, where transferring the control from one CM to another does not consider fatal failures that prevent such transference, our approach is based on a Binary Star pattern that serves for primary-backup node failover. That is, *COPERNIC* chooses two CMs per Group, one works in active mode while the other one in passive mode. Both active and passive CMs monitor each other and if the active CM disappears from the network, then the passive one will take over as active CM and send a request for another device to join as a passive CM. Using a ranking function based on four levels of device capabilities, as show in Table 1, *COPERNIC* is able to determine which devices will be the active and passive CMs, that is: $R_{cm} = Pri_{cm} \times Per_{cm}$, where R is the ranking of CM cm, Pri is a level-based priority function (i.e., Level-0 = 0... Level-3 = 3), and Per is a CM performance function based on composition completeness, composition time, etc.

Table 1. Composition manager levels

Feature	**Level-0**	**Level-1**	**Level-2**	**Level-3**
COPERNIC version	Minimal	Lightweight	Full	Full
Resources (CPU, memory)	Scarce	Limited	Rich	Rich
Can act as CM?	NO	YES	YES	YES
Cross-session composition?	NO	NO	NO	YES
Example devices	Sensors	Wearables	Smartphone, Laptop	PC, server

It is worth noting that only Level-3 can perform *global cross-session compositions* (i.e., involving multiple users and sessions), though Level-2 and Level-1 can still perform *local compositions* (i.e., within a group) so the whole composition task is distributed among multiple CMs, as shown in Fig. 2. Each Local CM tailors a partial composition plan that satisfies the needs of its user by using local resources and services; then a Global CM, which can see the whole picture, receives as inputs partial plans from Local CMs and makes "good enough" decisions using the available information. By good enough we mean that, rather than trying to look for an optimized solution, CMs exploit opportunities that contribute to the ongoing goal/plan while adapting to unpredictable changing situations. Figure 3 illustrates how CMs make good enough decisions as a result of the spreading activation dynamics coordinated by *COPERNIC* cognitive agents.



a) Bob's Local CM. b) Alice's Local CM. c) Global CM. Two sub-goals (buy-food and buy-beer) need to be coordinated between Alice and Bob, so the corresponding Local CMs tailor a partial composition plan for each user to determine the closest distance to both a supermarket and a beer shop (service executions (SvEx) 1-5 on the chart). SvEx 6, 7, 13 and 14 require cross-session coordination, so the Global CM receives the partial composition plans from Local CMs and makes a decision based on the highest accumulated activation of each nominated service. Since Alice is closer to a supermarket, \middleware suggests her go there, while suggesting Bob go to the beer shop (SvEx 6 and 7 respectively). At cognitive cycle 30, Alice starts walking and her location changes, while Bob receives on his phone a discount coupon from a supermarket near him (unpredictable condition). It is important to see how after cognitive cycle 30, the services' activation levels reactively adapt to reflect the changes in the current situation (SvEx 8-12). This time, \middleware behaves opportunistically and suggests Bob go to the supermarket and Alice go to a nearby beer shop (SvEx 13 and 14). Re-planning is inexpensive thanks to: (a) it does not start a new plan from scratch but it uses the results stored in WM from previously activated services and modify the plan from there; and (b) the cumulative effect of both forward and backward spreading activation helps service execution trigger faster

Fig. 3. Spreading activation dynamics for the example described in Sect. 1

4 Evaluation

A. Experimental Setup
We implemented the dynamic composition overlay on the NS-3 simulator using the experimental settings on Table 2. We compared *COPERNIC* against two state-of-the-art decentralized service composition models: GoCoMo, a goal-driven service model based on a decentralized heuristic backward-chaining planning algorithm [5]; and CoopC, a decentralized cooperative discovery model that employs a backward goal-driven service query and forward service construction mechanism but does not support runtime composite service adaptation [8].

Table 2. Experimental settings

Setting	**Value**
Number users (1 goal/user)	1, 2
Service density (# providers)	sparse (**SD-S**): 20, medium (**SD-M**): 40, dense (**SD-D**): 60
Composition length	5 services (**CL-5**), 10 services (**CL-10**)
Node mobility	slow (**M-S**): 0–2 m/s, medium (**M-M**): 2–8 m/s, fast (**M-F**): 8–13 m/s
Number of services	Abstract: 10, Concrete: 40 (4 per abstract service)
Communication range	250 m
Sem. matchmaking delay	0.2 (s) [5]
Sample per experiment	100 runs
Node movement	Random walking Monte Carlo model
Pre/post-cond per service	Random (1-4)

We run 2 different experiments and measured the composition efficiency using 4 different metrics: composition time (**CT** in seconds), average memory usage of all service providers involved in the composition (**MU** in Kb), a planning failure rate (**PFR**) calculated as the ratio of the number of failed planning processes to the number of all the issued requests during the simulation cycles, and the execution failure rate (**EFR**) computed as the ratio of the number of failed executions to the number of all the successful plans.

B. Flexibility of Service Composition
This scenario evaluates the flexibility of *COPERNIC*, GoCoMo, and CoopC during the generation of service composites while varying node mobility, service density, and service complexity (composition length). This scenario uses the configuration presented in Table 2 and only one user. The experimental results are shown in Table 3 (blue and red cells are the best and worst measurements for each category, respectively). Overall, GoCoMo got the lowest failure rates (PFR), followed by *COPERNIC* and then by CoopC, though *COPERNIC*'s composition time (CT) and memory usage (MU) were the lowest in comparison to the other two approaches. In particular, GoCoMo got a lower failure rate (12–38%) than *COPERNIC* when the mobility was slow. This difference dropped to 7–13% in fast-mobility high-density scenarios thanks to *COPERNIC* is less sensitive to mobility changes because the information about participant services is stored in the WM and gradually fades away, so when, for instance, a service provider disappears and reappears later in time, the probability that this service provider is still in the WM is high (due to its activation may have not decayed entirely), so it will be able to promptly participate again in the composition without producing significant planning failures. In comparison with CoopC, *COPERNIC* got 12–25% less failures due to CoopC's does not support runtime adaptation and poorly handles mobility and density changes. Regarding composition time, *COPERNIC* tailored composite services up to 42% faster than GoCoMo and up to 71% faster than CoopC; and it used up to 72% less memory than GoCoMo and up to 84% less memory than CoopC. The reason for this significant reduction in composition time and resource consumption is that *COPERNIC* is continuously filtering out the stream of incoming information (sensory stimuli), which keeps it into reasonable margins of resources usage, despite of the dynamism of the environment. It is worth noting that *COPERNIC* did not show a significant difference in memory usage when using a composition length of either 5 or 10 services (-4%–11%) in comparison with GoCoMo (60%–190%) and CoopC (157%–201%), which suggests that our approach could be smoothly scaled up.

C. Adaptability of Service Composition
In this scenario we simulated 2 users with one goal each. Then, in the middle of the composition users switched their goals (switch point). We measured the ability (in terms of planning and executing failure rates) of both *COPERNIC* and GoCoMo to adapt to the new situation and replan a different composite service for both users while using different settings for mobility, density, and composition length. In this experiment we did not consider CoopC due to it

Table 3. Flexibility of service composition

			M-S			M-M			M-F		
			SD-S	**SD-M**	**SD-D**	**SD-S**	**SD-M**	**SD-D**	**SD-S**	**SD-M**	**SD-D**
COPERNIC	**CL-5**	**PFR (%)**	18.2	3.7	1.1	17.5	1.4	1.4	21.1	3.3	1.1
		CT (sec)	0.9	0.5	0.8	1.1	1.2	1.2	1.1	1.4	1.4
		MU (Kb)	63	81	93	67	86	93	73	88	98
	CL-10	**PFR (%)**	17.8	3.7	1.1	17.7	1.5	0.5	19.7	3.8	1.4
		CT (sec)	1.2	0.6	0.8	1.2	1.2	1.1	1.2	1.3	1.9
		MU (Kb)	70	86	92	70	73	85	78	89	94
GoCoMo	**CL-5**	**PFR (%)**	13.1	3.3	0.6	16.1	1.2	0.3	18.0	3.1	0.9
		CT (sec)	1.3	0.7	0.9	1.3	1.4	1.4	1.3	1.3	1.4
		MU (Kb)	79	93	112	78	93	110	80	94	114
	CL-10	**PFR (%)**	16.2	2.3	0.8	24.7	1.1	0.4	22.1	3.5	1.3
		CT (sec)	2.1	2.2	2.2	2.2	2.3	2.3	2.3	2.4	2.4
		MU (Kb)	213	273	314	201	287	308	221	286	345
CoopC	**CL-5**	**PFR (%)**	16.2	2.4	0.8	21.9	1.3	2.3	24.5	3.7	1.2
		CT (sec)	1.8	1.9	1.9	1.9	1.8	2.1	1.9	2.1	2.2
		MU (Kb)	114	245	367	121	239	353	117	275	359
	CL-10	**PFR (%)**	24.0	2.3	1.3	25.2	2.4	1.2	31.8	4.2	1.6
		CT (sec)	4.1	4.2	4.2	4.5	4.7	4.9	5.0	5.1	5.5
		MU (Kb)	325	476	593	332	488	605	345	497	657

cannot perform runtime service composition adaptation. Also, for the sake of simplicity, we only used a composition length of 5 services. Since GoCoMo lacks both multi-goal and multi-user composition processing, we had to run simultaneously 2 instances of GoCoMo with one goal each, and then switched the goals at the specific switch point. The individual measurements of both instances were added up, this makes GoCoMo more comparable against *COPERNIC*. In order to demonstrate the adaptability of *COPERNIC*, we used two different configurations for its attentional mechanism. A configuration is defined as a tuple of values corresponding to the Behavior Network's parameters described in Sect. 4.E, such that $C_i = \langle \theta_i,\ \pi_i,\ \phi_i,\ \gamma_i,\ \delta_i \rangle$. Configuration C_1 uses the default values for the attentional mechanism: $C_1 = \langle 30, 20, 20, 20, 20 \rangle$, while configuration C_2 uses values discovered by the utility learning mechanism (described in Sect. 4.F) after 100 test runs: $C_2 = \langle 22, 27, 42, 23, 18 \rangle$. Results are presented in Table 4.

Table 4. Adaptability of service composition

		M-S			M-M			M-F		
		SD-S	**SD-M**	**SD-D**	**SD-S**	**SD-M**	**SD-D**	**SD-S**	**SD-M**	**SD-D**
COPERNIC - C1	**PFR (%)**	23.1	5.2	2.1	21.1	4.3	1.9	21.9	4.8	2.0
	EFR (%)	45.2	22.6	15.9	69.4	31.2	21.4	78.5	43.9	37.0
	CT (sec)	5.8	4.2	3.1	6.1	4.4	3.7	6.4	4.5	3.9
	MU (Kb)	85	93	110	90	102	117	95	111	123
COPERNIC - C2	**PFR (%)**	20.7	4.7	1.6	18.5	3.6	1.4	19.8	4.2	1.7
	EFR (%)	38.5	17.3	11.3	58.2	19.3	14.9	63.8	29.8	26.6
	CT (sec)	4.2	3.1	2.3	5.5	3.9	2.9	5.6	4.0	3.1
	MU (Kb)	84	93	109	91	103	115	96	112	120
GoCoMo	**PFR (%)**	20.3	4.2	1.4	18.1	3.5	1.1	19.1	3.9	1.6
	EFR (%)	43.5	18.2	13.2	67.4	24.2	16.8	76.3	35.7	32.4
	CT (sec)	5.4	3.9	3.2	5.9	4.3	3.5	6.6	4.6	3.9
	MU (Kb)	428	536	678	544	623	701	598	703	812

Despite the fact that GoCoMo had lower planning failure rates than *COPERNIC* (12%–42% and 2%–11% in comparison with C_1 and C_2 respectively), the memory required by it to complete the compositions was considerable higher (up to 5.7 times higher) than both configurations of *COPERNIC*. One of the main reasons for this significant divergence in memory usage is that GoCoMo's service discovery mechanism uses a cache to store the progress of resolving split-join controls for parallel service flows, which results in resource-intensive processes creating multiple potential execution branches. On the contrary, *COPERNIC* does not keep a record of multiple execution branches and does not store in memory different workflows for each fork of the plan; it keeps in memory only one single plan that is created on-the-fly, that is, when goals or sensory stimuli (internal signals and user requests) change then it adapts to the new situation by spreading more activation to those nodes (e.g., perception, WM, and attentional nodes) that should participate in the new plan, which becomes more attractive than the current plan. Additionally, *COPERNIC* does not replan at every time step. The "history" of the spreading activation also plays a role in the service selection since the activation levels are not reinitialized at every time step (instead, they are accumulated over time, so when changing the focus to another goal, services that may participate in the new goal reuse their current activation and continue accumulating activation on top of it). Furthermore, it is important to highlight that the cost of recomposing is significantly reduced by *COPERNIC* thanks to its distributed nature where multiple agents decompose the whole problem into smaller planning objectives. Similar to the previous experiment, one of the drawbacks of our approach is that its failure rate was higher than GoCoMo's one due to the attention mechanism could dismiss some crucial information pieces at any point affecting the final decision. Now, comparing the results of both configurations of *COPERNIC*, we can observe that in general C_2 outperforms C_1. We can infer that if γ (the amount of energy a goal injects into the attentional mechanism) $> \phi$ (the amount of energy that the WM units inject into the attentional mechanism) then *COPERNIC* will be more goal-oriented and less sensitive to changes in the current state (e.g., changes in mobility). On the contrary, if $\phi > \gamma$ then the system will be more sensitive to changes in the current state rather than changes in the goals. Also, if δ (the amount of activation energy a protected goal takes away from the system) is significantly greater than γ then the system will keep a stubborn position, that is, it will try to always stick with the original plan (protected goal) and will be reluctant to refocus its attention to the new goal. On the contrary, if γ is considerable greater than δ then the system will be continuously switching from one goal to another and never will conclude any plan. Finally, reducing the activation threshold θ may help the system make faster decisions (reactive behavior), useful during time-sensitive composition; by contrast, increasing θ will slow down the reasoning process (deliberative behavior), useful for long-term composition planning. Therefore, the utility learning mechanism has to find a proper ratio between these parameters so the performance of the system is improved. The learning mechanism found a tradeoff between the parameter and discovered that, in order to make the system sensitive to both current-state changes (e.g., mobility, perceived stimuli, etc.) and goal changes, without switching indefinitely between goals, and with the ability to undone previously reached (protected) goals in order to refocus on new goals (replanning), then γ should be slightly greater than δ at a ratio of $\approx$ 4:3; ϕ should be greater than γ at a ratio of $\approx$ 2:1; and $\phi > \pi > \gamma$ where ϕ is greater than π at a ratio of 14:9. When using values beyond those ratios (as C_1 does), *COPERNIC*'s planning and execution failure rates increased considerably in comparison with GoCoMo.

As a side note, the way CoopC (and other baseline service composition models) addresses faults in composition is for the system to restart the whole process if any service has failed during the execution, of course, this solution is unable to utilize the partial results. Unlike CoopC, *COPERNIC* neither creates a long-term plan upfront, nor constructs a search tree. Instead, plans are tailored on-the-fly through activation accumulation, so that it does not have to start from scratch when one path does not produce a solution, but smoothly moves from one composition plan to another. As a result, the computation of the composition plan is much less expensive. Creating a long-term plan in advance would require too much time (especially for a cognitive agent operating in a rapidly changing MPC environment), instead, plans are emergently created by *COPERNIC* as a result of multiple cascading cognitive cycles. The main differences between our approach and both GoCoMo and CoopC are that our approach can mediate smoothly between deliberation and reactivity by determining (through learning) a tradeoff between ϕ, γ, δ, π and θ; and that it can perform deliberative composition by accessing long-term intentions stored in the episodic memory. It is worth noting that there is a multiple correlation between resource consumption, execution failure rate, and planning failure rate, so the more *COPERNIC* filters out the information required for the composition the lesser resources are required during composition, the faster the composite service will be generated and, therefore, the lower the execution failure rate will be. That is, if a composite service can be planned and replanned quickly and without requiring too many resources (as *COPERNIC* does), then the discrepancies between planning and execution will be minimized and, as a consequence, the execution failure rate will be minimized as well. However, the more the information is filtered out the higher the planning failure rate will be due to the cognitive agent may dismiss critical information pieces during planning.

5 Related Work

In the existing literature, there are mainly two different techniques of drafting a composition plan [18]. The first one utilizes the classical planning techniques used in AI (e.g., HTN, petri-nets, state charts, rule-based, multi-agent systems, π-calculus, etc.). Under this approach [7,23], the composition of atomic services into a composite service is viewed as a planning and optimization problem. The second technique uses workflows, in which a composite service is broken down into a sequence of interactions between atomic services [4]. The third technique uses direct mappings between user requests and service descriptions without needing intermmediate representations such as ontologies [22]. In general, first and second approaches either rely on conditional plans and can therefore handle only a limited range of non-deterministic action outcomes, or have the queries about unknown information explicitly included in the predefined service composition procedure. These plans use to be computationally expensive, have to deal with composition length and strive to optimize the resources involved. Our service composition model is not as expensive as traditional approaches because plans are constructed emergently on-the-fly as the result of both the spreading activation dynamics defined at multiple overlays of the system, and the cognitive mechanism for filtering out and focusing on the most relevant information. To reduce composition and execution failures while dealing with complex user requirements, existing service composition techniques investigate flexible composition planning mechanisms, and service execution policies such as: open service discovery approaches and dynamic service planning approaches. A graph-based service aggregation method [1,26] models services and their I/O parameters in an aggregation graph based on the parameter dependence of the services. It dynamically composes services to support a task in a workflow when a direct match between the task and a single service does not exist. Such a workflow may need to be generated offline by a domain expert or a composition planning engine, which is inconvenient when a change is required at runtime. Dynamic service planning approaches such as [12,17] use classic AI-planning algorithms, such as forward-chaining and backward-chaining for dynamic composition planning, and usually employ a bi-direction planning algorithm to find a path with the smallest cost from the dependency graph. However, these approaches require central service repositories to maintain service overlays, and have no support for dynamic composition replanning for composition failures. AI-planning algorithms like Haley [27], and a fuzzy TOPSIS method [6] have been investigated for dynamic composition planning and have features for automatic re-planning to handle failures. However AI-planning algorithms rely on central composition engines that have not yet been applied on mobile devices. In addition, they need to re-generate a new plan for failure recovery, which is time-consuming and not suitable for dynamic environments.

6 Conclusions and Future Work

We described *COPERNIC*, an agent-based model for service composition in MPC environments. Our main contribution is the implementation of a cognitive model that efficiently and dynamically orchestrates distributed services under highly changing conditions. Our approach focuses on bounded rationality rather than optimality, allowing the system to compensate for limited resources by filtering out a continuous stream of incoming information. We tested our model against state-of-the-art service composition models while modifying mobility, service density and composition complexity features, and the promising results demonstrated that a cognitively-inspired approach may be suitable for pervasive environments where resources are scarce and the smart devices have computational and hardware constraints. Our future work will mainly focus on tightly integrating a context-awareness feature into our model so cognitive agents can make more accurate decisions during service selection.

References

1. Al-Oqily, I., Karmouch, A.: A decentralized self-organizing service composition for autonomic entities. ACM Trans. Auton. Adapt. Syst. **6**(1), 7:1–7:18 (2011)
2. Anderson, J.R., Bothell, D., Byrne, M.D., Douglass, S., Lebiere, C., Qin, Y.: An integrated theory of the mind. Psychol. Rev. **111**(4), 1036 (2004)
3. Balzer, S., Liebig, T.: Bridging the gap between abstract and concrete services a semantic approach for grounding owl-s. In: Semantic Web Services (2004)
4. Ben Mokhtar, S.: Semantic Middleware for Service-Oriented Pervasive Computing. Ph.D. thesis, Université Pierre et Marie Curie - Paris VI Dec 2007
5. Chen, N., Clarke, S.: Goal-driven service composition in mobile and pervasivecomputing. Serv. Comput. **11**(1), 49–62 (2018)
6. Cheng, D.Y., Chao, K.M., Lo, C.C., Tsai, C.F.: A user centric service-oriented modeling approach. World Wide Web **14**(4), 431–459 (2011)
7. Davidyuk, O., Georgantas, N., Issarny, V., Riekki, J.J.R.: MEDUSA: middleware for end-user composition of ubiquitous applications. In: Ambient Intelligence
8. Furno, A.: Efficient cooperative discovery of service compositions in unstructured p2p networks. In: Parallel Processing (2013)
9. Ibrahim, N., Mouël, F.L.: A survey on service composition middleware in pervasive environments. CoRR abs/0909.2183 (2009)
10. Immonen, A., Pakkala, D.: A survey of methods and approaches for reliable dynamic service compositions. SOCA **8**(2), 129–158 (2014)
11. Kanerva, P.: Sparse Distributed Memory. MIT Press, Cambridge (1988)
12. Khakhkhar, S., Kumar, V., Chaudhary, S.: Dynamic service composition. CS and AI **2**(3), 32–42 (2012)
13. Laird, J.E., Lebiere, C., Rosenbloom, P.S.: A standard model of the mind: toward a common computational framework across artificial intelligence, cognitive science, neuroscience, and robotics. AI Mag. **38**(4), 13 (2017)
14. Maes, P.: How to do the right thing. Connection Sci. **1**(3), 291–323 (1989)
15. Michael, P.: Cognitive Neuroscience of Attention. Springer, New York (2011)
16. Miller, G.A.: The magical number seven, plus or minus two: Some limits on our capacity for processing information. Psychol. Rev. **63**(2), 81–97 (1956)
17. Oh, S., Lee, D., Kumara, S.R.T.: Effective web service composition in diverse and large-scale service networks. IEEE Trans. SC **1**(1), 15–32 (2008)
18. Raychoudhury, V., Cao, J., Kumar, M., Zhang, D.: Middleware for pervasive computing: a survey. Pervasive Mob. Comput. **9**(2), 177–200 (2013)
19. Romero, O.: An evolutionary behavioral model for decision making. Adapt. Behav. **19**(6), 451–475 (2011)
20. Romero, O.J.: CogArch-ADL: toward a formal description of a referencearchitecture for the common model of cognition. Procedia Comput. Sci. **145**, 788–796 (2018)
21. Romero, O.J., Akoju, S.: An efficient mobile-based middleware architecture for building robust, high-performance apps. In: ICSA. In press (2018)

22. Romero, O.J., Dangi, A., Akoju, S.: NLSC: unrestricted natural language-based service composition through sentence embeddings. In: SCC (2019)
23. Santofimia, M.J., Fahlman, S.E., del Toro, X., Moya, F., López, J.C.: A semantic model for actions and events in ambient intelligence. AI **24**(8), 1432–1445 (2011)
24. Stavropoulos, T.G., Vrakas, D., Vlahavas, I.: A survey of service composition in ambient intelligence environments. Artif. Intell. Rev. **40**(3), 247–270 (2013)
25. Tomazini, L., Romero, O.J., Hruschka, H.: An architectural approach for developing intelligent personal assistants supported by NELL. In: ENIAC (2017)
26. Wang, Z., Xu, T., Qian, Z., Lu, S.: A parameter-based scheme for service composition in pervasive computing environment. In: CISIS, pp. 543–548 (2009)
27. Zhao, H., Doshi, P.: A hierarchical framework for logical composition of web services. Serv. Oriented Comput. Appl. **3**, 285–306 (2009)
28. Zhao, R., Romero, O.J., Rudnicky, A.: SOGO: a social intelligent negotiation dialogue system. In: Intelligent Virtual Agents (IVA) (2018)