Journal of
Sensor and Actuator Networks

Article

Learning-Based Coordination Model for On-the-Fly Self-Composing Services Using Semantic Matching

Houssem Ben Mahfoudh †, Ashley Caselli † and Giovanna Di Marzo Serugendo *,†

Centre Universitaire d'Informatique, University of Geneva, Route de Drize 7, CH-1227 Carouge, Switzerland; houssem.benmahfoudh@unige.ch (H.B.M.); Ashley.Caselli@unige.ch (A.C.)

* Correspondence: Giovanna.dimarzo@unige.ch; Tel.: +41-22-379-00-72

† These authors contributed equally to this work.

Abstract: Forecasts announce that the number of connected objects will exceed 20 billion by 2025. Objects, such as sensors, drones or autonomous cars participate in pervasive applications of various domains ranging from smart cities, quality of life, transportation, energy, business or entertainment. These inter-connected devices provide storage, computing and activation capabilities currently under-exploited. To this end, we defined "Spatial services", a new generation of services seamlessly supporting users in their everyday life by providing information or specific actions. Spatial services leverage IoT, exploit devices capabilities (sensing, acting), the data they locally store at different time and geographic locations, and arise from the spontaneous interactions among those devices. Thanks to a learning-based coordination model, and without any pre-designed composition, reliable and pertinent spatial services dynamically and fully automatically arise from the self-composition of available services provided by connected devices. In this paper, we show how we extended our learning-based coordination model with semantic matching, enhancing syntactic self-composition with semantic reasoning. The implementation of our coordination model results in a learning-based semantic middleware. We validated our approach on various experiments: deployments of the middleware in various settings; instantiation of a specific scenario and various other case studies; experiments with hundreds of synthetic services; and specific experiments for setting up key learning parameters. We also show how the learning-based coordination model using semantic matching favours service composition, by exploiting three ontological constructions (is-a, isComposedOf, and equivalentTo), de facto removing the syntactic barrier preventing pertinent compositions to arise. Spatial services arise from the interactions of various objects, provide complex and highly adaptive services to users in seamless way, and are pertinent in a variety of domains such as smart cities or emergency situations.

Keywords: self-composition; coordination model; semantic middleware; connected objects; spatial services; reinforcement learning; on-demand services; collective adaptive system; connected objects; seamless interaction

Citation: Mahfoudh, H.B.; Caselli, A.; Serugendo, G.D.M. Learning-Based Coordination Model for On-the-Fly Self-Composing Services Using Semantic Matching. *J. Sens. Actuator Netw.* **2021**, *10*, 5. https://doi.org/10.3390/jsan10010005

Received: 30 September 2020
Accepted: 15 January 2021
Published: 20 January 2021

1. Introduction

In his seminal and visionary paper Mark Weiser [1] describes the notions of "disappearing computers and disappearing technology", where connected objects are "embedded in the everyday world" and made invisibly available to the users. The latter then "use them unconsciously to accomplish everyday task", such as obtaining information, requesting actions, or animating otherwise inert objects-in other words, providing an "embodied virtuality". According to Weiser, the "real power of the concept comes from the interaction of all of the devices".

Today, thirty years later, connected objects are dramatically increasing in number, and more than 20 billion more objects are expected to be connected in the next five years. They relate to many domains such as smart city, smart building, transportation, energy, business, or quality of life. Internet of things (IoT) allows these devices to communicate, collaborate, gather data, and take decisions. Sensors and actuators can be locally connected via gateways or edge nodes and provide massive storage, computing and acting capabilities currently under-exploited, but available in a pervasive, and invisible manner in our everyday life. Weiser's vision takes a step closer to reality.

In this context, our research tackles Weiser's challenge above, namely we *provide complex services to the users, arising from seamless and spontaneous interactions among heterogeneous devices and connected objects, while hiding this complexity to the user*. We coined this new type of complex services and applications "Spatial services" [2,3]. They form a new generation of services, built and spontaneously composed on-demand, arising from the collective interactions of connected objects. Spatial services are spatially and temporally distributed over a geographic area, corresponding to the environment where the objects themselves reside or on which they act. Spatial services are built dynamically through collaboration with other services and composed on-demand for their users (providing information or acting on the environment). Spatial services arise from the interactions of multiple sensors and devices, working together as a decentralised collective adaptive system. They are issued from devices heterogeneous in nature (sensors, actuators, autonomous cars, etc.), which consume and produce data, act on their environment, in a fully distributed manner, without prior knowledge about each other, and without pre-designed compositions. In this setting, we consider that a connected object provides a set of services, enabled by its capabilities, such as sensing, acting on its environment and communicating with other objects.

The spatial services paradigm relies on a learning-based coordination model [2,4], where dynamic compositions arise, fully automatically and spontaneously at run-time exploiting IoT devices, with seamless adaptation to arriving and departing services.

In our previous works, composition of services relies on syntactic matching [2]. In this paper, we extend our learning-based coordination model with semantic reasoning, increasing the range of compositions and users' interactions that our system supports, still supporting dynamicity and lack of knowledge among services, and providing highly adaptive and complex services to the user. The implementation of our model, discussed in this paper, leads de facto to a semantic middleware for connected objects. Spatial services-implemented through a learning-based coordination model and using semantic matching-provide a highly adaptive paradigm allowing the exploitation of the collective capabilities of connected objects, themselves evolving in a dynamic and changing environment. They constitute an additional brick towards Weiser's vision of users unconsciously supported by connected objects in their everyday life.

Section 2 discusses related works. Section 3 summarises our learning-based coordination model approach to self-composition. Section 4 shows how we extend the above model with semantic matching and reasoning. Section 5 discusses the semantic middleware implementation, its deployment, and a series of experiments in various scenarios. Finally, Section 6 provides some discussion on the overall approach and Section 7 concludes the paper.

2. Related Works

Services composition. Service composition is commonly known as the mechanism that combines the functionalities of two or more basic services and creates a more complex one. It aims to exploit the available services to meet the user's requirements [5]. Service composition approaches may be categorised in terms of many orthogonal properties [6]. A possible grouping may be defined using the stage during which the actual composition, namely the services binding, occurs. Such classification divides the service composition approaches in: (i) static composition, and (ii) dynamic composition. Static composition approaches include orchestration and choreography [7]; both approaches use pre-defined composition arrangements. The resulting services depend on these pre-designed compositions. These approaches lead to correct compositions but lack dynamicity and adaptation, and lead to reduced robustness and fault-tolerance in case services that have to take part in the composition are not available or reliable at the moment of the composition. Therefore, their static character has been challenged by approaches involving dynamic service composition, in order to ensure scalability and adaptability, even though these new approaches add computational overhead to the system.

An alternative grouping, for the service composition approaches, may be defined using the composition policy they adopt: (i) syntax-based, the matching among services is driven by evaluating the syntactic equality among the service input/output parameters; (ii) semantic-based, the composition process computes the matches relying on concepts classification and their relationships (i.e., a taxonomy); and (iii) other solutions, e.g., AI-planning techniques.

Syntax-based composition approaches rely on the equality evaluation among services parameters. In this case the functional aspects of services are described through a representation that is not exploitable by any machine. Such a representation does hold an intrinsic meaning that cannot be automatically understood by machines. For instance, let us consider two services that provide exactly the same functionalities described in a syntactically different manner, there is no way for an automated process to evaluate them as equal, as there would be for a human being. Therefore a syntax-based composition approach brings several limitations in an automated composition process since it reduces the matching algorithm to a mere equality evaluation among words. In order to improve the service composition matching a semantic machine-readable description of the service and its functionalities would be needed.

The birth of the semantic web and its technologies doubtlessly brought enhancements that increased the possibility to perform automated computations in many fields. The machine-readable representation with which a service may be described allows semantic-based composition approaches to be adopted. Such processes, relying on existing or ad-hoc concepts taxonomies, are able to perform a semantic reasoning on the service functionalities and generate the most appropriate composition graph, according to the defined policies. Among the above-named categories, we can mention the following works. From the semantic web field, Talantikite et al. [8] present a model for automatic Web services discovery and composition. Such model exploits web services that have previously been semantically annotated through an ontology (i.e., OWL-S [9]). Talib et al. [10] provide a semi-automatic method to generate static web service composition in BPEL4WS language. Tzortzis et al. [11] extend the SYNAISTHISI platform [12] to allow semi-automatic composition of services provided by devices. The SYNAISTHISI platform itself is a message-oriented middleware, providing tools for administrators facilitating the integration and coordination of devices and algorithms. They are presented as semantic web services and registered in a RDF triple-store. Other works focus on the description of ontologies well suited for capturing IoT scenarios on the Web, such as [13], itself deriving from an ontology for describing "Things" in a more general manner [14].

Semantic middlewares or engines. Gyrard et al. [15] propose a semantic engine equipped with various domain ontologies for various devices types, a converter annotating data, a reasoning engine and a query engine. The specificity of this proposal is that it can be deployed in various cases: in cloud, into mobile devices, or inside gateways. Zgheib [16] provides a message-oriented semantic middleware, well-suited for healthcare applications. It is based on the publish-subscribe paradigm. In addition to the publishers and subscribers, specific transformers wrap the physical devices transforming them into virtual semantic ones. The latter interact with the publish-subscribe broker. More generally, Zgheib et al. [17] provide an overview of semantic middleware for IoT, identifying various cases, such as message-oriented, tuple-based, agent-based, database-oriented, or with service-oriented architecture.

Coordination and agent-based approaches. We can mention various approaches favouring dynamic compositions of services or components at run-time: from more static ones, such as workflow-based approaches and AI Planning [18], to more dynamic ones based on a central coordinator [19], to more decentralised ones, involving channels for devices to send and receive messages [20]. The approach of Vallee et al. [21] combines multi-agents with semantic web services and provides dynamic context-aware service composition. Other approaches in the same field usually involve planification techniques, where agents reason on their services and the users' needs [22]. Works on self-composition of method fragments bring a more dynamic approach involving cooperative agents, each representing a fragment of the composition [23]. Using similar cooperative principles, Degas [24] proposes a syntax-based composition approach with collaborative agents for dynamic composition of aerial plane trajectories. Viroli [25] presents a tuple- and syntax-based approach that involve the notion of competition among services. Frei et al. [26] exploit chemical reactions to produce self-designing industrial assembly systems. Di Napoli et al. [27], always using chemical reactions, dynamically instantiate specified workflows. De Angelis [28] proposes a chemical-inspired model that promotes self-composition of services at run-time adopting a syntax-based composition approach. Finally, Ben Mahfoudh et al. [4] extend the original tuple space model with learning-based capabilities in order to accommodate self-composition.

Learning-based approaches. Inspired by cognitive sciences, learning approaches remove pre-defined goals and objective functions [29]. The Self-Adaptive Context-Learning (SACL) Pattern [30] involves a set of collaborative agents learning contexts and mapping agents' perceptions with actions and effects. Reinforcement learning solutions are suitable for problems or search space modelled into a Markov decision process (MDP) [31]. In the case of pre-defined workflows, multi-agent reinforcement learning [32] allows selecting the most reliable services to be used in the workflow. Other works involve semi-dynamic solutions using AI planning methods [18] without self-composition at run-time.

Wang et al. [33] approach is very close our learning-based proposal. Both QoS and multi-agent reinforcement learning serve as a basis for identifying the workflow with the best cumulative reward. To alleviate the limitations of their approach in large-scale scenarios with many state-actions pairs, Wang et al. [33] suggest using deep Q-learning to improve prediction and efficiency [34,35]. A full review of service composition approaches for internet of things is presented by Aoudia et al. [36].

We propose a tuple-based, learning-based semantic model and its middleware implementation. It is completed with learning agents working on behalf of devices, applications, or users. Our proposal combines: (1) complex adaptive services to users arising from the interaction of connected devices; (2) fully automatic and spontaneous composition of services (or capabilities) provided by the devices; (3) a semantic composition based on a specific domain ontology; (4) learning agents receiving rewards from other agents, and forwarded to them by the middleware; (5) services come with QoS, which further serve selecting reliable services. This last point is not discussed in this paper, but fully exposed in [4]. Compared to the above works, event though some of the presented approaches provide a dynamic and/or semantic composition, none of them combines a decentralised architecture, without pre-designed composition, and a semantic and learning approach.

3. Service Composition

This section discusses the different concepts of our learning-based coordination model, and describes a running example using learning and syntactic matching that will be revisited in the next section. This work is based on a previous work [2,4] that we will briefly describe and reproduce here to allow the reader to understand the remainder of the paper.

3.1. Coordination Model

Coordination models provide a general solution to coordinate agents execution. They are not specifically meant for self-composition. The most suitable ones for composition are those providing tuples and a blackboard (favouring asynchronous coordination) and using a chemical inspired approach (favouring spontaneous coordination). Our proposal is inspired by nature (stigmergy and blackboard) and does not involve formal aspects. It can be deployed on several heterogeneous nodes and it supports spontaneous composition. It provides autonomous coordination between entities and offers them the capability to update their behaviour regarding their local environment.

Coordination models inspired by nature and dealing with connected objects such as SAPERE [37], ASCENS [38] and TOTA [39] may be used to compose services and provide reliable solutions at run-time. We have chosen the Self-Aware Pervasive Service Ecosystem (SAPERE) project as it is well designed, easy to use and offers a detailed documentation and open-source implementation. Indeed, previous works demonstrate [40–42] that SAPERE is suitable for designing spatial services and self-composition.

The SAPERE model, shown in Figure 1 (black part only) is a coordination model for multi-agent pervasive systems. It is inspired by chemical reactions and composed of the following concepts:

- Software Agents (i.e., coordination entities): active software entities that act as an interface between the tuple space and the real world. An entity could be any sort of device (e.g., sensors or actuators), application or service;
- Live Semantic Annotations (LSA): tuples of data and properties which are managed and updated by the software agent (e.g., a property value is updated when the sensor updates its value);
- Tuple space: shared space (i.e., coordination media) that hosts all the tuples in a node. A node could be a Raspberry Pi, a smartphone, or any connected object that can host a shared space;
- Eco-laws (i.e., coordination laws): chemical-based coordination rules derived from bio-inspired mechanisms, dynamically acting on LSAs (see below).
- Operations: A set of operations (e.g., inject a new LSA, update an LSA's content or remove an LSA) that are executed by the system.

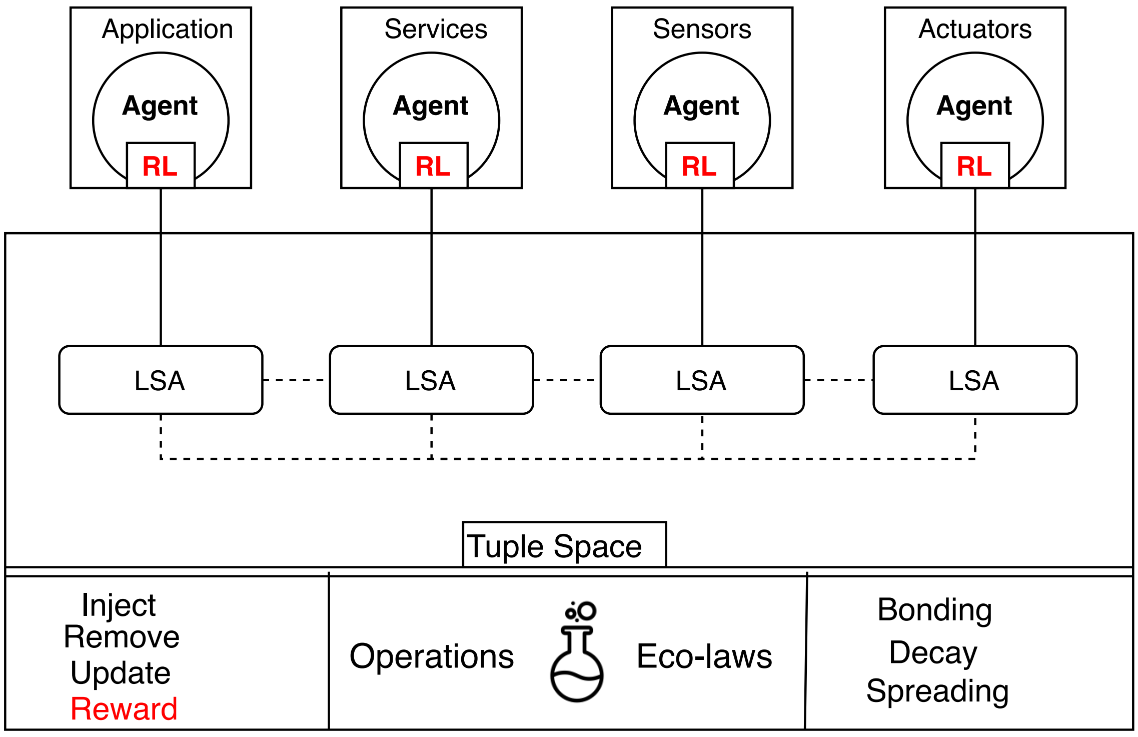


Figure 1. Learning-based coordination model (derived from the SAPERE [37] coordination model).

The coordination model offers a set of bio-inspired mechanisms [43], under the form of eco-laws (or coordination laws) that can be used in different domains like optimisation, heuristic search or solving computational problems. We are interested in the following ones for the rest of this work as they meet our needs, and serve to solve our problem:

- Bonding: links an agent with data provided by another agent that it was waiting for, referred to, concerns it, etc.
- Decay: (or evaporation) is a pattern used to mark the relevance of the information located in the tuple space. It regularly decreases the relevance of the data and ultimately removes outdated data.
- Spreading: similar to broadcast as it diffuses information within a network. However, the spread is done with a fixed propagation hops.
- Gradient: is build on the spreading pattern. It aggregates data using some algebraic operation (e.g., min, max or avg) and offers additional information about hops distance.

Software agents follow their own logic and may be of different types, reactive, intelligent, equipped with reasoning, etc. In any case, the minimum actions taken by the agents consist in: (1) inserting data (or queries) incoming from the device (or the user) (for which they work) into the tuple space (e.g., a new value for a sensor is available), and/or vice-versa, (2) instructing the device to perform some action upon some event (i.e., a bonding) incoming from the coordination media (e.g., performing some computation based on some input value, or taking some action in the physical world, or providing some information to the user). Actions taken in response to bonding events lead the concerned devices in taking part in the corresponding compositions, de facto producing some collective result.

Figure 1 shows a single computational node to which several agents are directly connected. This could be the case in a household, where all devices are connected through their own agent to the same computational node. SAPERE, as well as our extensions, works in a fully distributed and decentralised manner across geographical and spatial zones. Compositions can span several computational nodes connected together, each equipped with the middleware implementing the model, each having agents attached to them. Hence the term "spatial services" for the corresponding compositions. Section 5.3 discusses deployment options.

3.2. Data Structure

In order to accommodate dynamic composition, we extended the original LSAs from SAPERE as follows. A live semantic annotation or *LSA* is a tuple of data and properties. Its value could change with time. An *LSA* is controlled (injected, removed, updated) by the software agent. It is of the following form:

$$LSA ::== \{S = [svc_1, \ldots, svc_m], P = [P_1, \ldots, P_n]\}.$$

$\mathcal{P}$ is a set of property names and $svc_j \in \mathcal{P}$ are property names to which the agent wants to bond with. The agent is notified when a property that corresponds to service svc_j is injected or updated into the Tuple space.

1. A set of *Service properties* or property names which we note S: A software agent is sensitive to some property name or input to which it wants to be alerted when they become available.
2. A set of *Properties* which we note P: Each software agent provides a set of properties or output which correspond to the service that it provides. It is defined by:

$$P_i ::== \{< key_i : v_i >, < \#B_i : BondedAgent >,$$
$$< \#Q_i : QueryAgent >, < \#C_i : Schema >, \#True/False\}$$

   where:

   - key_i: is the property name, $key_i \in \mathcal{P}$.
   - v_i: is the value of property key_i.
   - $\#B_i$: the id of the agent that manages the LSA to which a property key_i bonded.
   - $\#Q_i$: the id of the agent that is at the origin of the request.

- $\#C_i$: a sequence of property names representing the composition schema. The requested property names are separated by a vertical line "|" from the provided properties during composition.
- $\#True/False$: a flag that indicates if a property key_i has been consumed (#True) or not (#False) by other agents.

3. A set of *Synthetic properties* that contains some features related to the operation of the middleware. It is not shown in the above LSA representation for presentation concerns.

3.3. Learning-Based Coordination Model

In this section, we present how we extended the SAPERE model with learning. To do so, we equipped the agents with a Reinforcement Learning module (RL) and provided a new operation in the coordination model, called Reward (the red elements in Figure 1). The RL module allows the agents to regulate the outcomes of the Bonding eco-law. Depending on their current learning tables, they will or will not be sensitive to a bonding notification. The Reward informs the agent about the positive or negative outcome of their involvement in a service composition.

Reinforcement learning (RL) algorithms provide a suitable solution for optimisation and decision making under uncertainty in sequential decision problems. QLearning [44] is one of the RL algorithms that allows agents to learn a policy that maximises rewards by taking action when being in a given state. A Markov decision process (MDP) [45] models our problem as a set of finite actions and states. When a bonding happens, an agent has to select the best action, from the set of actions available to him given its current state.

Our model is defined by:

- **States** S: a set of all possible composition schemas. A state is updated in the #C composition property attribute of the LSA.
- **Actions**: $A = \{Ignore, React\}$;
  - Ignore the bonded LSA: useless bonding are avoided.
  - React to the bonded LSA: a new property is added or updated in the agent's LSA, as the result of the internal computing of the agent following the bonding.
- **Reward**: A positive or negative reward is attributed to all agents that participated in a successful composition (with final composition schema). The user feedback depends on the actual relevance of the result [46].
- **Exploration algorithm**: ϵ-greedy [47]. This algorithm has a probability ϵ to select a random action and a probability $1 - \epsilon$ to select the action that maximises the value of the approximation of $Q(s, a)$.
- **Q function**: $Q : S \times A \rightarrow \mathbb{R}$, where:
  $Q^i_{t+1}(s_t, a_t) = Q^i_t(s_t, a_t) + \alpha \times (R^i_t + \gamma \times max_a Q^i_t(s_{t+1}, a) - Q^i_t(s_t, a_t)), \forall i \in \{1, \ldots, n\}$,
  where n is the number of agents that participated in the service self-composition, t is the current time, s_t is the state at time t in which the agent took action a_t, s_{t+1} is the next state reached by the agent after taking action a_t, α is the learning rate and γ is the discount factor that determines the cumulative discounted future reward.

Our system has to learn the best action to take in order to decide on the appropriate action in response to a notification received from the bonding eco-law. This helps to optimise the overall system behaviour and to provide the most relevant and reliable self-composed services. Agents will collaborate and coordinate together to produce all possible compositions including inconclusive or useless ones for the end-user. They will progressively update their behaviours by following the RL module. Our system converges towards pertinent and efficient composition, i.e., the ones actually expected by the user.

In the learning-based extension, software agents as before follow their own logic, however now, once notified of an event (i.e., through a bonding) they learn whether they need to react to that bonding or simply ignore it. Reacting to a bonding arising from a composition means that the device controlled by the agent is solicited in a composition, and some action and/or result is expected from it. Learning whether or not it is pertinent to enter into a composition is important to prevent participating in compositions that in the end remain partial or do not satisfy the user or the end-system. In cases with a large number of services and possible compositions, this also may reduce the number of activated compositions.

3.4. Composition Schema

A composition schema is a sequence of properties names that have been consumed during the services' composition. In Figure 2, we suppose that a query is injected providing a Property of name "A" as input and requesting a Property of name "D" as output. We define two different composition schemas. We say the composition schema is *partial* when the input property is present but the requested output property is not yet reached. We say the composition schema is *final* when it starts with the input property and ends with the output property. The requested properties and the composition schema are separated by a vertical bar |.

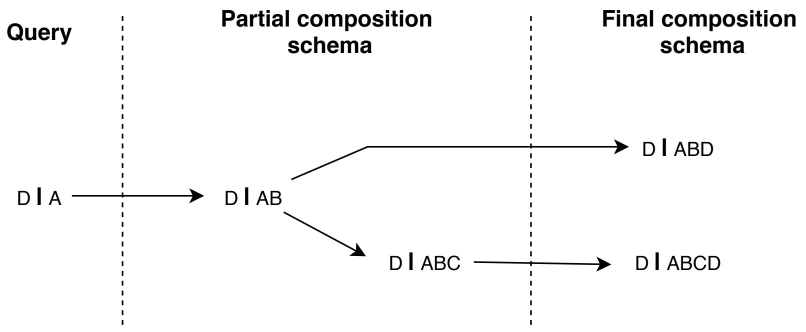


Figure 2. Composition schema: a sequence of property names representing the composition schema.

All objects are connected-in a peer-to-peer network, in a single node or through a cloud-this allows them to interact and cooperate. Our solution supports multiple nodes, thanks to the use of the spreading mechanism. When an agent bonds, a new property is added or updated. A copy of its LSA is automatically spread throughout the network. All tuple spaces will be updated and agents will be able to bond with these new properties. All copies are dynamically deleted using the Decay eco-law after a short time.

3.5. Reward

Our system uses an RL module that needs feedback from users to adapt agents' behaviours. A user (e.g., a human being or a system) is able to provide a feedback regarding the provided result. The provided feedback is propagated back to all agents that participated in the composition. Agents update their Qmatrix using the Reward information R. Positive rewards are attributed to the agents that provide pertinent and efficient services while negative rewards are attributed to the agents that did not participate in a final schema or did not successfully provide any results. This helps to avoid unnecessary bonding and executions.

A sparse or gradient reward [46] might be a solution to avoid long composition schema as further partial composition schema are less rewarded. In Reinforcement learning, reward functions depend on the problem and are tricky to choose. In this paper, we chose to use for the positive reward the value $R = +10$ and for negative reward the value $R = -10$.

Each agent has a Q matrix that corresponds to all partial and final schema received during execution. In order to push agents to react, at the beginning of the process, we initialise the *Ignore* action with 0 and the *React* action with 5. The actions value are then updated after each received reward R (Section 3.3). When the agent bonds, it will check its Qmatrix to decide the appropriate action to take for each partial composition.

Sometimes, an agent may not receive any feedback although it reacts to the partial composition. The composition may not be achieved (i.e., remains partial) or the user may not give any feedback. Thus, to avoid all unnecessary bonding, the agent will receive an internal negative reward when it randomly decides to ignore the bonding. After a few iterations, all agents that provide useless properties will automatically learn not to react to the partial composition that did not lead to a result.

3.6. Scenario

Let us consider the following scenario. A user arrives at a gas station, drives an electric car and wishes to find an electric parking spot to park and charge their car, possibly booking also the parking spot at the same time. To do so, they will simply utter a query mentioning their location and the keyword electric car. More technically, this scenario considers that electric vehicles are increasing in number and are able to communicate with several connected objects. These objects form a communication network and cooperate together to achieve a specific task. We consider that connected objects provide or request services (information or actual actions in the physical world) using a syntactic matching (i.e., using keywords and exact syntactic matching to identify matching services and queries).

In this scenario, we consider the following entities (see Figure 3):

- A user driving an *Electric car* is looking for an electric parking sport. The *Electric car* makes a query for an electric parking spot, on behalf of the user.
- A *Gas station* provides a set of services, such as restaurant, gas pump, or electric parking spot. The *Gas station* advertises itself (globally) as a gas station.
- A *Booking service* checks the availability of a parking spot and books it for the user. It waits for requests for parking spots.

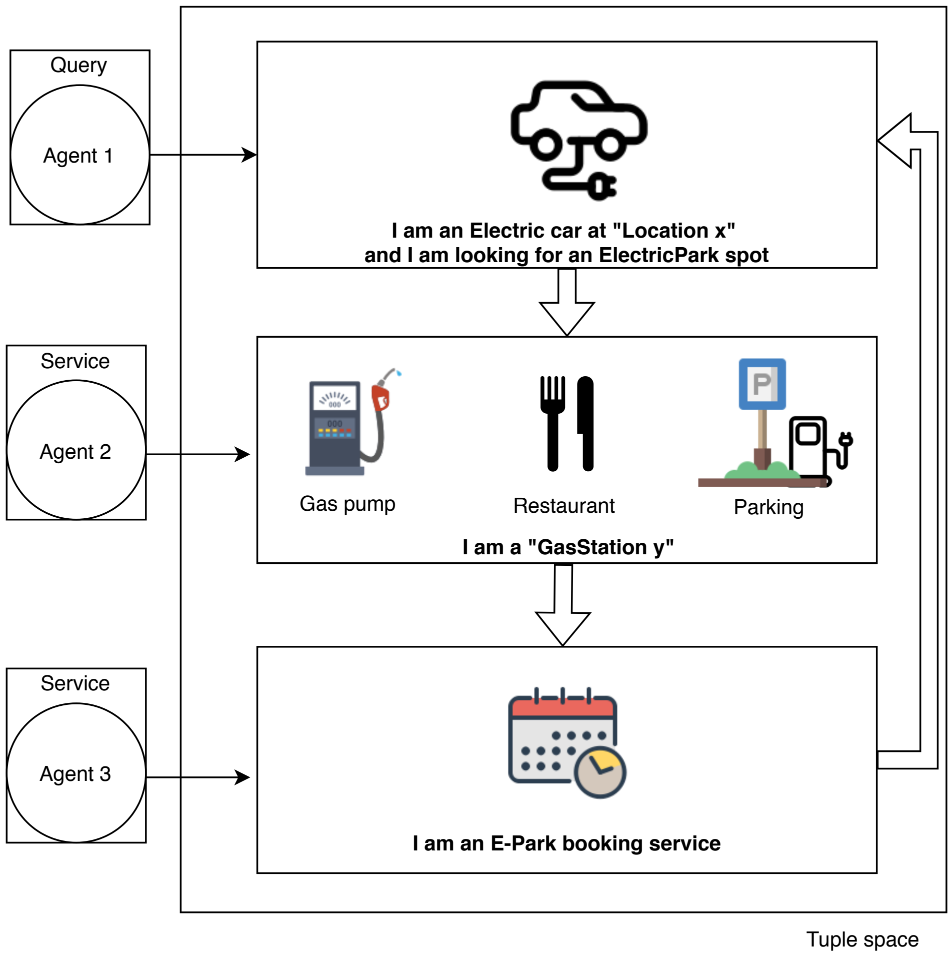

Query
Agent 1
Service
Agent 2
Service
Agent 3
I am an Electric car at "Location x"
and I am looking for an ElectricPark spot
Gas pump
Restaurant
Parking
I am a "GasStation y"
I am an E-Park booking service
Tuple space

Figure 3. Scenario of a spatial service providing identification and booking of an electric car parking spot in a gas station.

The resulting spatial service consists in both providing the information about the existence and availability of an electric car spot within the gas station, and a concrete action consisting in booking the spot in question for the user. The spatial service arises from the seamless interactions of the electric car, the gas station and the booking service, and depends on the currently available services. Indeed, the booking service, in this scenario, provides a richer service than necessary, since it not only identifies electric parking spots, but also provides a booking for them.

Agents

In this scenario, we define three agents, acting on behalf of the three entities of our scenario, in the following manner (see Figures 3 and 4):

- Agent_1: is the query agent. It works on behalf of the *Electric car*. It injects in the tuple space an LSA, with a property P with name "Location" and requests a property S with name "ElectricParkSpot".
- Agent_2: is sensitive to the "Location" property. It works on behalf of the *Gas station*. Internally, it controls a set of services as it represents a series of Gas station services like the parking, restaurants and gas pump as shown in Figure 3. Agent_2 injects an LSA advertising itself with property P, as a "GasStation", and waits for a query corresponding to a location indicated in property S. In this example, we are interested in the internal service of the gas station that provides parking.
- Agent_3: is a parking booking service specialising in electric parking spots. It works on behalf of the *Booking service*. Its LSA will bond with a request for any output with the "Parking" property and informs that it provides a specific electric parking spot, by providing the "E-Park" property P.

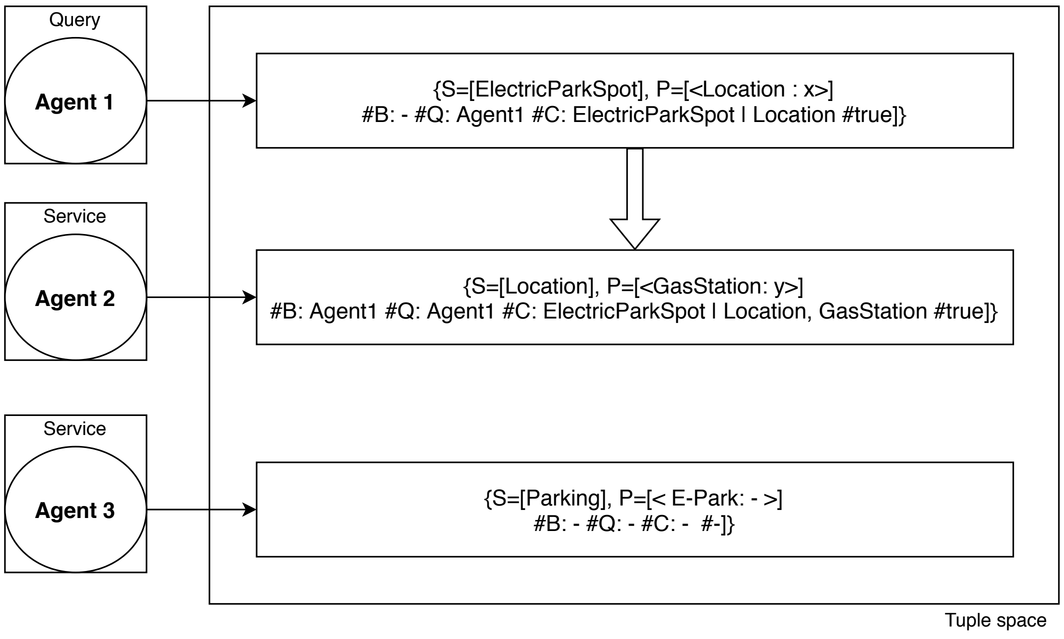

Query
Agent 1
Service
Agent 2
Service
Agent 3
{S=[ElectricParkSpot], P=[<Location : x>]
#B: - #Q: Agent1 #C: ElectricParkSpot I Location #true]}
{S=[Location], P=[<GasStation: y>]
#B: Agent1 #Q: Agent1 #C: ElectricParkSpot I Location, GasStation #true]}
{S=[Parking], P=[< E-Park: - >]
#B: - #Q: - #C: - #-]}
Tuple space

Figure 4. Syntactic composition diagram of the electric car parking scenario.

3.7. Syntactic Composition

The bonding mechanism, as defined in Algorithm 1, using syntactic matching between LSAs has few weaknesses, such as the case sensitivity or the impossibility to recognise synonyms thus cannot create matching that would be correct. The former weakness may be solved during the implementation phase, whilst the latter is impossible to solve with a syntax-based approach.

Algorithm 1 Syntactic Bonding

```
initialisation
1: for $outLSA \in LSAs$ do
2:   for $inLSA \in LSAs$ do
3:     if (syntacticMatching($outLSA, inLSA$) and shouldBond($outLSA, inLSA$)) then
4:       bondLSAToLSA($outLSA, inLSA$)
5:     end if
6:   end for
7: end for
```

Figure 4 shows the different LSAs corresponding to the different agents, as discussed above. In this particular case, through syntactic matching, only one bonding can happen. Indeed, property S of Agent_2 bonds with property P of Agent_1 (on the same "Location" keyword). However, even though the gas station at that location can offer the electric parking spot, no further bonding can happen. Agent_1 S property cannot bond with Agent_3 P property because they do not use exactly the same wording ("ElectricParkSpot" and "E-Park" are synonyms but not considered to be the same). Furthermore, no further relation (outside the location) can be made between the gas station and the electric car (looking for an electric park spot at that location), or between the gas station and the booking system. This is a case where the system ends up with a partial schema. This case highlights three issues with the syntactic matching: (1) synonyms cannot be used or consumed during bonding; (2) no reasoning allows to consider that an electric parking spot is, in fact, a parking spot; (3) no reasoning allows to derive that a parking is part of a gas station.

The Qmatrix of Agent_2 and Agent_3 are presented below in Table 1. No rewards are provided by the user as the system did not succeed to return any result that corresponds to the requested property.

Table 1. Qmatrix of the agents after the first query (syntactic version).

Agent	Schema	Ignore	React
Agent_2	ElectricParkSpot | Location,GasStation	0	5
Agent_3	-	-	-

4. Enhancing Self-Composition with Semantic Matching

The example presented above is here revisited using a semantic-based approach. We update the bonding algorithm above, and present the new version in Algorithm 2, in order to integrate semantic matching of services' properties.

Algorithm 2 Semantic Bonding

```
initialisation
1: for $outLSA \in LSAs$ do
2:   for $inLSA \in LSAs$ do
3:     if (semanticMatching($outLSA, inLSA$) and shouldBond($outLSA, inLSA$)) then
4:       bondLSAToLSA($outLSA, inLSA$)
5:     end if
6:   end for
7: end for
```

Figure 3 shows how a "GasStation" element may be composed of various services: restaurant, gas pump and electric car spots. A formal definition of the former is required in order to be able to perform semantic reasoning on it. Its formalisation is shown in Figure 5, where the relations between each component of the *GasStation* are also specified. Briefly, a *GasStation* may be defined as an object that is composed of many elements, i.e., *Restaurant*, *GasPump*, *Parking*. In turn, the elements may be defined as a composition of others, i.e., a *Parking* is the composition of one (or many) *ParkSpot*. Figure 5 also shows a possible hierarchy of different kinds of *ParkSpot* (Electric or Accessible), as well as the synonyms *E-Park* and *ElectricParkSpot*.

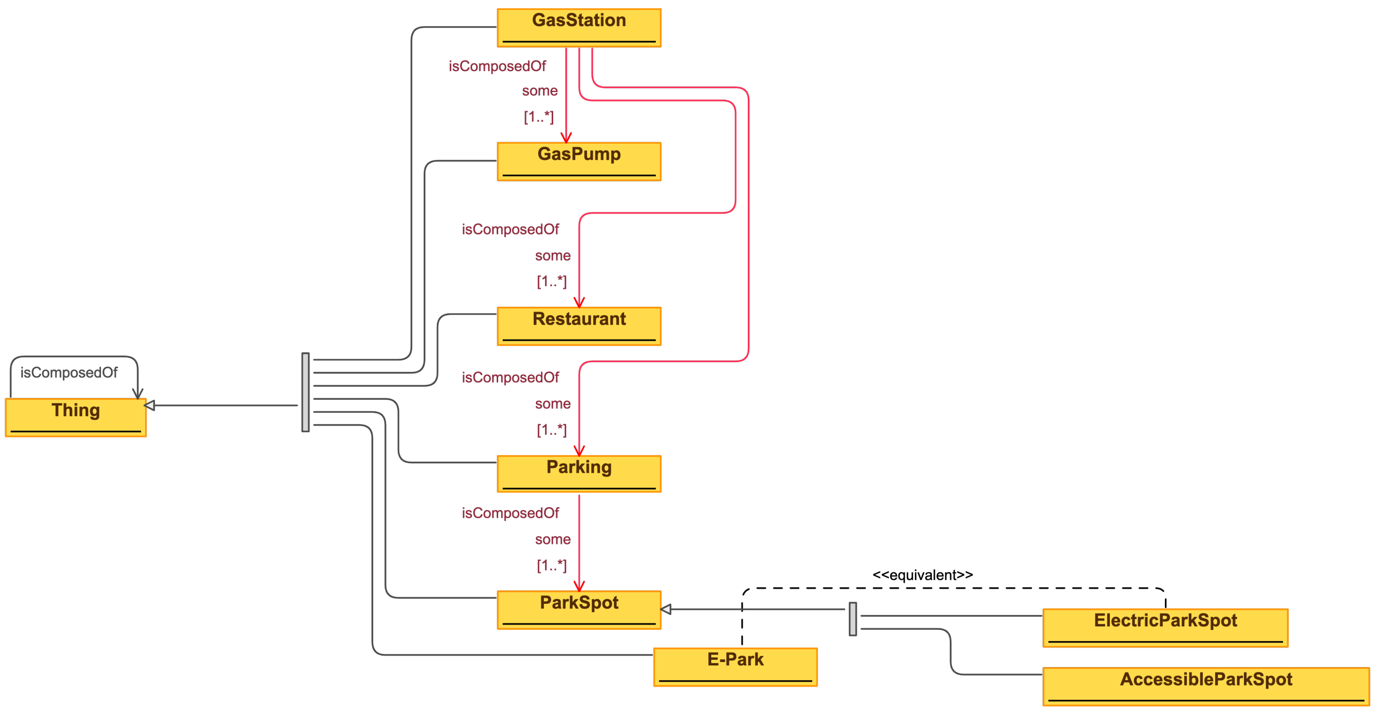


Figure 5. A taxonomy of concepts related to a generic Gas Station.

4.1. Scenario Revisited

Let us again unroll our scenario where an electric car, through a query, requests a property S named "ElectricParkSpot" in output and provides a property P named "Location" as the input. Thanks to the added semantic matching feature, and the corresponding reasoning, the following behaviour compositions can now arise. Figure 6 shows the different LSAs corresponding to the different agents and the bonding events.

Step 1: Agent_1, working on behalf of the electric car, injects the query above as an LSA, asking to book an electric parking spot at a location nearby.

Step 2: A copy of Agent_1 LSA spreads in the network.

Step 3: Agent_2 bonds with the "Location" property, executes the service corresponding to this query and updates its LSA by adding the "GasStation" property (Figure 6-arrow between LSA of Agent_1 and Agent_2, and update of Agent_2 LSA).

Step 4: A copy of the updated LSAs of Agent_2 spreads in the network.

Step 5: Agent_3 is not syntactically sensitive to the "GasStation" property. However, it will use the newly added semantic reasoning module to bond with the "Parking" property as it is one of the gas station components (*isComposedOf* in Figure 5). Agent_3 internally identifies an electric parking spot and updates its LSA by adding the "E-Park" property value which allows the booking of the returned electric park spot (Figure 6-arrow between LSA of Agent_2 and Agent_3, and update of Agent_3 LSA).

Step 6: A copy of the updated LSAs of Agent_3 spreads in the network.

Step 7: Agent_1, waiting for a "ElectricParkSpot" property, will use the semantic module to check if similar properties returned by Agent_3 can be provided as a result to the user. "ElectricParkSpot" and "E-Park" have equivalent meaning (they are synonyms). Thus, Agent_1 bonds with the new property added by Agent_3 and retrieves the booking information about the electric park spot in the nearby gas station, and informs the user (Figure 6 arrow between LSA of Agent_1 and Agent_3, and update of Agent_1 LSA).

Step 8: A feedback is sent by the user to evaluate the returned composition. If it is a positive reward, agents that participated in the composition will all be rewarded positively, otherwise they will all be rewarded negatively. Agents update their learning tables according to the received reward. In this case, we assume that the system has returned the requested service and the user has given a positive reward. Instantly, the Qmatrix of all the agents that participated in the composition are updated as shown in Table 2 after a first query (rewarded positively) and in Table 3 after a second query (also rewarded positively).

To summarise, in this example we exploited the following relationships within the concepts defined by ontology shown in Figure 5: (1) the use of equivalent concepts (*equivalentTo* or *sameAs*); (2) the subsumption relationship (*is-a*); (3) the parts composing an element (*isComposedOf*).

Interestingly, Agent_1 S property "ElectricParkSpot" will also bond with Agent_2 P property "GasStation". By performing semantic reasoning using the concepts defined by the ontology given in Figure 5, we infer on the one hand that "ElectricParkSpot" is a "Park Spot", and on the other hand, that a "GasStation" is composed of a "Parking", which in turn is composed of "Park Spot". This possible new bonding will lead to a final schema, but not to the expected booking service, and so if proposed to the user, they would reward it negatively, that composition would be progressively discarded.

Table 2. Qmatrix of the agents after the first query (semantic version).

Agent	Schema	Ignore	React
Agent_2	ElectricParkSpot | Location,Parking	−3	6.5
Agent_3	ElectricParkSpot | Location,Parking,E-park	−3	6.5

Table 3. Qmatrix of the agents after the second query (semantic version).

Agent	Schema	Ignore	React
Agent_2	ElectricParkSpot | Location,Parking	−5.1	7.5
Agent_3	ElectricParkSpot | Location,Parking,E-park	−5.1	7.5

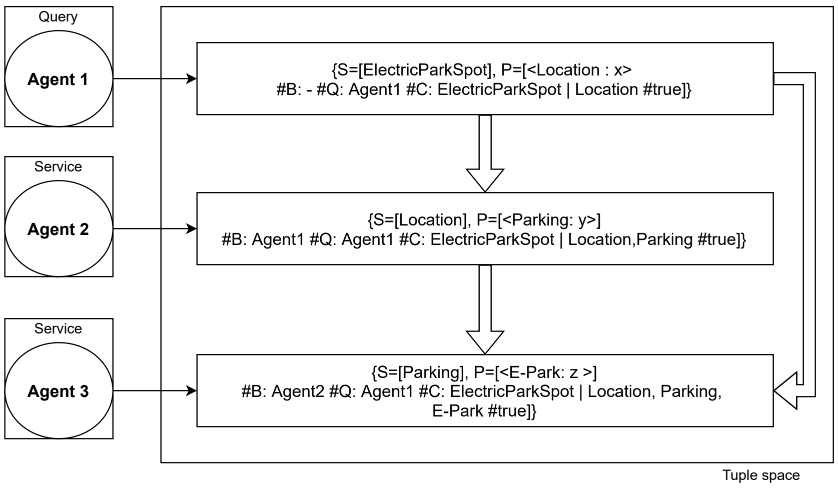

Query
Agent 1
Service
Agent 2
Service
Agent 3
{S=[ElectricParkSpot], P=[<Location : x>
#B: - #Q: Agent1 #C: ElectricParkSpot | Location #true]}
{S=[Location], P=[<Parking: y>]
#B: Agent1 #Q: Agent1 #C: ElectricParkSpot | Location,Parking #true]}
{S=[Parking], P=[<E-Park: z >]
#B: Agent2 #Q: Agent1 #C: ElectricParkSpot | Location, Parking, E-Park #true]}
Tuple space

Figure 6. Semantic composition diagram of the electric car parking scenario.

5. Implementation and Experiments

This section provides architectural details and implementation information, including details of the semantic module. It discusses alternative deployments and lists the various experiments we implemented and deployed so far, including the electric car parking scenario discussed above. We also present experiments related to parameters settings and discuss further validation issues.

5.1. Architecture

We propose an architecture composed of the following layers:

- **Presentation layer:** we propose to use an Angular web application to control the system. It allows to configure services and inject queries.
- **Service layer:** a Spring Boot application used to create a RESTful web service to instantiate and communicate with the "SAPERE Kernel" project. It is named "SAPERE api". It is a Maven project that provides a RESTful web service. It is considered as a mediator between the "SAPERE Kernel" and the web application. It uses a local MongoDB database to save user credentials and SAPERE configuration.
- **Application layer:** A Java-based project called "SAPERE Kernel" that provides the essential features of our proposed solution. It provides a bidirectional socket communication between all nodes in the network. The core of the SAPERE coordination model has been extended in order to support semantic-based composition, as shown in Figure 7. We defined a new bonding eco-law called *Semantic Bonding* (see Figure 7) that holds semantic reasoning capabilities thanks to the Apache Jena rules engine [48]. The former provides a general-purpose rule-based reasoner that is exploited to perform reasoning on RDF data. Both the reasoner and the RDF data are included in the semantic bonding module.

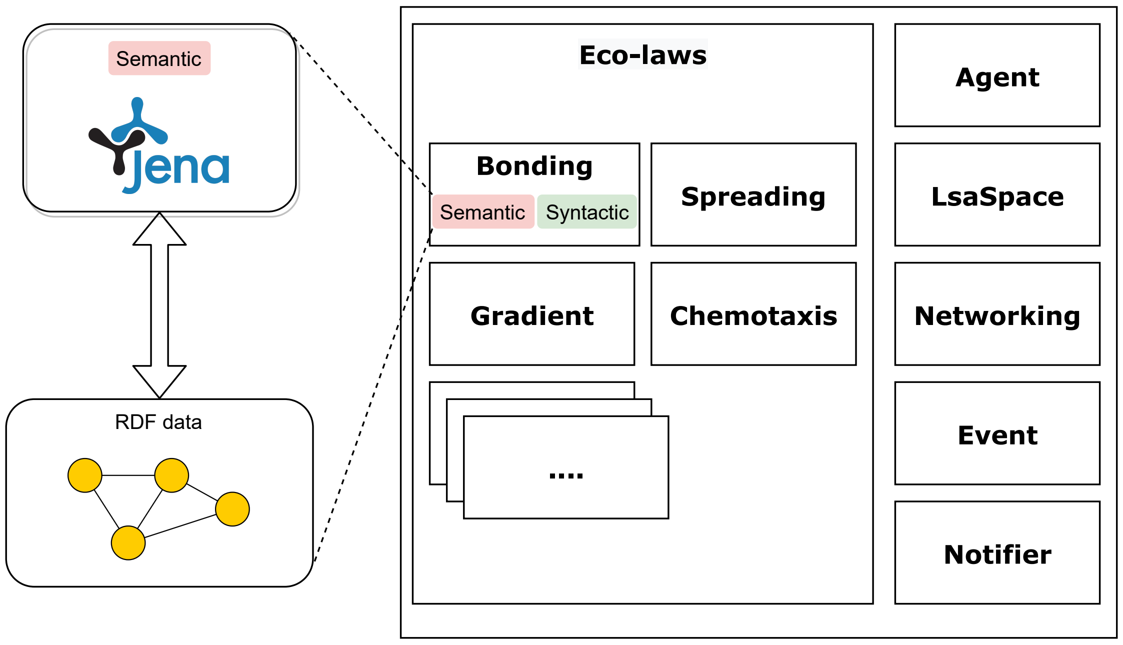


Figure 7. Learning-based semantic coordination middleware extended with Jena rule-based reasoner and RDF data to support semantic reasoning.

5.2. Implementation

We used the Spring Boot 2.3.2 framework to create micro-services and provide a RESTful API that allows to easily interact with the coordination model kernel. Besides, we used Angular 9 to build a web application and offer a graphical user interface.

We adapted and implemented the new features designed to compose services on-the-fly via semantic matching using Java 8. In particular, we developed the coordination model extension that enables the semantic matching composition. In support of it, we designed a serialisation/deserialisation module that allows representing the service parameters under the form of URIs, thus making them exploitable by the Apache Jena rules engine.

Such URIs are composed of *(i)* a base URI: the leftmost part of the URI, shared by multiple entities; and *(ii)* a URI fragment: the rightmost part that defines the entities/properties name. We defined a base URI (i.e., "http://cui.unige.ch/isi/onto/MDPI-JSAN#") that is used during the service matchmaking phase. During the composition process, each service parameter is appended to the base URI, creating a URI for each parameter. Figure 8 shows an example of URI generation on the parameters passed by Agent_1, Agent_2, and Agent_3 of the electric car parking scenario presented in Section 4.1. Such URIs are then used by the Apache Jena rules engine to evaluate the semantic relationship among the defined parameters.

It is important to highlight that the Jena engine is performing the reasoning on the entities defined by the ontology stored in the RDF data store, which are represented using URIs. For the electric car parking scenario, this corresponds to Figure 5. The URIs generated by appending the service parameters are only used in place of the ones defined in the ontology. Due to this reason, it is not possible to evaluate the semantic relationship among two parameters if they are not both defined in the stored ontology.

Finally, each component is wrapped in a Docker image. Moreover, a Docker Compose has been created to instantly launch the application. The project is publicly available at https://bitbucket.org/houssembenmahfoudh/sapere.

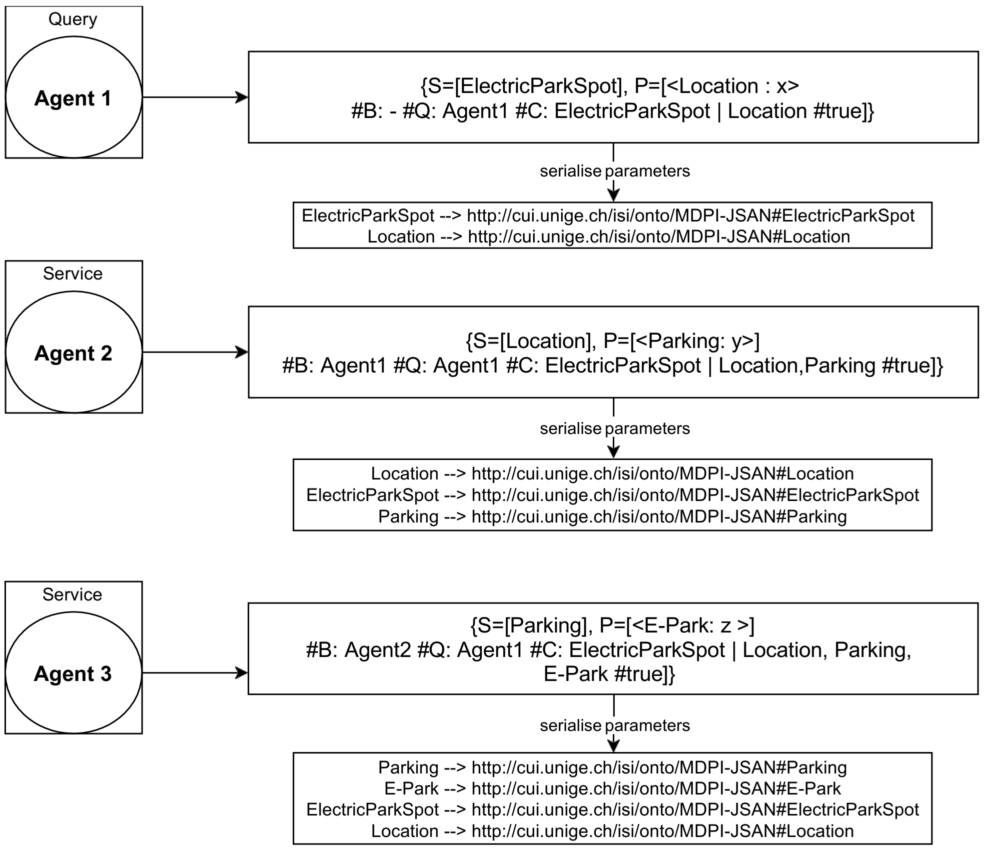

Query
Agent 1
{S=[ElectricParkSpot], P=[<Location : x>
#B: - #Q: Agent1 #C: ElectricParkSpot | Location #true]}
serialise parameters
ElectricParkSpot --> http://cui.unige.ch/isi/onto/MDPI-JSAN#ElectricParkSpot
Location --> http://cui.unige.ch/isi/onto/MDPI-JSAN#Location
Service
Agent 2
{S=[Location], P=[<Parking: y>]
#B: Agent1 #Q: Agent1 #C: ElectricParkSpot | Location,Parking #true]}
serialise parameters
Location --> http://cui.unige.ch/isi/onto/MDPI-JSAN#Location
ElectricParkSpot --> http://cui.unige.ch/isi/onto/MDPI-JSAN#ElectricParkSpot
Parking --> http://cui.unige.ch/isi/onto/MDPI-JSAN#Parking
Service
Agent 3
{S=[Parking], P=[<E-Park: z >]
#B: Agent2 #Q: Agent1 #C: ElectricParkSpot | Location, Parking,
E-Park #true]}
serialise parameters
Parking --> http://cui.unige.ch/isi/onto/MDPI-JSAN#Parking
E-Park --> http://cui.unige.ch/isi/onto/MDPI-JSAN#E-Park
ElectricParkSpot --> http://cui.unige.ch/isi/onto/MDPI-JSAN#ElectricParkSpot
Location --> http://cui.unige.ch/isi/onto/MDPI-JSAN#Location

Figure 8. URIs generation of the services/queries parameters used in the semantic matching process by the Apache Jena rules engine-electric car parking scenario.

5.3. Deployment

We consider essentially three variants for the deployment of systems supporting spatial services, a decentralised peer-to-peer deployment, centralised within a single node or a cloud-based deployment. We could also consider hybrid cases where some computation occurs at the edge and some through the cloud. For all these deployments, the semantic module (both Jena reasoning engine and RDF data) is fully included in each node. We do not yet consider cases, where knowledge is shared or updated among nodes (see a discussion on this point in Section 6).

We discuss here three different deployment cases, exemplified through the electric car parking scenario.

Figure 9 shows the case where the three entities run in three distinct computational nodes (mobile embedded devices, Raspberry of stationary nodes) connected in a P2P fashion. Each node hosts the learning-based semantic middleware.

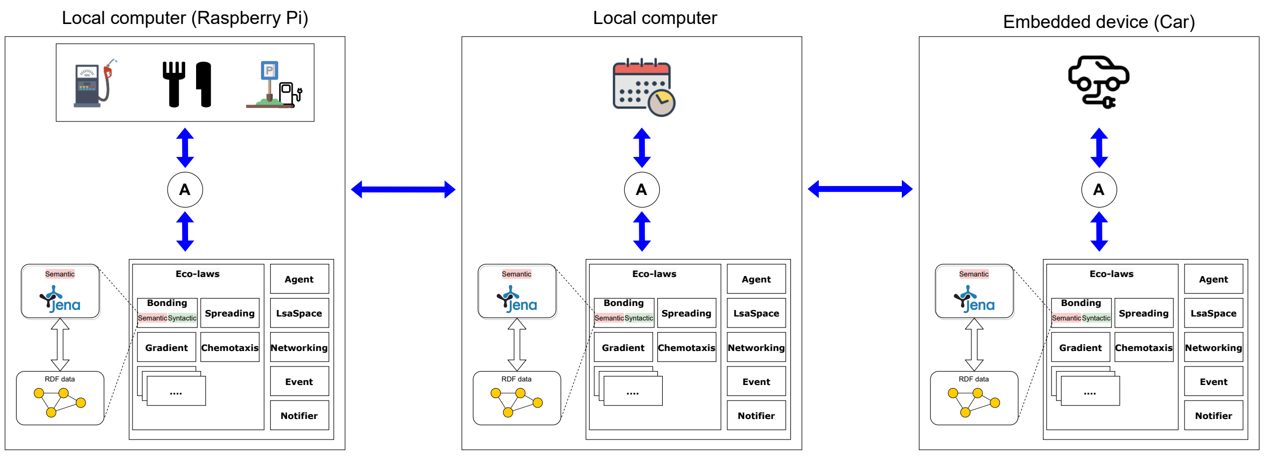


Figure 9. Deployment in a P2P network.

Figure 10 shows the three entities running inside the same node, which hosts one instance of the middleware, shared by the agents working on behalf of the three entities.

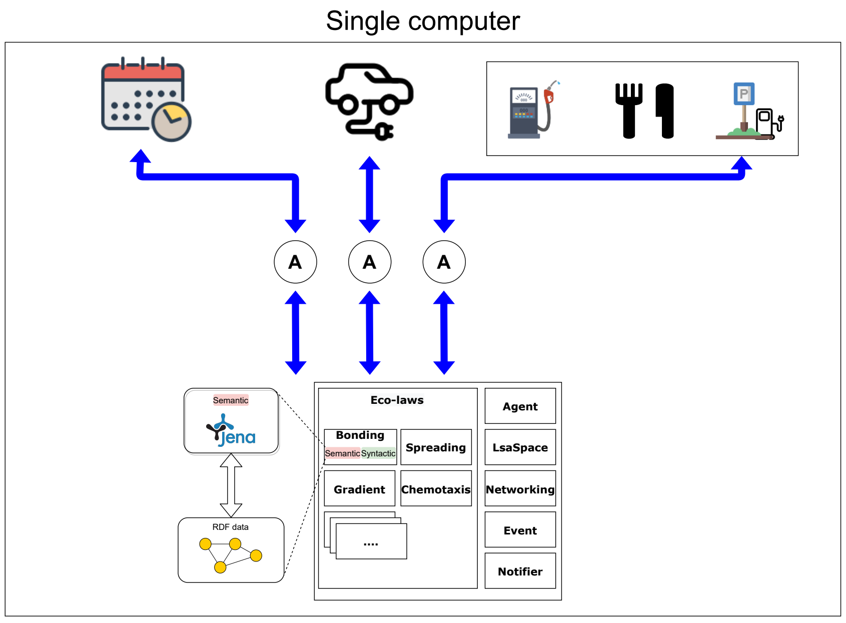


Figure 10. Deployment on a single computer.

Figure 11 shows a deployment through the cloud, where the three entities run on their own (possibly on separate computers) and connect through the Internet to a cloud, which hosts a single instance of the middleware shared by all entities.

In all cases, the agents (A) working on behalf of the entities run in the same node as the middleware. More details about implementation, deployment and adaptation could be found on previous papers [2,4].

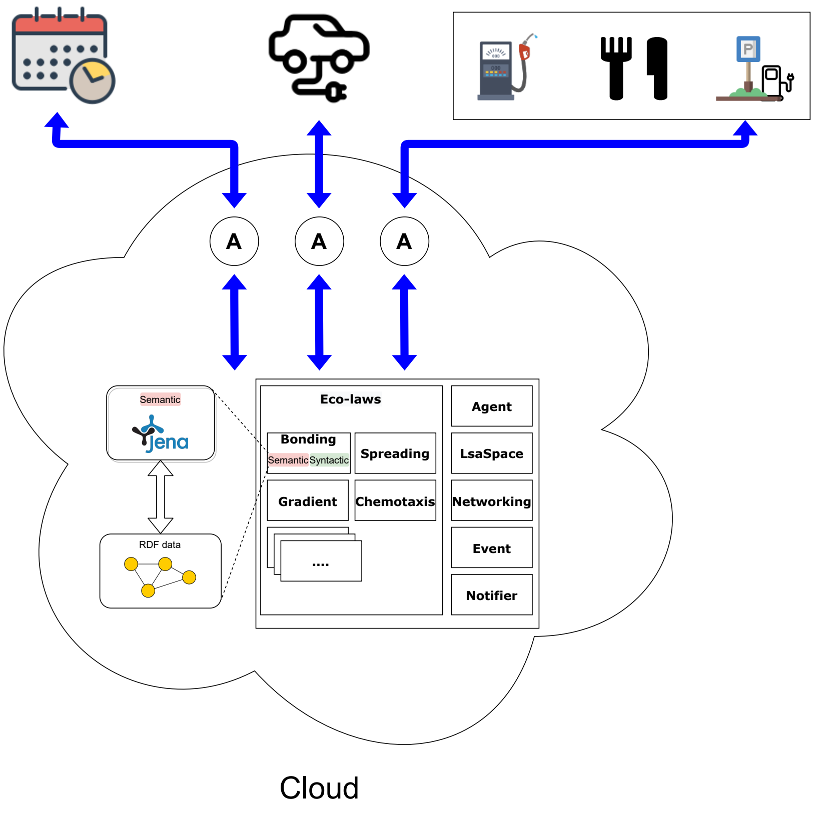


Figure 11. Deployment in the cloud.

5.4. Experiments

Several experiments support our development. They are different in their purpose, some validating the syntactic version of the middleware, other the semantic one, and in the type of services provided to the users (information, actions in the physical world).

5.4.1. Synthetic Services

In order to provide a variety of cases on which to test syntactic composition and learning, we worked on synthetic services developed using the coordination model implementation described in the previous section. We created a program that randomly generates such services, with random input and output names (taken in the alphabetic characters). For visualisation purposes, the program also automatically draws the services graph for all possible compositions based on the services' inputs and outputs names. All agents and services are attached to the same node. We automatically generate random services up to 100 services, in order to observe performance and correctness of compositions and update of learning tables (Qmatrix). A detailed description of synthetic services can be found in [2].

5.4.2. Electric Car Parking Scenario

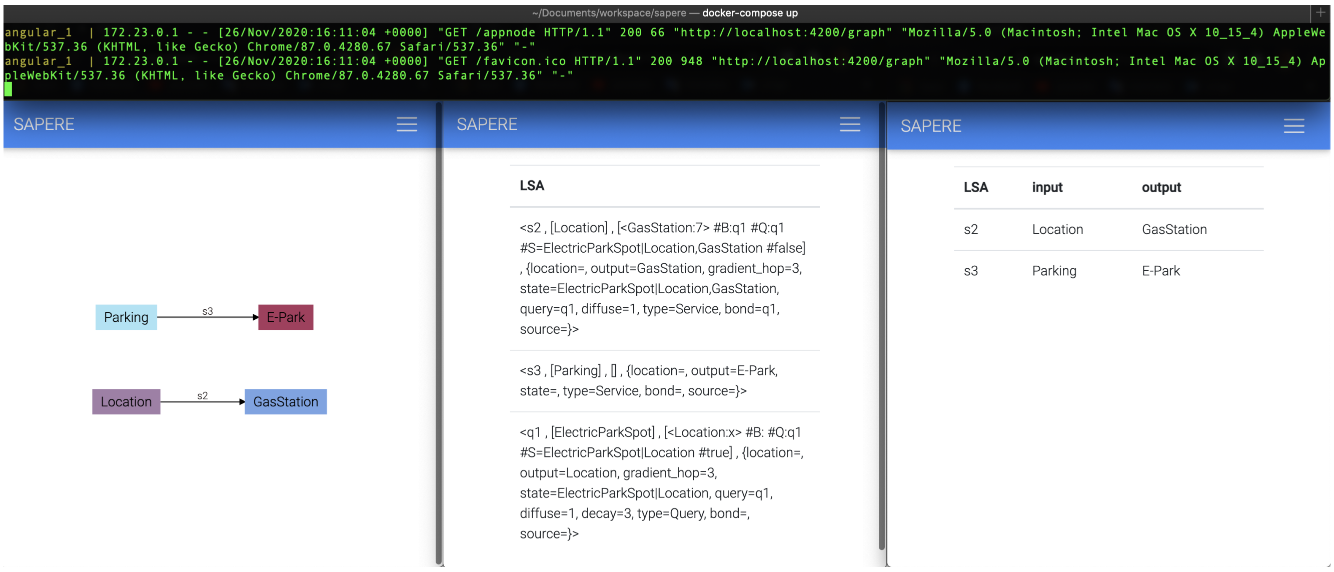
We deployed the electric car parking scenario, discussed in Sections 3.6 and 4.1, in a peer-to-peer fashion (see Figure 9), using a wireless ad hoc network. For experimentation purposes, we used three smart nodes equipped with our learning-based semantic middleware-the syntactic version for original scenario (Section 3.6) and the semantic version for the revisited scenario (Section 4.1). We attached to these nodes the services and the corresponding agents presented for the scenario, namely the Electric Car, the Gas Station and the Booking Service (see also Figure 3). We deployed as follows:

- A Raspberry Pi 3 hosting the gas station services (Agent_2).
- A Raspberry Pi 3 hosting the booking service (Agent_3).
- A computer which is used to inject the service request and evaluate results, representing the Electric Car (Agent_1).

We used the interface developed in the synthetic services experiments above to define, add, remove and connect services to the system, inject queries and visualise the various services, their compositions and the computed results.

We created the three agents described above and linked them to the middleware running on their respective node (computer or Raspberry Pi).

Figure 12 shows the result of this deployment through our interface. On the left, we see an automatically built graph representation of the services that advertised themselves. In this case, Agent_3 advertised service s3 (Booking of E-Park spots in a Parking), and Agent_2 advertised service s2 (GasStation is at Location). In the middle of the figure, we see the LSAs corresponding to Agent_2 (s2), Agent_3 (s3) and the query q1 of the Electric car. We can notice that the LSA of Agent_2 (s2) consumes the Location property provided by the Query q1 and injects the "GasStation" property. However, there is no services that can consume this property as it doesn't belong to any of the existing input services. Hence, no further composition can occur and the user won't get any result. On the right, we see a flat list of all services that advertised themselves (in this case here s2 and s3). We can see in practice, how services composition unfolds, and in this particular example, how the above scenario cannot return any result to the user as their input and output do not syntactically match.



```
angular_1  | 172.23.0.1 - - [26/Nov/2020:16:11:04 +0000] "GET /appnode HTTP/1.1" 200 66 "http://localhost:4200/graph" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_4) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/87.0.4280.67 Safari/537.36" "-"
angular_1  | 172.23.0.1 - - [26/Nov/2020:16:11:04 +0000] "GET /favicon.ico HTTP/1.1" 200 948 "http://localhost:4200/graph" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_4) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/87.0.4280.67 Safari/537.36" "-"
```

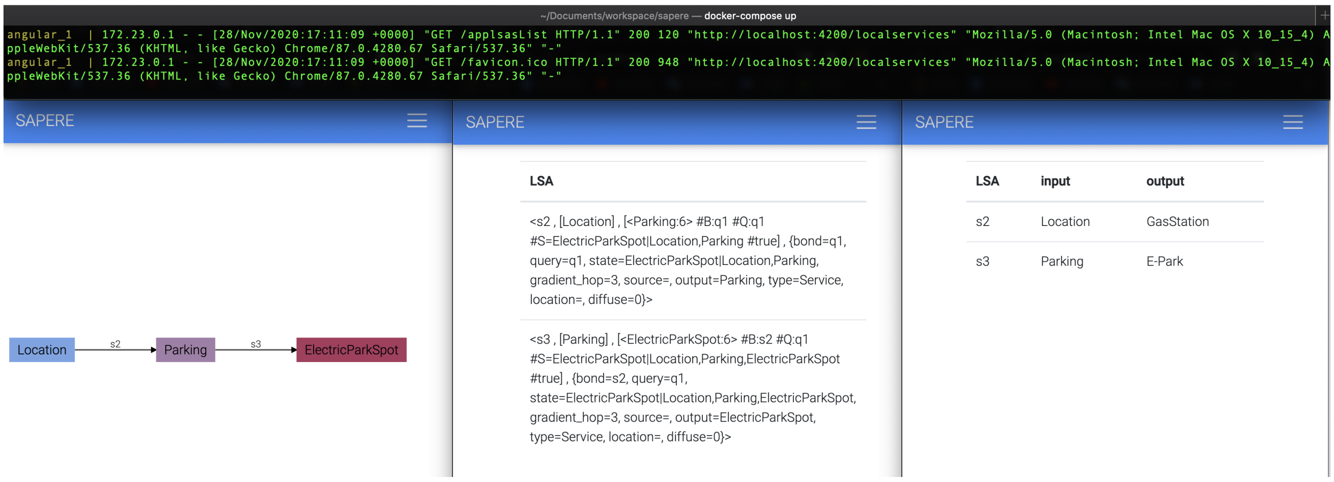
```
LSA

<s2 , [Location] , [<GasStation:7> #B:q1 #Q:q1 #S=ElectricParkSpot|Location,GasStation #false] , {location=, output=GasStation, gradient_hop=3, state=ElectricParkSpot|Location,GasStation, query=q1, diffuse=1, type=Service, bond=q1, source=}>

<s3 , [Parking] , [] , {location=, output=E-Park, state=, type=Service, bond=, source=}>

<q1 , [ElectricParkSpot] , [<Location:x> #B: #Q:q1 #S=ElectricParkSpot|Location #true] , {location=, output=Location, gradient_hop=3, state=ElectricParkSpot|Location, query=q1, diffuse=1, decay=3, type=Query, bond=, source=}>
```

LSA	input	output
s2	Location	GasStation
s3	Parking	E-Park

Figure 12. The experiment of the electric car parking scenario using the **syntactic** matching.

With the same interface and system, we executed the same scenario with the semantic version of our middleware. Figure 13 shows the new results. As before, on the right side of the figure, we find the two services s2 and s3 with their input and output. On the left side of the figure, this time we find an automatically built graph representation of a possible composition that provides the result expected by the user. In the middle, we see the LSAs. While in the syntactic version, it was impossible to associate a GasStation with a Parking, in the semantic version this becomes now possible.

Using the semantic bonding, Agent_2 will inject the GasStation property after bonding with the query agent. Using the semantic module, Agent_3 will check if there is any semantic matching between the GasStation and the parking property. In our case, the matching event will be triggered and Agent_3 will inject the new property E-Park. Similarly, Agent_1 will check if there is any semantic matching between the E-Park property and the ElectricParkSpot. Hence, E-Park being equivalent to the ElectricParkSpot (as requested in the query), a full service composition providing an ElectricParkSpot at the requesting Location is now found.



```
angular_1  | 172.23.0.1 - - [28/Nov/2020:17:11:09 +0000] "GET /applsasList HTTP/1.1" 200 120 "http://localhost:4200/localservices" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_4) A
ppleWebKit/537.36 (KHTML, like Gecko) Chrome/87.0.4280.67 Safari/537.36" "-"
angular_1  | 172.23.0.1 - - [28/Nov/2020:17:11:09 +0000] "GET /favicon.ico HTTP/1.1" 200 948 "http://localhost:4200/localservices" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_4) A
ppleWebKit/537.36 (KHTML, like Gecko) Chrome/87.0.4280.67 Safari/537.36" "-"
```

LSA
<s2 , [Location] , [<Parking:6> #B:q1 #Q:q1 #S=ElectricParkSpot|Location,Parking #true] , {bond=q1, query=q1, state=ElectricParkSpot|Location,Parking, gradient_hop=3, source=, output=Parking, type=Service, location=, diffuse=0}>
<s3 , [Parking] , [<ElectricParkSpot:6> #B:s2 #Q:q1 #S=ElectricParkSpot|Location,Parking,ElectricParkSpot #true] , {bond=s2, query=q1, state=ElectricParkSpot|Location,Parking,ElectricParkSpot, gradient_hop=3, source=, output=ElectricParkSpot, type=Service, location=, diffuse=0}>

LSA	input	output
s2	Location	GasStation
s3	Parking	E-Park

Figure 13. The experiment of the electric car parking scenario using the **semantic** matching.

5.4.3. Emergency Situation-ICRC

We detailed an emergency situation where a doctor asks for blood bags to be brought to their location [2]. In this case study, five services are available: two ICRC tents providing blood bags, and three means of transport-a drone, a car, and a helicopter. The doctor's query starts a series of interactions among the agents working on behalf of the five services and progressively six potential spatial services emerge (each tent with each transportation means). We deployed this scenario with the syntactic version of the learning-based coordination middleware, both in a single node, and in a P2P network, and showed the correctness of the compositions, and the progressive convergence towards the services preferred by the doctor (e.g., the ones involving the drones and the car, but not the helicopter).

5.4.4. Earlier Experiments

In earlier experiments, we investigated various IoT scenarios [3], with connected objects providing actual actions in the physical environment in response to users requests.

Smart lighting In this experiment, the Spatial service provided to the user consists of providing an illuminated path made of progressively lighting a series of bulbs. In this example, the service provides some coordination actions of the bulbs in the environment [3]. We attached five connected bulbs, as shown in Figure 14, each one to a dedicated Raspberry Pi running our middleware. We deployed the five Raspberry Pis in a peer-to-peer manner in a sequential line. The user injected a query to get an illuminated path. This information propagated (from the user to the end of the line, under the form of a gradient) among the Raspberries, causing an interaction among the agents controlling the bulbs. Once the information reached the end of the line, the final agent then lights its bulb and informs back the other agents. Upon receiving that information from the previous agent in the line, the current agent also lights its bulb, and so on until the illuminated path is complete and reaches the user. A demonstration video (https://www.youtube.com/watch?v=nSOJHKM95lg) shows the feasibility of the approach.

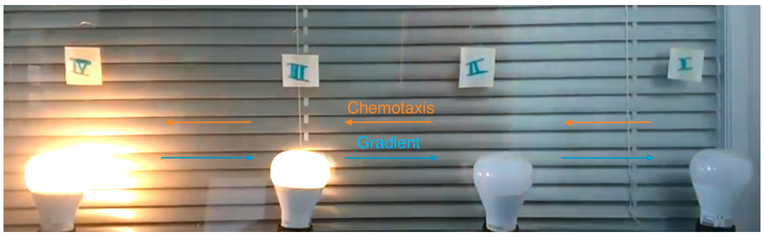


Figure 14. Smart lighting scenario.

Smart energy sharing We also experimented a more complex scenario [49], with the same bulbs (representing household) and Raspberry Pis. We provide a service requesting and negotiating energy sharing, as shown in Figure 15 (reproduced by the bulbs becoming lighter or darker depending on whether the household they represent receive or need energy). A demonstration video (https://www.youtube.com/watch?v=xvI_cK-qF6Y) shows the feasibility of the approach. Figure 15 shows the scenario that we built using a set of connected objects.

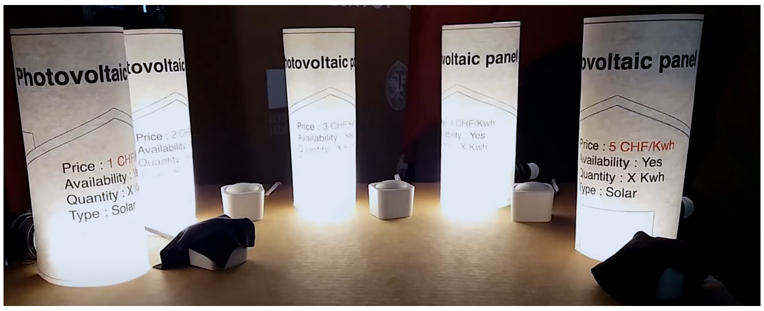


Figure 15. Service composition for energy management in a distributed system.

Tracing and visualising runners in a race In other experiments, with similar settings (P2P arrangement of Raspberry Pis and mobile phones running the middleware), we were able to trace and visualise the progression of runners, wearing beacons, during a race. The deployment occurred in our University campus and involved active participants [3], as shown in Figure 16.

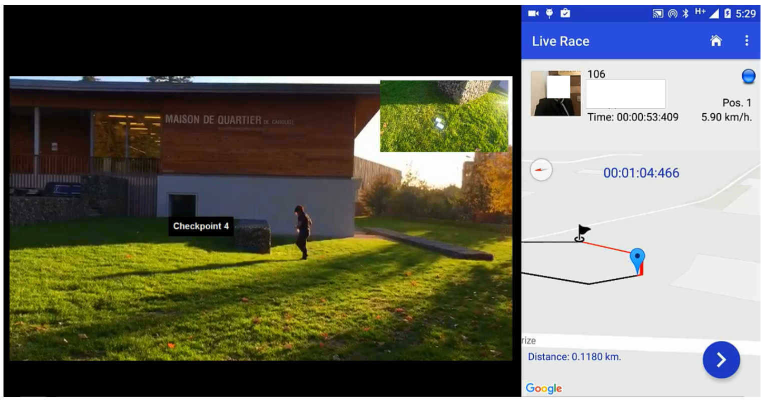


Figure 16. Tracing and visualising runners.

5.4.5. Parameters

We also carried out experiments aiming at identifying appropriate values for the various learning parameters and to validate the adaptability of the system to dynamic changes (such as arrival or departure of agents) [4,50]. Experiments concerned 15 agents involved in up to 800 learning cycles (queries followed by dynamic compositions and positive or negative rewards).

- **Alpha** The learning rate α is set between 0 and 1. Figure 17 presents how values in the Q matrix varies according to α. The x-axis, that we named iteration, presents a user feedback under the form of a reward regarding a query. When the learning rate is close to 1, our system learns faster than when it is set close to 0 (where a higher number of feedback are required). Similarly, an agent changes quickly its behaviour after few opposite feedback. Since in our system, agents are not systematically rewarded, and we are dealing with a dynamic environment, α has to be relatively small in order to limit the sensitivity of the agents to received feedback, accommodating by the way resistance to false negative and positive feedback.
- **Epsilon** We have employed the ϵ-greedy reinforcement learning algorithm [47] that has a probability ϵ to select a random action and a probability $1 - \epsilon$ to select the action that maximises the value of the approximation of $Q(s, a)$. ϵ-greedy ensures a permanent exploration which is necessary to allow adaptation to changing environmental conditions. A high ϵ value downgrades learning since the agent will explore more frequently the system, by making more frequent random choices. A small ϵ value lowers the adaptation capability of the system when the environmental conditions change. Therefore, choosing a suitable value of ϵ is critical. For example, when $\epsilon = 0.2$, the agents explore the system in 20% of the cases.
- **Gamma** is a discount factor that weights the future rewards. It shows the importance of such rewards in the learning process. When γ is close to 0, agents are short-sighted as γ gives importance to the cumulative discounted future reward. However, When it is close to 1, it helps agents to have further observations.
- **Dynamicity: adding/removing agents** Results in relation with dynamic changes of agents show that when agents disappear, the compositions in which they were involved are no longer available, negative feedbacks are sent to the the remaining agents, who quickly invert their Qmatrix and learn to avoid those compositions. In the case of new agents arriving in the system, and consequently new possible compositions, the system needs a few iterations cycles in order for the whole set of agents to learn the new meaningful compositions.

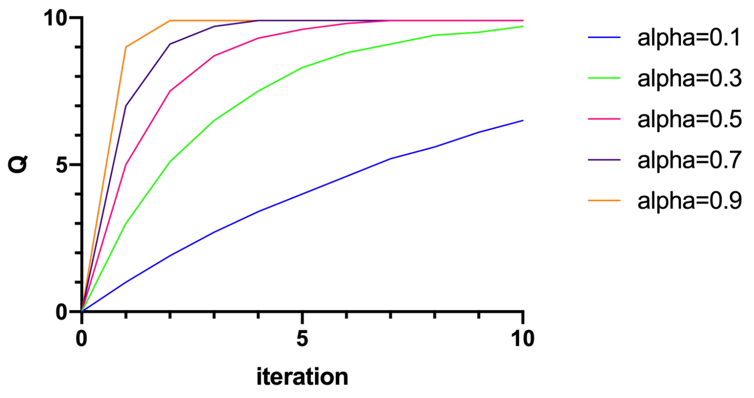


Figure 17. Learning rate-alpha.

For further optimisation, these parameters could be adapted according to the use case or could be dynamically adapted during the run-time.

5.5. Further Validation

We validated our system by comparing some metrics like response time, oversizing, QoS. An analysis and discussion covering strengths, weaknesses and limitations is detailed in the following Section 6.

For further validation, we intend to focus on deploying and testing our system on large-scale examples, as well as conducting thorough studies about the system parameters like spreading frequency and any possibility to adapt it at run-time. We will also investigate detailed comparison of different scenarios with and without semantic matching, and comparison with other learning approaches. We are in the process of providing additional physical deployment with robots and connected objects to target spatial services having an actual effect on the physical environment.

6. Discussion

In this section, we will have a general discussion about the strengths, weaknesses, and limitations of our approach.

Initial exploration and convergence of learning: Firstly, our system needs several queries and feedback to adjust the agents' behaviour. During this phase, solutions corresponding to possible compositions are produced, caused by the exploration process and QoS. However, these compositions may not be meaningful or useful to the user. Thus, the system then provides non-pertinent services to the user. It requires several users' feedback in order to converge towards pertinent services.

We propose to use a reinforcement learning module in each agent to learn the best action to take regarding the partial schema that it receives during the composition. Our system progressively learns the appropriate compositions based on the users' requests, providing the pertinent composition (the actual service expected by the user). In addition, the system progressively avoids unnecessary bonding operations and compositions as the agents learn the optimal policy. As long as the users are providing feedback regarding the returned results, the agents can adapt to changes and adjust their behaviour.

We have been using the QLearning algorithm throughout this work. A comparison with other reinforcement learning algorithms, such as SARSA or Deep QLearning, could improve the effectiveness of our work.

Personalised learning: As previously mentioned, our system requires several users' feedback in order to learn how to deal with each query. The proposed platform provides on-the-fly service composition at run-time, and correctly addresses the answer of a query (information or action) to the user that made that query. However, the platform does not provide personalised learning. Users may request different services even though they use the same query.

Indeed, we propose an emergent solution based on users' evaluation and agents do not actually discriminate between users' queries. Two users, performimg the exact same query, may not reward the same result in the same manner. If we need to go further, a three-dimensional Qmatrix or a user classification could be a solution to provide personalised learning.

Adaptability: Our platform provides a collective adaptive solution as we are dealing with a dynamic environment. The system is able to automatically detect at run-time the absence or the addition of new agents. The detection of such events is supported thanks to the indirect retrieval and injection of property in the shared tuple space. It mainly depends on the learning rate α and the exploration factor of the greedy algorithm ϵ parameters. When α is close to 1, our system learns faster than when it is set close to 0. On the other hand, ϵ is used in the epsilon-greedy algorithm. This algorithm has a probability ϵ to select random action and a probability $1 - \epsilon$ to select the action that maximises the value of the approximation of $Q(s, a)$. When a change occurs, the agents are not able to instantly adapt to it. The adaptation process requires more feedback compared to the ones needed for the initial exploration. For further optimisation, these parameters could be adapted according to the use case or could be dynamically adapted at the run-time.

Results: We decided to select and return the most pertinent and reliable solution according to the learning modules. However, this strategy may raise some design faults as a user has to make multiple queries before being provided with a pertinent result. An alternative may be to return all possible compositions to the user and offer the possibility to evaluate results. This approach is not user friendly either as it may return a long meaningless list of results.

Spreading: Our strategy to coordinate between all available nodes in the network is to spread a copy of the LSA every time a new property is added in the property list. This strategy is not efficient as it can flood the network with useless data. This strategy could be optimised by learning the spreading policy. The LSA's copy has to be sent to the nodes that host services that it can bond with. This can avoid broadcasting and limit the traffic between nodes.

Composition length: Service composition is a sequence of parallel and sequential composition among all available services. If we deal we numerous services, the length of the composition schema may be too long. However, the longer we go, the more resources we use. Limiting the length of the composition schema and dynamically adapting it to the available resources could be an optimisation to consider in the future.

Multiple queries: Our platform supports multiple queries. Thanks to the #*Q* field that we have added for each property, we can discriminate between properties in the property list. At the same time, many queries can be executed. Each query may be injected in any node in the network. If a composition can return the requested property, the result will be returned to the query agent.

Exceptions: An agent learns the right action to take according to the partial schema that it gets when it bonds with another LSA. However, it takes time to change its behaviour when a change occurs. This depends on the parameters used in the learning module. Furthermore, when agents do not succeed to provide a result, the user can imagine that there is no possible response to his query even if it exists. Automatically iterating the query more than once, until we get a result, could be a better solution for enhancing the user experience.

Internal loop: All agents have a list of properties related to all different compositions and queries at a given time. We have implemented a FIFO method to keep the latest relevant data in the list. Thus, if the number of actual processing properties exceeds the size of the property list, a bonding loop can occur. For example, if we fix the property list's size to five and bond with more than five services at the same time, a bonding loop will arise. We suggest to adapt the property list's size to the use case need or apply the decay for properties.

Composition Loops: During composition, a loop can arise among services. To avoid loops during the execution, each agent checks that its LSA does not bond with the same LSA more than once for the same query. When a new property is injected in the property list, it keeps the #B and #Q attribute. Thus, when it bonds again with the same LSA, it verifies that the #B and #Q of the new property are different from the properties already in the property list. If it is the case, the property will be rejected.

Evaluation: When a composition arises and obeys to the query that has been injected by the user, he/she is asked to evaluate the result. The latter contains the requested property and its value, the last agent that returns the result, the query name and the composition schema. The user can assess the result based on the returned value and the composition schema. However, the composition schema is not significantly efficient to evaluate the result.

Performance: When the network gets bigger and the services become numerous, many compositions may arise. This may confuse the user as it has to choose among many results. Our coordination model has an internal clock. At every cycle, all the eco-laws and operations are executed. Thus, every node loops over its LSAs hosted in the tuple space and matches them together to fire a bonding event. This event will be triggered by the agent to execute the controlled service. Thus, the clock rate is the main factor of the service composition's speed.

Semantic: Adding semantic reasoning capabilities to the coordination model allows to create compositions that could not be detected, thus either generated, using a syntax-based approach. The adoption of a semantic-based approach enlarges the number of possible matchings and the number of service compositions that could be generated. Thus, the resulting composition graph will become more and more complex and may lead to misleading feedback provided by the users. Additionally, despite the improvement we bring with the ontology, the requests and services advertisements are still prone to interpretation.

Sharing knowledge: Semantic reasoning relies on the existence of a shared knowledge that formally represents the properties advertised by the services and queried by the users. In our electric car parking scenario, we assumed that such knowledge fully describes the domain and it is replicated in each node, making it available to each agent. Removing the completeness assumption, each agent might have a piece of knowledge that differ from other agents'. With such configuration each agent might communicate with others, requesting the needed piece of knowledge that it does not have. Another viable configuration might be having a centralised knowledge that is accessible by each agent. The former would allow decoupling the agents and the knowledge, enabling the modification/replacement of the knowledge while keeping the agents unaware of it.

7. Conclusions

Our research picks up on Weiser's vision of services made available to users, on-demand, in a seamless manner and arising from the interaction among various devices embedded in the environment. To this end, we specifically provide a learning-based semantic middleware deployable locally, in a peer-to-peer fashion, on a cloud, or on a hybrid manner across edge and cloud. We discussed a series of experiments, from a specific case study to synthetic services, and parameters' setting.

On-demand service composition is an appealing approach for an enormous list of applications. It relies on connected objects, deals with dynamic environments and does not need any complex or fixed infrastructure. It could have an impact on several domains and applications like social distancing, energy sharing, smart city, emergency situation, disaster crisis, traffic steering, etc.

Future works include: (1) deployment of scenarios involving concrete actions in the physical environment through swarm robotics, as well as large-scale scenarios; (2) study of the parameters linked to spreading frequency or with learning and how to adapt those parameters at run-time; (3) comparison with other learning approaches; (4) complementing our approach to facilitate users' interaction with the system by adding a Natural Language Understanding (NLU) module.

Author Contributions: Conceptualisation, H.B.M., A.C. and G.D.M.S.; Software, H.B.M., A.C.; Validation, H.B.M.; Writing—original draft preparation, H.B.M., A.C. and G.D.M.S.; Writing—review and editing, H.B.M., A.C. and G.D.M.S.; Visualisation, H.B.M.; Supervision, G.D.M.S.; Project administration, G.D.M.S.; Funding acquisition, G.D.M.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The data presented in this study are available at: https://bitbucket.org/houssembenmahfoudh/sapere.

Acknowledgments: The authors wish to thank Roberto Tomaylla for the implementation of the tracing and visualisation of runners in a race as part of his MSc thesis project at the University of Geneva.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Weiser, M. The Computer for the 21st Century. *Sci. Am.* **1991**, *265*, 94–104. [CrossRef]
2. Ben Mahfoudh, H. Learning-Based Coordination Model for Spontaneous Self-Composition of Reliable Services in a Distributed System. Ph.D. Thesis, University of Geneva, Geneva, Switzerland, 2020.
3. Di Marzo Serugendo, G.; Abdennadher, N.; Houssem, B.M.; De Angelis, F.L.; Tomaylla, R. Spatial Edge Services. In Proceedings of the 2017 Global Internet of Things Summit (GIoTS), Geneva, Switzerland, 6–9 June 2017.
4. Ben Mahfoudh, H.; Di Marzo Serugendo, G.; Naja, N.; Abdennhader, N. Learning-based coordination model for spontaneous self-composition of reliable services in a distributed system. *Int. J. Softw. Tools Technol. Transf.* **2020**. [CrossRef]

5. Kalasapur, S.; Kumar, M.; Shirazi, B.A. Dynamic Service Composition in Pervasive Computing. *IEEE Trans. Parallel Distrib. Syst.* **2007**, *18*, 907–918. [CrossRef]
6. Lemos, A.L.; Daniel, F.; Benatallah, B. Web Service Composition: A Survey of Techniques and Tools. *ACM Comput. Surv.* **2015**, *48*, 1–41. [CrossRef]
7. Peltz, C. Web services orchestration and choreography. *IEEE Comput.* **2003**, *36*, 46–52. [CrossRef]
8. Talantikite, H.N.; Aissani, D.; Boudjlida, N. Semantic annotations for web services discovery and composition. *Comput. Stand. Interfaces* **2009**, *31*, 1108–1117. [CrossRef]
9. Martin, D.; Burstein, M.; Hobbs, J.; Lassila, O.; Mcdermott, D.; Mcilraith, S.; Narayanan, S.; Paolucci, M.; Parsia, B.; Payne, T.; et al. OWL-S: Semantic Markup for Web Services. Available online: https://www.w3.org/Submission/2004/SUBM-OWL-S-20041122/ (accessed on 30 September 2020).
10. Talib, M.A.; Yang, Z. Semi-Automatic Code Generation of Static Web Services Composition. In Proceedings of the Student Conference on Engineering, Sciences and Technology, Karachi, Pakistan, 30–31 December 2004; pp. 132–137. [CrossRef]
11. Tzortzis, G.; Spyrou, E. A Semi-Automatic Approach for Semantic IoT Service Composition. In Proceedings of the Workshop on Artificial Intelligence and Internet of Things (AI-IoT), at the 9th Hellenic Conference on Artificial Intelligence (SETN 2016), Thessaloniki, Greece, 18–20 May 2016.
12. Pierris, G.; Kothris, D.; Spyrou, E.; Spyropoulos, C. SYNAISTHISI: An enabling platform for the current internet of things ecosystem. In Proceedings of the 19th Panhellenic Conference on Informatics, PCI 2015, Athens, Greece, 1–3 October 2015; Karanikolas, N.N., Akoumianakis, D., Nikolaidou, M., Vergados, D.D., Xenos, M., Giaglis, G.M., Gritzalis, S., Merakos, L.F., Tsanakas, P., Sgouropoulou, C., Eds.; ACM: New York, NY, USA, 2015; pp. 438–444. [CrossRef]
13. Noura, M.; Gaedke, M. WoTDL: Web of Things Description Language for Automatic Composition. In *IEEE/WIC/ACM International Conference on Web Intelligence*; Association for Computing Machinery: New York, NY, USA, 2019; pp. 413–417. [CrossRef]
14. Editor, W. Thing Description (TD) Ontology. 2020. Available online: https://www.w3.org/2019/wot/td (accessed on 29 November 2020).
15. Gyrard, A.; Datta, S.K.; Bonnet, C.; Boudaoud, K. A Semantic Engine for Internet of Things: Cloud, Mobile Devices and Gateways. In Proceedings of the 2015 9th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing, Santa Catarina, Brazil, 8–10 July 2015; pp. 336–341. [CrossRef]
16. Zgheib, R. SeMoM: A Semantic Middleware for IoT Healthcare Applications. Ph.D. Thesis, Université Toulouse 3 Paul Sabatier, Toulouse, France, 2017.
17. Zgheib, R.; Conchon, E.; Bastide, R. Semantic Middleware Architectures for IoT Healthcare Applications. In *Enhanced Living Environments: Algorithms, Architectures, Platforms, and Systems*; Springer International Publishing: Cham, Switzerland, 2019; pp. 263–294. [CrossRef]
18. Rao, J.; Su, X. A survey of automated web service composition methods. In Proceedings of the First International Conference on Semantic Web Services and Web Process Composition, San Diego, CA, USA, 6 July 2004; Springer: Berlin/Heidelberg, Germany, 2005; pp. 43–54. [CrossRef]
19. Grondin, G.; Bouraqadi, N.; Vercouter, L. MaDcAr: An Abstract Model for Dynamic and Automatic (Re-)Assembling of Component-Based Applications. In *Component-Based Software Engineering*; Lecture Notes in Computer Science; Gorton, I., Heineman, G.T., Crnkovic, I., Schmidt, H.W., Stafford, J.A., Szyperski, C.A., Wallnau, K.C., Eds.; Springer: Berlin/Heidelberg, Germany, 2006; Volume 4063, pp. 360–367. [CrossRef]
20. Hellenschmidt, M. Distributed Implementation of a Self-Organizing Decentralized Multimedia Appliance Middleware. In *Dagstuhl Seminar Proceedings, Mobile Computing and Ambient Intelligence: The Challenge of Multimedia, 1–4 May 2005*; Davies, N., Kirste, T., Schumann, H., Eds.; IBFI: Schloss Dagstuhl, Germany, 2005.
21. Vallée, M.; Ramparany, F.; Vercouter, L. A Multi-Agent System for Dynamic Service Composition in Ambient Intelligence Environments. In *PERVASIVE 2005, Advances in Pervasive Computing*; Austrian Computer Society (OCG): Vienna, Austria, 2005; Volume 191, pp. 175–182.
22. Gabillon, Y.; Calvary, G.; Fiorino, H. Composing interactive systems by planning. In Proceedings of the 4th French-Speaking Conference on Mobility and Ubiquity Computing (UbiMob '08), Saint Malo, France, 28–30 May 2008; pp. 37–40. [CrossRef]
23. Bonjean, N.; Gleizes, M.P.; Maurel, C.; Migeon, F. SCoRe: A Self-Organizing Multi-Agent System for Decision Making in Dynamic Software Developement Processes. In Proceedings of the International Conference on Agents and Artificial Intelligence (ICAART), Barcelona, Spain, 15–18 February 2013.
24. Degas, A. Auto-Structuration de Trafic Temps-réel Multi-objectif et Multi-Critère dans un Monde Virtuel. Ph.D. Thesis, Université de Toulouse III—Paul Sabatier, Toulouse, France, 2020.
25. Viroli, M. On Competitive Self-composition in Pervasive Services. *Sci. Comput. Program.* **2013**, *78*, 556–568. j.scico.2012.10.002. [CrossRef]
26. Frei, R.; Şerbănuţă, T.F.; Di Marzo Serugendo, G. Self-organising assembly systems formally specified in Maude. *J. Ambient. Intell. Humaniz. Comput.* **2012**, *5*, 491–510. [CrossRef]
27. Di Napoli, C.; Giordano, M.; Németh, Z.; Tonellotto, N. Using chemical reactions to model service composition. In *Proceedings of the 2nd International Workshop on Self-Organizing Architectures (SOAR'10)*; ACM: New York, NY, USA, 2010; pp. 43–50. [CrossRef]
28. De Angelis, F.L. A Logic-Based Coordination Middleware for Self-Organising Systems: Distributed Reasoning Based on Many-Valued Logics. Ph.D. Thesis, University of Geneva, Geneva, Switzerland, 2017.

29. Mazac, S.; Armetta, F.; Hassas, S. Bootstrapping sensori-motor patterns for a constructivist learning system in continuous environments. In Proceedings of the 14th International Conference on the Synthesis and Simulation of Living Systems (Alife'14), New York, NY, USA, 31 July 2014.
30. Boes, J.; Nigon, J.; Verstaevel, N.; Gleizes, M.P.; Frederic, M. The Self-Adaptive Context Learning Pattern: Overview and Proposal. In Proceedings of the International and Interdisciplinary Conference on Modeling and Using Context (CONTEXT 2015), Larnaca, Cyprus, 2–6 November 2015; Springer: Berlin/Heidelberg, Germany, 2015; pp. 91–104.
31. Ren, L.; Wang, W.; Xu, H. A Reinforcement Learning Method for Constraint-Satisfied Services Composition. *IEEE Trans. Serv. Comput.* **2017**. [CrossRef]
32. Wang, H.; Chen, X.; Wu, Q.; Yu, Q.; Zheng, Z.; Bouguettaya, A. Integrating On-policy Reinforcement Learning with Multi-agent Techniques for Adaptive Service Composition. In Proceedings of the International Conference on Service-Oriented Computing, Paris, France, 3–6 November 2014.
33. Wang, H.; Wang, X.; Hu, X.; Zhang, X.; Gu, M. A multi-agent reinforcement learning approach to dynamic service composition. *Inf. Sci.* **2016**, *363*, 96–119. [CrossRef]
34. Wang, H.; Gu, M.; Yu, Q.; Tao, Y.; Li, J.; Fei, H.; Hong, T. Adaptive and large-scale service composition based on deep reinforcement learning. *Knowl. Based Syst.* **2019**, *180*, 75–90. [CrossRef]
35. Ahmed, M.; Takayuki, I. A Deep Reinforcement Learning Approach for Large-Scale Service Composition. In Proceedings of the International Conference on Principles and Practice of Multi-Agent Systems, Tokyo, Japan, 29 October–2 November 2018.
36. Idir, A.; Saber, B.; Laid, K.; Kazar, O. Service composition approaches for Internet of Things: A review. *Int. J. Commun. Netw. Distrib. Syst.* **2019**, *23*, 194–230.
37. Zambonelli, F.; Castelli, G.; Ferrari, L.; Mamei, M.; Rosi, A.; Di Marzo, G.; Wally, B. Self-aware Pervasive Service Ecosystems. *Procedia Comput. Sci.* **2011**, *7*, 197–199. [CrossRef]
38. Martin, W.; Matthias, M.H.; Nora, K.; Philip, M. (Eds.) *Software Engineering for Collective Autonomic Systems—The ASCENS Approach*; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2015; Volume 8998. [CrossRef]
39. Mamei, M.; Zambonelli, F. Programming Pervasive and Mobile Computing Applications: The TOTA Approach. *ACM Trans. Softw. Eng. Methodol.* **2009**, *18*, 15:1–15:56. [CrossRef]
40. De Angelis, F.L.; Fernandez-Marquez, J.L.; Di Marzo Serugendo, G. Self-composition of services in pervasive systems: A chemical-inspired approach, Multi-Agent Systems: Technologies and Applications. In *Advances in Intelligent Systems and Computing*; Springer: Berlin/Heidelberg, Germany, 2014; Volume 296.
41. De Angelis, F.L.; Fernandez-Marquez, J.L.; Di Marzo Serugendo, G. Self-composition of services with chemical reactions. In Proceedings of the 29th Annual ACM Symposium on Applied Computing (SAC), Gyeongju, Korea, 24–28 March 2014.
42. Ben Mahfoudh, H.; Di Marzo Serugendo, G.; Boulmier, A.; Abdennadher, N. Coordination Model with Reinforcement Learning for Ensuring Reliable On-Demand Services in Collective Adaptive Systems. In *International Symposium on Leveraging Applications of Formal Methods*; Springer: Cham, Switzerland, 2018.
43. Fernandez-Marquez, J.L.; Di Marzo Serugendo, G.; Montagna, S.; Viroli, M.; Arcos, J.L. Description and composition of bio-inspired design patterns: A complete overview. *Nat. Comput.* **2013**, *12*, 43–67. [CrossRef]
44. Manju, S.; Punithavalli, M. An Analysis of Q-Learning Algorithms with Strategies of Reward Function. *Int. J. Comput. Sci. Eng.* **2011**, *3*, 814–820.
45. Richard, S.S.; Andrew, G.B. *Reinforcement Learning: An Introduction*; MIT Press: Cambridge, MA, USA, 1998.
46. Shauharda, K.; Kagan, T. Evolution-Guided Policy Gradient in Reinforcement Learning. In Proceedings of the 32nd Conference on Neural Information Processing Systems, Siem Reap, Cambodia, 13–16 December 2018.
47. Kaelbling, L.P.; Littman, M.L.; Moore, A.W. Reinforcement learning: A survey. *J. Artif. Intell. Res.* **1996**, *4*, 237–285. [CrossRef]
48. Apache Jena-Reasoners and Rule Engines: Jena Inference Support. Available online: https://jena.apache.org/documentation/inference/ (accessed on 30 September 2020).
49. Ben Mahfoudh, H.; Di Marzo Serugendo, G.; Abdennadher, N.; Rumsch, A.; Upegui, A. Spatial services for decentralised smart green energy management. In Proceedings of the 2018 IEEE International Energy Conference (ENERGYCON), Limassol, Cyprus, 3–7 June 2018.
50. Naja, N. Utilisation de L'apprentissage par Renforcement pour la Composition de Service dans un Milieumdistribuée. Master's Thesis, University of Geneva, Geneva, Switzerland, 2019.