Quality-Based Reinforcement Learning in Intelligent Opportunistic Software Composition

Kahina Hacid, Sylvie Trouilhet, Jean-Paul Arcangeli, Françoise Adreit

Quality-Based Reinforcement Learning in Intelligent Opportunistic Software Composition

K. Hacid
Univ. of Toulouse, UPS, IRIT
Toulouse, France
Kahina.Hacid@irit.fr

S. Trouilhet
Univ. of Toulouse, UPS, IRIT
Toulouse, France
Sylvie.Trouilhet@irit.fr

F. Adreit
Univ. of Toulouse, UT2J, IRIT
Toulouse, France
Francoise.Adreit@irit.fr

J.-P. Arcangeli
Univ. of Toulouse, UPS, IRIT
Toulouse, France
Jean-Paul.Arcangeli@irit.fr

***Abstract*—Internet of Things and cyber-physical systems are characterised by openness and an increasing number of devices and their associated services. In a previous work, we have proposed to exploit opportunistically these services in order to automatically make emerge customised applications that suit user preferences. For that, we have developed a generic solution for bottom-up opportunistic service composition, based on reinforcement learning. In this work, it is extended to handle more efficiently the appearance of new components using *service annotation* and *quality attributes* in order to generalise and share knowledge with new discovered services. A didactic use case is used for illustration and demonstration purposes.**

***Index Terms*—Ambient Intelligence, Smart Assistance, Reinforcement Learning, Quality, Knowledge Sharing, Agents, Emergence, Software Composition, Internet of Things**

I. Introduction

Today's users are living in cyber-physical pervasive environments that are more and more complex with the increasing number of devices of the Internet of Things. These environments consist of fix or mobile devices driven by software components using communication networks to operate. Due to user and device mobility, software components may appear with unpredictable dynamics, giving to pervasive environments an open nature. In such a context, applications based on component assemblies are hard to design, maintain and adapt.

In order to tackle these issues, our project aims to design and build an "Opportunistic Composition Engine" (OCE) that opportunistically assembles software components [1] in order to build applications that are both adapted to the current state of the environment and to the user. In this opportunistic approach, applications are automatically built on the fly in a bottom-up manner from the components that are available at that time, without explicit user requirements or predefined assembly plans. In this way, applications emerge from the environment, taking advantage of opportunities as they arise. However, the user is "in the loop" and keeps control on the environment: she/he finally decides on the relevance of the emergent application before it is deployed, and OCE learns from this user feedback to build future applications [2].

This work is part of the AILP (Assistance InteLligente et proactive en environnement Professionnel) project, which is supported by the French region Occitanie and the operational program FEDER-FSE Midi-Pyrénées et Garonne.

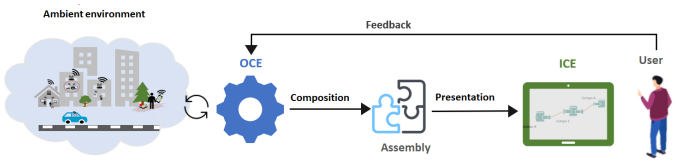

Fig. 1. Overall architecture of the composition system

A weakness of the current solution is OCE random decision in the presence of new components, i.e., components about which OCE has no knowledge: if such a component is a candidate to participate in an assembly, it may be selected by OCE with an arbitrarily set probability called novelty sensitivity coefficient [2]. This paper proposes an improvement to handle the appearance of new components more efficiently by considering component properties and exploiting knowledge about "similar" components: we enrich the OCE learning mechanism to generalise and transfer knowledge gained from past experiences with some components to components that have not yet been encountered.

Our issue can be assimilated to the cold start problem encountered in recommendation systems [3] and to knowledge transfer in learning [4]. Unlike recommendation systems, the cold start problem confronted here is a recurrent problem, faced each time a new component is discovered. This is of particular importance in the case of OCE which is used in dynamic and open environments.

This paper is structured as follows. Section II describes the current global solution of the learning-based opportunistic composition system and its limits. The extension of this solution to enhance OCE decisions on new components is presented in Section III. A didactic case study is discussed throughout these sections in order to demonstrate the approach. Finally, Section IV overviews different related approaches. A conclusion ends this paper and identifies future research directions.

II. Learning-Based Opportunistic Composition

This section presents the principles of the learning-based opportunistic software composition system.

A. Architecture of the composition system

Figure 1 shows the overall architecture of the composition system. At the center, the Opportunistic Composition Engine

(OCE) has the main task of assembling the components. It periodically senses the ambient environment in order to discover the available components with the services they provide and those they require to be operational, connects component required services to provided services [1] and thereby constructs on the fly composite applications.

As those applications have not been demanded by the user, the latter has to be informed of them before they are actually deployed. For that, she/he is integrated in the architectural loop through a graphical user interface called "Interactive Control Environment" (ICE) that provides the user a multi-view description of the application for control [5].

Using ICE, the user can either accept, modify or reject the application. OCE deduces feedback data from these user's actions. These data are the only source of feedback: to avoid burdening the user, the engine does not explicitly ask questions about the user experience. This feedback is central as it allows OCE to learn about the user in the current environment in an endless online reinforcement learning process [6].

OCE is designed in a multi-agent system (MAS) architectural style [7] which is known to meet main challenges raised by ambient environments: decentralisation, distribution, scalability, dynamics and adaptiveness. Any provided or required service of a component is managed by a dedicated agent. Agents interact in order to locally decide on a correct and pertinent connection to realise. They use a four-step communication protocol: Advertise, Reply, Select and Agree [2]. An agent's decision is based on its local view of the ambient environment and on estimated values about neighbouring agents that have been computed by reinforcement from previous user feedback. Thus, learning and decision are distributed through the MAS and each agent decides for its own connections. In the absence of information, decisions are random.

B. Reference situations and learning

When a component is detected, each service of this component is associated with an agent. In the case of a service not yet encountered, the agent has no knowledge; otherwise it has a set of *reference situations* which constitutes its own knowledge base. A reference situation Ref_i^k is a situation numbered k that an agent A_i has encountered in the past. It gathers the agents A_j that A_i has encountered in the situation k whose service is compatible with the one of A_i. More precisely, Ref_i^k is a set of pairs $(A_j, Score_{ij})$, where $Score_{ij}$ is a numerical value that represents the interest for A_i to connect its service with the one of A_j in this situation. More A_i participates to application assemblies, more it learns and more reference situations it owns. The decision strategy of A_i relies on these reference situations and their similarity with the current situation S_C: a reference situation Ref_i^k is *similar* to S_C if the intersection between the sets of agents in Ref_i^k and S_C contains at least ξ elements, ξ being a parameter of the strategy. Thus, an agent A_i that is candidate to the current OCE assembly chooses the agent to respond to (the connection to pursue) among all the agents that responded to its connection request in the current situation, by applying a Best Score Selection strategy: the best scored agent in the similar reference situations is considered as the best one to establish a connection in the current situation and therefore is chosen by A_i.

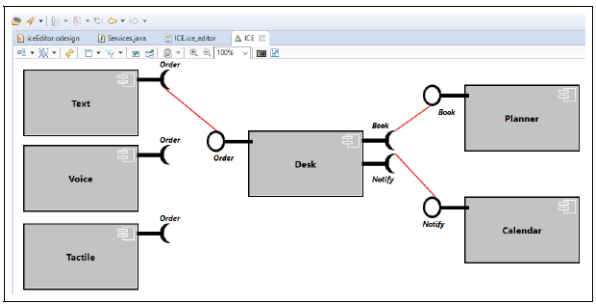

Fig. 2. *Presentation of the emergent application (ICE interface)*

Then, the emergent assembly is presented to the user. Last, after the latter has accepted, modified or rejected the presented assembly, each service agent adjusts and updates its knowledge base (i.e., its reference situations) according to the user actions and the feedback she/he provides.

C. Mary didactic use case

The solution has been applied to a didactic case study presented in [2]: Mary, who works in a manufacturing company has the opportunity to book a room for a meeting. To highlight the limits of the current solution, we take up and extend this scenario: Mary's company is located on two sites, *Toulouse* (France) and *Hamburg* (Germany). Mary is assigned to *Toulouse* but she regularly has appointments in *Hamburg*.

Let us replay the beginning of the use case. In the *Toulouse* ambient environment, there are components supplied by the company: a room Planner providing the **Book** service, a booking Desk providing the **Order** service and requiring both the **Book** and **Notify** services, and a Tactile input device that requires **Order** service. There also are Mary's personal components: a Text input and a Voice input interfaces both requiring **Order** service, and Mary's Calendar that provides **Notify** service (note that all these components are named according to their function, i.e., **Voice** references *a* voice-like component provided by Mary). In such a situation, OCE makes emerge an application that allows Mary to book a room for a business meeting. Figure 2 presents the emergent application. We focus on the agent B that manages the provided **Order** service of Desk. We show how it first reacts regarding the agents A_1, A_2, and A_3 that respectively manage the **Order** services required by **Text**, **Voice** and **Tactile**. OCE is started and runs until an assembly emerges. Since B has no knowledge when starting OCE, A_1, A_2 and A_3 being the candidates to connect (they constitute the current situation of B), they all receive an identical score of 1/3. Then OCE randomly chooses to connect B with agent A_1 (which manages the service required by Text component). As Mary prefers the **Voice**'s service (managed by A_2), she modifies the proposed

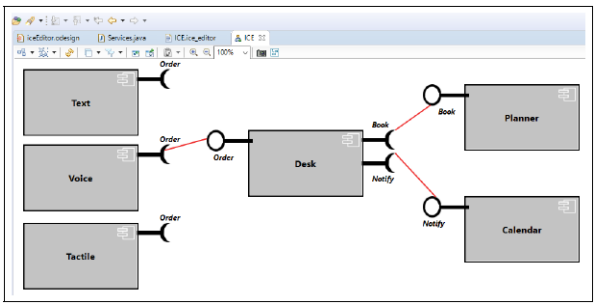

Fig. 3. *Emergent application modified by Mary (ICE interface)*

assembly using ICE functionalities. The final application is presented in Figure 3

OCE learns from this modification: from the randomly scored current situation, $\{(A_1{:}0.33); (A_2{:}0.33); (A_3{:}0.33)\}$, it builds, by reinforcement and normalisation, the reference situation Ref_B^k=$\{(A_1{:}0); (A_2{:}0.6); (A_3{:}0.4)\}$ (calculation method can be found in [2]) and stores it in its knowledge base. As OCE has now learned Mary's preference in the current situation, it will automatically propose a booking application integrating the agent A_2 in situations like this one.

For the rest of the use case, we follow what happens when Mary faces a comparable situation in *Hamburg*.

D. Learning-based OCE Limits

OCE inability to decide for a new component other than randomly reveals a limitation in the current approach. Indeed, if OCE succeeds in learning and saving user preference for a specific service, it does not generalise this information and does not apply the resulting actions to new discovered and *functionally equivalent* services (i.e., service that performs a same task). It can be said that OCE has learned but not enough. As a result, in case of changing environments, OCE has to interact more often with the user and repeatedly in order to make emerge a *functionally equivalent* assembly. This issue could be assimilated to a cold start problem associated with a knowledge transfer problem. But, in our case, the cold start problem is encountered each time a new service is discovered, which can be extremely frequent in dynamic environments.

Let us consider the use case extension. OCE has managed to learn Mary's preferences between agents A_1, A_2 and A_3. Assuming that Mary goes back to the same room again, OCE would automatically build the relevant application from now on. But **what happens if Mary is now in *Hamburg*?** Assuming that Mary's phone runs out of battery, its provided service (**Order** service of Voice component), managed by A_2, is no more available. By chance, the *Hamburg* site proposes all the components present in *Toulouse* (Planner, Desk, Tactile), including a Company calendar providing a **Notify** service, a Text component and a Microphone, all requiring the **Order** service. Therefore, new agents A'_1, A'_2 and A'_3 are associated (by OCE) to the newly discovered **Order** services of Text, Microphone and Tactile. In these conditions, OCE manages these new services as any other and proposes a **random** assembly which is happen to associate the Desk component with Tactile rather than taking into account Mary's preference of a Voice-like component expressed in *Toulouse* site.

Then, Mary has to modify again the assembly to manually choose a Voice-like service - which is the Microphone's service - one more time as she already did in Toulouse for the service of Voice.

It would be worthwhile for OCE to generalise Mary's preference of a Voice-like service, i.e. to prefer any *functionally equivalent* Voice-like service once she has indicated her preference at the first time. This would avoid the user having to interact again for the same type of connection each time a new Voice-like service is discovered (for example, resulting from changes in the user's environment when moving from one site to another).

To overcome this limitation, we propose to associate *service annotation* and *quality attributes* with the services and use this information in the decision process for newly discovered services. This extension requires to make explicit the existing *functional equivalences* between known services and newly discovered ones.

III. Quality-based learning

The Best Score Selection strategy is refined to take into account *service annotation* and specific *quality attributes*. This raised questions such as:

1) how to explicit functional equivalences between services?
2) how to represent and manipulate quality attributes?
3) when to use annotations and quality attributes, for the cold start problem only or throughout the OCE live cycle?
4) what weight do they have in relation to the agents' score in reference situations?
5) what effect do they have on learning?

These issues are discussed in the following.

A. Annotations to explicit functional equivalence between services

Annotations are defined to functionally categorise the services and make explicit the existing *functional equivalences* between identified services. Each functional category of services has a unique identifier and each service is annotated with a functional category identifier; for example services of a Voice-like component are annotated with a "VO" tag. Thus, all *functionally equivalent* services share a same annotation. Figure 4 shows such services annotations.

Annotation definition as well as *functional equivalences* formalisation should be performed using consensual descriptive models like domain ontologies [8]. Note that the domain information formalisation and the *functional equivalence* mechanism are out of scope of this paper.

B. Quality attributes description and representation

In this approach, *quality attributes* are variables associated to a service *S* in order to describe its properties. These

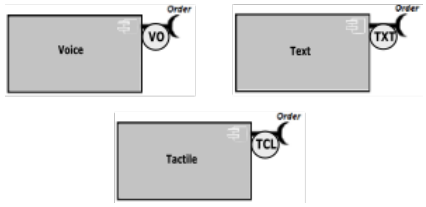

Fig. 4. An example of component annotation

```
<?xml version="1.0" encoding="UTF-8"?>

<component Name="Calculator"
           Role="Receive data">
    <service xsi:type="ProvidedService"...>
        <qualityattributes>
        <attribute name="ResponseTime"
         value="f()=10ms" type="absolute"/>
        </qualityattributes>
    </service>
</component>
```

Listing 1. *Quality attributes* XML description

attributes can represent for example a service performance (e.g. response time, availability) or situation (e.g. location, opening hours). They are defined by the service provider and associated to a set of possible values given within the service description.

A quality attribute is represented by three fields: **name**, **value** and **type**. **value** is a mathematical function which can be either a *constant function* (e.g., location value) or a *polynomial function* (e.g., distance for which the value depends on the user current location). **type** explicits the function type of **value**; it is *absolute* in case of a *constant function* or *relative* in case of a *polynomial function*. **type** is exploited by OCE to decide either to use a **value** directly (in case of an absolute value) or to run the given function. Listing 1 illustrates how *quality attributes* are represented within an XML description (used in [5] to describe components and services).

The mathematical function associated to **value** may be directly defined by the component provider or formalised and then referenced using consensual descriptive models like domain ontologies [8].

C. Integration of annotations and quality attributes in OCE

Annotations and *quality attributes* are analysed by OCE when discovering a new component and are then integrated into OCE learning and decision strategy in two steps as follows.

1) Service annotation: the first OCE enhancement consists in annotating the services available in the ambient environment. OCE should then exploit *functional equivalence* relationships to identify *functionally equivalent* services.

Service annotation offers a first OCE improvement regarding new discovered services management. It allows knowledge sharing and generalisation. OCE learning strategy is extended to take into account these annotations of services: when a new service *S* is discovered and categorised, the service agents of the same category share their knowledge (i.e., their reference situations) with the newly appearing service agent.

Priority	Category	Attribute	Value	Rate
0	VO	Bandwidth	18kbps 64Kbps	- 0.2 +0.6
1	VO	Maximum_Frequency	15KHZ 8KHZ	+ 0.13 -0.3
5	TXT	Keyboard_Language	Azerty Qwerty	+ 0.8 -0.4
...	...	...	...	...

TABLE I
A PREVIEW OF QUALITY ATTRIBUTE TABLE

2) Integration of quality attributes in OCE decision algorithm: the second enhancement of OCE consists in integrating the *quality attributes* within the decision algorithm. Once a new service is detected, its *quality attributes* are analysed by OCE. The quality attributes are referenced in a specific table. Table I gives a preview of such a table which is composed of five columns: Priority, Category, Attribute, Value and Rate.

- Priority column is used to indicate the existing priority between the *quality attributes* (0 is the highest priority). This is essential in case of a service having various *quality attributes*. It would indicate the importance of these attributes regarding the user preferences and which weight would OCE give to each of these attributes for its binding decisions (e.g. should it focus on the physical distance of an ambient service rather than its attendance?). For now, the priorities are randomly fixed but should be automatically set or adjusted by OCE, by learning, based on its analysis of user preferences.
- Category column identifies the category of the service to which the attribute belongs.
- Attribute column contains the *quality attributes* **name** (extracted from the XML service description).
- Value column repertories all the encountered **values** that have been associated to the specific attribute using the defined function in the XML description. The possible values can either directly populated by OCE (OCE can add a value line each time it encounters a new value) or be referenced using consensual descriptive models as ontologies.
- Rate column is used to associate a rate to each value of an attribute. These rates are deduced from user feedback: when a feedback is made by the user, OCE analyses it and decides service agents rewards in consequences [2]. In the same way, this reward is also given to values of the attributes associated to this service agent. This reward is positive if the service connection is approved within the assembly and negative if not.

These *quality attributes* and especially their rate values give a more explicit knowledge of user preferences regarding its services. Hence, *quality attribute* table is exploited by OCE to better handle the newly discovered services and make more

Fig. 5. Voice-like component annotation

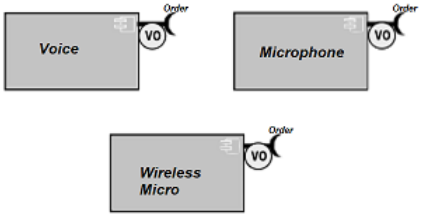

Fig. 6. A new Voice-like component annotation

accurate assembly propositions.

D. Mary didactic use case

We now reconsider the use case presented in subsection II-D and apply the OCE enhanced strategy. Using this new strategy, **what happens if Mary is now in Hamburg?** The new discovered service of Microphone (Order required service) is first identified and classified using a *functional equivalence* analysis.

As shown in Figure 5, the service of Microphone is annotated with the identifier *"VO"*. It is thus identified as *functionally equivalent* to the service of the Voice component.

Exploiting these annotations, OCE becomes able to accurately handle the new discovered service and to propose a solution that suits Mary's preference of Voice. OCE considers and handles the agent A_2' (that manages the service of Microphone) as the same agent as A_2. Thus, Mary's preference has been shared between all *functionally equivalent* services known by OCE without Mary's intervention.

Furthermore, it would be reasonable to consider that the *Hamburg* site proposes more than one Voice-like service as shown in Figure 6 (in which the agent A_2'' manages the service of Wireless Micro). Another question then arises: **what happens if there is more than one Voice-like service available?** That is, **how can OCE choose the Voice-like service that best meets Mary's preferences?**

To address this problem, OCE exploits *quality attributes*. The sensed Voice-like services are associated to a *quality attribute*: *Maximum_Frequency* which indicates the maximum frequency a Voice-like service can support. Suppose the values of *Maximum_Frequency* indicates 15KHZ for the A_2 service, 8KHZ for A_2' and 15KHZ for A_2''. Table II represents the corresponding *quality attributes* table.

Priority	Category	Attribute	Value	Rate
1	VO	Maximum_Frequency	15KHZ 8KHZ	+ 0.13 -0.3

TABLE II
QUALITY ATTRIBUTE TABLE: MAXIMUM_FREQUENCY ATTRIBUTE

Priority	Category	Attribute	Value	Rate
1	VO	Maximum_Frequency	15KHZ 8KHZ	+ 0.18 -0.63

TABLE III
USER FEEDBACK APPLICATION ON MAXIMUM_FREQUENCY ATTRIBUTE

Following Table II, OCE chooses the 15KHZ value (with the rate of 0.13) and then the A_2'' service which best meets Mary's preference of a greater frequency support on her voice command service. Mary is satisfied with the OCE proposition and accepts the assembly. This new feedback affects *quality attribute* rates as shown in table III.

E. Experimentation and evaluation

A proof-of-concept implementation of our solution has been developed and ongoing demonstrations of quality-based learning are being performed.

Our solution is going to be applied on a real case study borrowed from the AILP project[1] taking place in an airport with resources and services available to travellers. We already have developed real software components that are assembled by OCE in emergent applications that actually work [9]. The *Airport case study* offers a realistic environment full of all kind of components present in the ambient environment. We focus on the composition of guidance applications allowing the user to find the closest phone charging station. The airport environment offers several charging stations. In this context, based on the already collected user preferences, OCE has to build a guidance application to the best charging station.

IV. RELATED WORK

In this section, we review papers that are related to our work. In a first part, we consider service composition systems that use extra-functional data to decide on assemblies. We study in particular the data they use (as context data) and the methods they implement to do this (mainly deep learning and ontologies). In a second part, we consider systems that address the cold start problem.

In [10], authors propose a solution for service composition in complex and dynamic environments based on reinforcement learning, QoS prediction and neural networks. Q-Learning algorithm (a form of reinforcement learning algorithm) is used to deal with the complex and extremely dynamic environment and QoS changing. In [11], possible applications of deep reinforcement learning techniques for service composition in a Cloud manufacturing context are explored. Deep reinforcement learning provides an alternative approach for solving cloud manufacturing service composition. An AI planning-based composition framework for automated service composition, that integrates formal composition requirements and context awareness, is proposed in [12]. A *context service* is used to perceive and adapt to changes in the environment. The framework detects the changes of the environment and

[1]AILP is an acronym for InteLligent and proactive Assistance in a Professional environment

dynamically adjusts service execution by using BPEL (Business Process Execution Language) and agent technologies. In [13], authors propose a self-adaptive and context-aware service composition system. Contexts are formalised in dedicated context ontologies using OWL (Ontology Web Language). Using these ontologies, the system can handle evolving and changing contexts.

In [14], a distributed algorithm is proposed to dynamically optimise Web service composition in varying environments: within a multi-agent system, agents learn by reinforcement using a Q-learning algorithm and share their experience to improve efficiency and speed up the learning rate. A composition algorithm based on the clustering of services in relation to QoS is proposed in [15]. In [16], self-adaptive composition of Web services in dynamic environments maximises the global QoS of the composition: service composition is modelled as a Markovian decision process with several alternative processes, the best one being chosen using a Q-learning algorithm. In order to provide predictions for new users, recommendation systems use mechanisms based on similarity techniques. In particular, a heuristic similarity measure composed of three similarity factors, Proximity, Impact and Popularity, is used [3], [17], [18].

V. Conclusion

This paper presents a new and innovative approach for automated user-oriented service composition in ambient open and dynamic environments, based on online reinforcement learning from user feedback. This approach has been improved to efficiently deal with the appearance of new services. A new *quality based* solution has been developed and associated to the existing learning based opportunistic solution. The *quality based* solution exploits *service annotation* and *quality attributes* to help the Opportunistic Composition Engine (OCE) to take the best decision when it comes to select and integrate new unknown services in an application. Mary's didactic use case has been discussed throughout the paper for illustration and demonstration purposes. Demonstrations on larger and real case studies are in progress.

Some improvements of our *quality based* solution are envisaged and can be discussed. First, service annotation is done manually. but automated mechanisms might be implemented using unsupervised learning techniques to formally categorise the new discovered services regarding their functions. Second, for now the *quality attributes* priorities are decided arbitrary and manually affected. A solution allowing OCE to perform a profound and refined analysis on user feedback in order to deduce user preferences regarding *quality attributes* priorities and automatically affecting them is under study.

References

[1] I. Sommerville. Component-based software engineering. In *Software Engineering*, pages 464–489. Pearson Education, 10^{th} edition, 2016.

[2] W. Younes, S. Trouilhet, F. Adreit, and J.-P. Arcangeli. Agent-mediated application emergence through reinforcement learning from user feedback. In *29th IEEE Int. Conf. on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*. IEEE Press, 2020. https://hal.archives-ouvertes.fr/hal-02895011.

[3] A. I. Schein, A. Popescul, L. H. Ungar, and D. M. Pennock. Methods and metrics for cold-start recommendations. In *Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '02, page 253–260, New York, NY, USA, 2002. Association for Computing Machinery.

[4] S. J. Pan and Q. Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, 2010.

[5] S. Trouilhet, J.-P. Arcangeli, Bruel J.-M., and M. Koussaifi. Model-Driven Engineering for End-Users in the Loop in Smart Ambient Systems. *Journal of Universal Computer Science*, 27(7), July 2021.

[6] R. Sutton and A. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2nd edition, 2018.

[7] M. Wooldridge. *An Introduction to MultiAgent Systems*. Wiley Publishing, 2nd edition, 2009.

[8] K. Hacid. *Handling domain knowledge in system design models. An ontology based approach.* PhD thesis, Institut National Polytechnique de Toulouse, France, 2018.

[9] K. Delcourt, F. Adreit, J.-P. Arcangeli, K. Hacid, S. Trouilhet, and W. Younes. Automatic and Intelligent Composition of Pervasive Application (Demo). In *19th IEEE Int. Conf. on Pervasive Computing and Communications (PerCom)*. IEEE CS Press, 2021.

[10] H. Wang, J. Li, Q. Yu, T. Hong, J. Yan, and W. Zhao. Integrating recurrent neural networks and reinforcement learning for dynamic service composition. *Future Generation Computer Systems*, 107:551–563, 2020.

[11] H. Liang, X. Wen, Y. Liu, H. Zhang, L. Zhang, and L. Wang. Logistics-involved qos-aware service composition in cloud manufacturing with deep reinforcement learning. *Robotics and Computer-Integrated Manufacturing*, 67:101991, 2021.

[12] Z. Cao, X. Zhang, W. Zhang, X. Xie, J. Shi, and H. Xu. A Context-Aware Adaptive Web Service Composition Framework. In *IEEE Int. Conf. on Computational Intelligence Communication Technology*, pages 62–66, 2015.

[13] B. Wang and X. Tang. Designing a self-adaptive and context-aware service composition system. In *2014 IEEE Computers, Communications and IT Applications Conference*, pages 155–160, 2014.

[14] H. Wang, X. Wang, X. Hu, X. Zhang, and M. Gu. A multi-agent reinforcement learning approach to dynamic service composition. *Information Sciences*, 363:96–119, 2016.

[15] M. Khanouche, F. Attal, Y. Amirat, A. Chibani, and M. Kerkar. Clustering-based and QoS-aware services composition algorithm for ambient intelligence. *Information Sciences*, 482:419–439, 2019.

[16] H. Wang, X. Zhou, X. Zhou, W. Liu, W. Li, and A. Bouguettaya. Adaptive service composition based on reinforcement learning. In *Service-Oriented Computing*, pages 92–107. Springer, 2010.

[17] H. J. Ahn. A new similarity measure for collaborative filtering to alleviate the new user cold-starting problem. *Information Sciences*, 178(1):37–51, 2008.

[18] X. He, Z. He, J. Song, Z. Liu, Y. Jiang, and T. Chua. Nais: Neural attentive item similarity model for recommendation. *IEEE Transactions on Knowledge and Data Engineering*, 30(12):2354–2366, 2018.