Automatic and Intelligent Composition of Pervasive Applications - Demonstration

Kévin Delcourt, Françoise Adreit, Jean-Paul Arcangeli, Kahina Hacid, Sylvie Trouilhet, Walid Younes

Automatic and Intelligent Composition of Pervasive Applications

K. Delcourt, F. Adreit, J.-P. Arcangeli, K. Hacid, S. Trouilhet, W. Younes
Institut de Recherche en Informatique de Toulouse, University of Toulouse, France
Email : {Kevin.Delcourt, Francoise.Adreit, Jean-Paul.Arcangeli, Kahina.Hacid, Sylvie.Trouilhet, Walid.Younes}@irit.fr

***Abstract*—Opportunistic service composition is a novel and disruptive approach for building software in dynamic and open pervasive environments. It aims to tackle the growing complexity of software design in such environments by dynamically providing relevant applications in the absence of explicit user needs: an intelligent engine composes software components that are present in the pervasive environment in order to build user-tailored context-adapted applications, relying on reinforcement learning.**

This demonstration presents the current status of our opportunistic composition prototype: the intelligent engine builds pervasive applications in bottom-up mode from actual software components that are discovered via the UPnP (Universal Plug and Play) protocol, taking into account learned user preferences.

I. Introduction

Today's users are living in cyber-physical pervasive environments that are more and more complex with the increasing number of devices in the Internet of Things. These environments consist of fix or mobile devices, driven by software components using communication networks to operate.

Those devices and components are generally developed, installed, activated, and assembled independently of each other. Due to user and device mobility, software components may appear or disappear with unpredictable dynamics, giving to pervasive environments an open and unstable 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 a solution that automatically and dynamically assembles software components in order to build applications that are adapted to the current state of the environment and the user. In the opportunistic approach, applications are built on the fly in a bottom-up manner from the available components at that time, with no explicit user needs or predefined assembly plans. In this way, applications emerge from the environment, taking advantage of opportunities as they arise. Besides, the user is put in the loop in order to keep control on the environment: she/he decides on the relevance of the emergent application before it is deployed. The composition engine learns from this feedback in order to build relevant applications.

This paper presents the latest developments of our prototype OCE - Opportunistic Composition Engine, previously discussed in more details in [1]. Indeed, OCE is now able to build, from the bottom up, fully operational component-based applications, using actual software components discovered via the UPnP - Universal Plug and Play - protocol and taking into account the user feedback.

For this purpose, we have developed a Java API, allowing to create *UPnP components* with provided and required interfaces. In order to assess the validity of OCE, a demonstration has been built around a realistic use case in the context of the AILP project[1]: a user in his car benefits from an emerging application that guides him to the charging station closest to his current position, without explicitly asking for it.

This paper is structured as follows: Section 2 presents a quick summary of component-based software engineering, Section 3 presents the global architecture of our solution, focusing on OCE main original principles, and Section 4 details the use case of the demonstration. Section 5 concludes and states the current limitations of our prototype.

II. Software components and composite services

Component-based software engineering[2] consists in building software as assemblies of reusable and versatile *software components*. This paradigm emphasizes composability, reuse and software flexibility: an application can be modified by replacing one component by another in the assembly.

Software components [3] are runtime units that implement and *provide* services. Symmetrically, they exhibit the services they *require* to be operational. Composing them consists in binding required services to matched provided ones, and finally building applications with added value. Fig. 1 shows the example of a component-based implementation of a location application, made from a GPS component and a Map component. The Map DrawPoint provided service is bound to the GPS SetLocation required service, this application allowing the measured coordinates to be displayed on the map.

Traditional approaches for designing component-based applications are problematic when applied to open dynamic pervasive environments: at design time, it is difficult to predict which components and devices will actually be available at runtime. Furthermore, building applications on the basis of *a priori* stated user needs may lead to disregard arising needs and to miss opportunities the environment could bring.

[1]AILP (Assistance InteLligente et proactive en environnement Professionnel) is supported by the French region Occitanie and the operational program FEDER-FSE Midi-Pyrénées et Garonne.

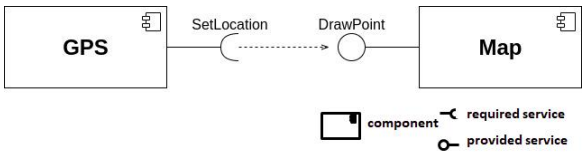

Fig. 1: A component-based location application

III. User-oriented opportunistic composition

Fig. 2 shows the overall architecture of the composition system, which consists of three main elements: OCE, ICE - Interactive Control Environment - and the user.

A. OCE - Opportunistic Composition Engine

At the center, OCE has the main task of assembling the components. It periodically senses the pervasive environment in order to discover available components, connects corresponding provided and required services with each other and thereby constructs on the fly composite applications.

OCE is designed in a MAS - Multi-Agent System - architectural style [4] which is known to meet main challenges raised by pervasive environments: decentralization, distribution, scalability, dynamics and adaptability. Any service sensed in the pervasive environment is managed by a dedicated agent [1]. The agents cooperate in order to choose the correct and pertinent connections to realize between the available components. Agent's decision is based on its local view of the assembly and of the environment, and its estimated values of the other agents: these values are computed from the feedback gathered throughout the previous OCE executions.

B. ICE - Interactive Control Environment

ICE allows to put the user in the loop: as soon as an application emerges, i.e., is assembled by OCE, ICE presents it to the user. Using model-driven engineering, ICE aims to present emerging applications in an intelligible manner: several graphical representation styles are available, from abstract ones to concrete and technical ones. Non-expert users can choose the one that best suits her/his preferences [5].

Thanks to ICE, the user can either accept, reject or modify the proposed application. Feedback data (including the newly produced composition model if the application has been modified) are then deducted from these actions. This feedback is central in our approach as it allows OCE to learn and build knowledge about the user related to the current situation, in an endless online reinforcement learning mode [6].

IV. Demonstration

A. Demonstration background

The early version of our prototype, presented in [1][2], exploited dummy components loaded from an XML file, with no actual implementation.

In this new demonstration, we show the current status of our solution, which now uses actual software components distributed on multiple devices. Thus, we demonstrate how our solution works in a pervasive environment where components can appear and disappear dynamically.

Those components rely on the UPnP protocol [7] for discovery, and connect to each other through the user's local network. We have developed an API to build *UPnP components* that can be detected by OCE[3]. This API includes a wrapper class designed to add the concept of *required* service to the definition of a UPnP component, which is not available in the UPnP specification [7]. OCE senses the UPnP components in the environment, reads the description of their provided/required services and can issue binding orders to the components.

B. Use case: seeking an outlet for an electric car

A demonstration video based on the following scenario is available[4]. Consider Paul driving his electric car, in need of a power outlet. With the assistance of our solution that makes emerge an application from the pervasive environment, he is able to locate the nearest available power outlet.

Since Paul is traveling with his Android smartphone, both components of the phone and the car are present in the environment. The latter contains:

- GPS components, locating Paul's car and phone.
- An AndroidMap component displaying points and routes.
- BatteryLevel components, giving information on the car and phone battery level.
- An OutletLocalizer component, which requires a map and a BatteryLevel component in order to locate the nearest available outlet from a set of coordinates.

All components presented in the demonstration have been fully developed and are functional. Although it is possible, the car components aren't deployed on a true car, but on a separate desktop computer; as such, they provide simulated positions and battery levels. Thus, the pervasive environment presented in this demo is actually composed of one desktop computer and one Android device.

1) Initial composition: When starting the engine, the agents have no knowledge about the environment or Paul's preferences: the first assembly is built in an exploratory manner using the available components, then it is presented to Paul. Fig. 3 shows a screenshot of ICE, with this assembly.

Then, using ICE, Paul edits this proposition, giving his feedback. He makes changes according to:

- his current need: to charge his car's battery; therefore he chooses CarBatteryLevel over PhoneBatteryLevel.
- his preferences: both the PhoneGPS and CarGPS components can locate him, thus he chooses CarGPS.

Fig. 4 presents the resulting application. OCE learns from this feedback: each agent updates its knowledge. It also carries

[2]A video that demonstrates the early prototype is available online at https://www.irit.fr/%7eSylvie.Trouilhet/demo/wetice2020.mp4

[3]https://github.com/KevinDelcourt/UPnPComponents

[4]https://www.irit.fr/%7eSylvie.Trouilhet/demo/outletSeeking.mp4

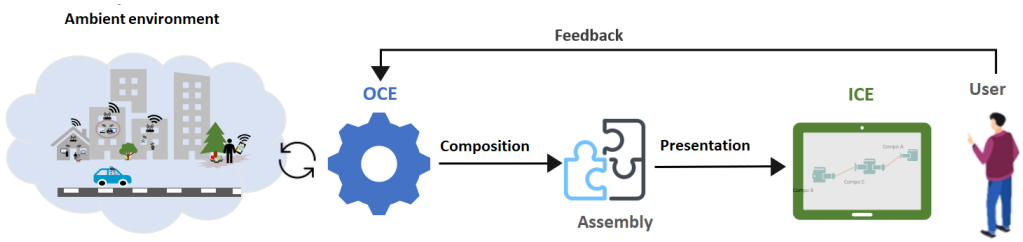

Fig. 2: General architecture of the composition system

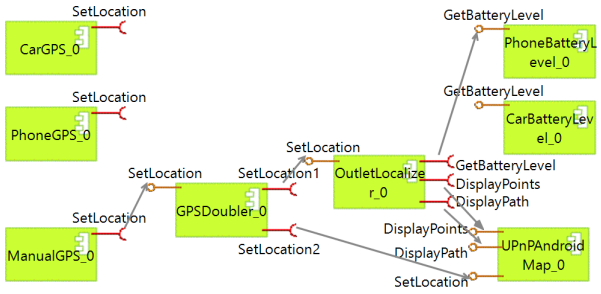

Fig. 3: First assembly proposition - ICE screenshot

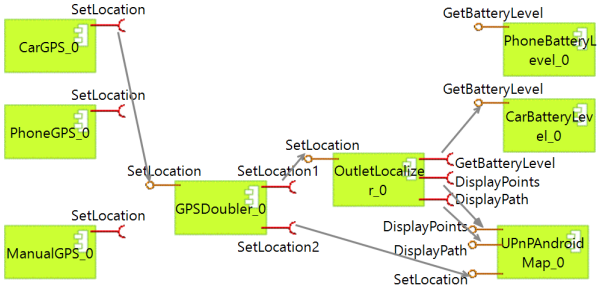

Fig. 4: First assembly proposition, edited by the user

out the bindings between the components: Paul can now see the route to the nearest outlet.

2) Disappearance of OutletLocalizer: The OutletLocalizer component is turned off, breaking the previous application. This change in the environment is sensed by OCE, which in turn proposes a new application, using its acquired knowledge. Reusing the preferred components, CarGPS and AndroidMap, OCE proposes a new application (Fig. 5) that displays the car's position on Paul's phone. Paul accepts it, sending positive feedback to OCE which orders the component bindings.

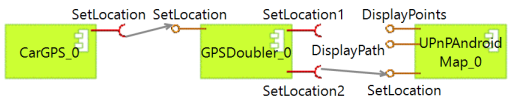

Fig. 5: Assembly after disappearance of OutletLocalizer

3) Appearance of components: We show the case of the appearance of two components: GPSToCityConverter which converts a set of coordinates into the corresponding city name, and TextReceiver which displays text. OCE composes a new application, giving priority to Paul's preferred components as well as the new ones in order to build a potentially interesting application. Fig. 6 shows a part of the resulting application: it displays the name of the city where Paul is located as well as its position on a map. It is then accepted and deployed.

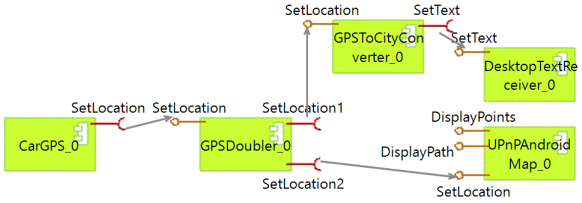

Fig. 6: Assembly after appearance of new components

V. Conclusion

Additional tests on realistic use cases, with real-world users and components, are necessary to consolidate the validation of our solution. For example, scalability problems could arise with high numbers of components and extended uses.

However, our experiences show that the current prototype works. It builds on the fly emergent applications depending on the situation, and proposes them to the user. As the user goes along, it learns about his/her preferences, and uses this knowledge to make the next proposals, more and more relevant. The engine does not require the prior expression of the user's needs to propose an assembly, and that it is sensitive to changes in the environment.

References

[1] 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.

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

[3] OMG. *Unified Modeling Language*, chapter 11.6. 2017. https://www.omg.org/spec/UML/2.5.1/PDF.

[4] M. Wooldridge. *An Introduction to MultiAgent Systems*. Wiley, 2009.

[5] M. Koussaifi, J.-P. Arcangeli, S. Trouilhet, and Bruel J.-M. Model-Driven Engineering for End-Users in the Loop in Smart Ambient Systems. Technical report IRIT/RR–2020–06–FR, IRIT, September 2020. https://www.irit.fr/~Sylvie.Trouilhet/publiInfo/IRIT_RR_2020_06_FR.pdf.

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

[7] C. Lee and S. Helal. Protocols for service discovery in dynamic and mobile networks. *Int. J. of Computer Research*, 11(1):1–12, 2002.