Automatic and Intelligent Composition of Services in Ambient Environments: A Demonstration

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

Institut de Recherche
en Informatique de Toulouse

UNIVERSITÉ
TOULOUSE III
PAUL SABATIER

Automatic and Intelligent Composition of Services in Ambient Environments: A Demonstration

IRIT/RR–2020–07–FR

September 2020

Kévin DELCOURT
Françoise ADREIT
Jean-Paul ARCANGELI
Kahina HACID
Sylvie TROUILHET
Walid YOUNES

Institut de Recherche en Informatique de Toulouse - UMR 5505
Université Paul Sabatier, 118 route de Narbonne, 31062 TOULOUSE cedex 4

Abstract

This report demonstrates our prototype solution of opportunistic software composition through a realistic use case: a user in his car benefits from an emerging service that guides him to the battery recharging station closest to its current position. The demonstration video is available online.

Keywords

Ambient Intelligence, Software Component, Service, User-Centered Composition, Emergence, Use Case, Demonstration

Acknowledgment

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.

Chapter 1

Introduction

Opportunistic service composition is a novel and disruptive approach for building software and services in ambient and IoT dynamic and open environments. In the absence of a priori explicited needs, services that are adapted to the user and the situation emerge from the current environment. They are composed of software components and composition is made automatically in bottom-up mode by an intelligent engine that learns user needs and preferences by reinforcement according to the situation.

In the following, we introduce software components and opportunistic software composition. Then, we present our approach and the overall architecture of our solution. At last, we demonstrate our prototype solution through a realistic use case provided in the context of the AILP project[1]: a user in his car benefits from an emerging service that guides him to the battery recharging station closest to its current position. The demo relies on an interface dedicated to experiment, the intelligent engine, and true software components.

[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. AILP associates academic and industrial partners.

Chapter 2

Background and Problem

Component-based software engineering [1] consists in building software as assemblies of reusable and versatile *software components*. This paradigm emphasizes composability and reuse. It fosters software flexibility: an application can be modified by replacing one component by another in the assembly.

Software components [2] are runtime units that implement and *provide* services. Symmetrically, at the same level as the services they provide, software components exhibit the services they *require* to be operational. Thus, since the provided and required interfaces are explicit, they are easily composable. Composing components consists in binding the components' required services to provided ones to deliver composite services with added value. In order to make a component fully operational, i.e., actually provide its services, its required services must be bound to (so realized by) a service that is provided by another component. Fig. 2.1 shows the example of the component-based implementation of a service that provides a lighting service.

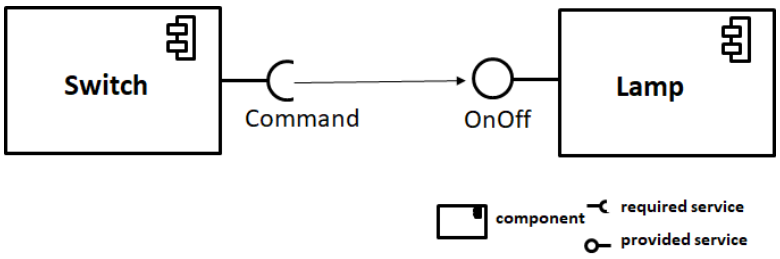

Figure 2.1: Implementation of a component-based lighting service

The challenge is to design an intelligent engine that proposes usable applications and ensures proactivity and adaptability of execution time in ambient environments, e.g., in a context of openness, dynamics and unpredictability.

For this purpose, we propose an original and disruptive approach for building software and services from software components[1]: the opportunistic software composition. Applications are built on the fly in a bottom-up manner from the components that are present and available at that time, without the user needs to be made explicit. In this way, composite applications emerge from the environment, taking advantage of opportunities as they arise. Context-adapted applications are provided in

[1] In [3], authors show how, relying on an ontology, developers may be assisted when assembling components to build a composite service and how composite service descriptions may automatically be generated by combining component unit descriptions.

"push mode", unlike traditional service engineering "pull mode" where the user specifies the needed service. Furthermore, the user is put in the loop, keeps control of the compositions and finally decides on their pertinence.

Chapter 3

User-oriented automated composition

Figure 3.1 shows the overall architecture of the composition system. OCE - Opportunistic Composition Engine - which has the main task of assembling the components is placed at the center of our architecture. OCE periodically senses the ambient environment in order to discover available components, connect corresponding provided and required services with each other and thereby construct on the fly composite applications.

As the user has to be informed of the new generated applications to be deployed on her/his devices, she/he is integrated in the architectural loop: as soon as an application emerge (assembled by OCE), a graphical user interface called ICE - Interactive Control Environment - presents it to the user for control. One of our research axes is to use model-driven engineering to present emerging applications in an intelligible manner. Several graphical representation styles are available on ICE, from the most abstract one to the most concrete and technical one. The user chooses the one that best suits her/his comprehension capabilities [4].

With ICE, the user can either accept, reject or modify the proposed application. Feedback data is deducted from these actions. This feedback is central in our approach as it allows OCE to learn and define the user profile and preferences in an endless online reinforcement learning mode [5]. The response given by the user is the unique form of feedback required from her/him.

OCE is designed in a multi-agent system (MAS) architectural style [6] which is known to meet main challenges raised by an ambient environment: decentralization, distribution, scalability, dynamics and adaptiveness. A service sensed in the ambient environment is managed by a dedicated agent [7]. An agent communicates with others in order to choose the correct and pertinent connection to

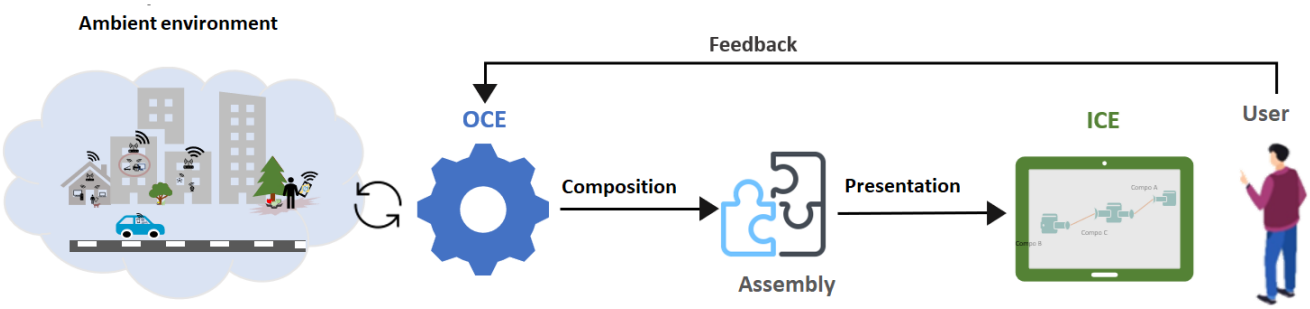

Figure 3.1: General architecture of the composition system

realize within the available ones. This decision is based on the agent's local view of the assembly, the ambient environment and its estimated values of other agents: these values are computed from the feedback collected during the previous OCE executions. Thus, learning and decision are distributed through the MAS and each agent decides for its service.

Chapter 4

Demonstration

We have developed a proof-of-concept implementation of our solution. Besides, a demonstration of our reinforcement learning algorithm has already been presented in [7][1].

4.1 Description of the Demonstration

The objective of this demonstration is to show through a composition example that the system works with real components to produce a usable application. The video of this demonstration is available at `https://www.irit.fr/~Sylvie.Trouilhet/demo/DelcourtM1Demo.mp4`. The components featured in this demonstration are based on the UPnP (Universal Plug And Play) protocols [8], allowing them to discover and connect to each other through the user's local network. For example the Car-GPS component randomly generates the position of the user (to act as if he was moving in his car). An API is available to build UPnP components that can be detected by OCE at `https://github.com/KevinDelcourt/UPnPComponents`. This API includes a wrapper class designed to transform a web service into a component-based service.The distinction between the two is that a web service makes available its provided interfaces only whereas a component-based service will allow others to connect to its provided and required interfaces for composition.

For experimentation purposes only, we use a graphical interface that allows to monitor and follow step by step the engine operations and to inspect the agents (in particular the evolution of their knowledge on the encountered situations). Note that this demonstration can be reproduced regardless of the composition of the environment (number and types of components). Though, we have limited the number of components in order to not overload the demo.

The removal of a component (OutletLocalizer in our demonstration) leads to a new engine cycle and the proposal of a new assembly. Likewise, several composition proposals may follow one another, in particular if a component (OutletLocalizer in our demonstration) reappears or if a component offering an equivalent service appears.

The current prototype offers emergent assemblies that it builds on the fly depending on the situation. It is dedicated to a user and learns as it goes along about his/her preferences, which it uses for the following assembly proposals. The process goes up to the physical connection of the components. So the application runs on the user's final device (the smartphone screen is inlaid in the demo to show how the application works after its deployment on the user phone). Indeed, when the user

[1] A demo of the learning solution is available at `https://www.irit.fr/~Sylvie.Trouilhet/demo/wetice2020.mp4`

validates the description of an application, the assembly is actually carried out and it is the latter that is used in the demonstration.

Testing shows that the engine does not require to know the user's needs to propose an assembly, and that it is sensitive to changes in the environment. It also shows that if the same situation arises, the engine may not make the same proposal because between the two moments, it will have learned from the user's preferences and will use them to adapt its decisions. OCE is therefore an AI that learns and is sensitive to its ambient environment in order to automatically compose emergent applications tailored to the user it assists.

4.2 Demonstration Requirements

In order to run the demonstration presented in this report, the following elements are required:

- A Windows Computer, running the OCE test interface
  `https://github.com/SylvieTrouilhet/OCE`,
  the ICE user interface
  `https://github.com/marounkoussaifi/ICE/`
  and the component-based services available at
  `https://github.com/KevinDelcourt/DesktopUpnpComponents`.
- A WiFi router, to which the demonstration devices will connect.
- One Android 7.0 device, running the component-based services available at those Web addresses:
  - `https://github.com/KevinDelcourt/UPnPAndroidMap`
  - `https://github.com/KevinDelcourt/UPnPAndroidGPS`
  - `https://github.com/KevinDelcourt/UPnPAndroidComponents`
- If desired, additional Android devices and/or desktop computers running Java 11 can be included for a more complex demonstration.

With all the components running on the same local network, one can use OCE and ICE to experiment with the composition engine and build applications, following or not the use case showed in the demonstration presented in this report.

Bibliography

[1] I. Sommerville, "Component-based software engineering," in *Software Engineering*, ch. 16, pp. 464–489, Pearson Education, 10^{th} ed., 2016.

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

[3] G. Alary, N. Hernandez, J.-P. Arcangeli, S. Trouilhet, and J.-M. Bruel, "Comp-O: an OWL-S Extension for Composite Service Description," in *22nd International Conference on Knowledge Engineering and Knowledge Management (EKAW)*, LNCS, Springer, 2020. https://hal.archives-ouvertes.fr/hal-02986555v1.

[4] M. Koussaifi, J.-P. Arcangeli, S. Trouilhet, and B. 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.

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

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

[7] 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.

[8] C. Lee and S. Helal, "Protocols for service discovery in dynamic and mobile networks," *International Journal of Computer Research*, vol. 11, no. 1, pp. 1–12, 2002.