Ambient Intelligence Users in the Loop: Towards a Model-Driven Approach

Maroun Koussaifi, Sylvie Trouilhet, Jean-Paul Arcangeli, Jean-Michel Bruel

Ambient Intelligence Users in the Loop: Towards a Model-Driven Approach

M. Koussaifi, S. Trouilhet, J.-P. Arcangeli, J.-M. Bruel

University of Toulouse, France
Institut de Recherche en Informatique de Toulouse

Abstract. Ambient and mobile systems consist of networked devices and software components surrounding human users and providing services. From the services present in the environment, other services can be composed opportunistically and automatically by an intelligent system and proposed to the user. The latter must not only to be aware of existing services but also be kept in the loop in order to both control actively the services and influence the automated decisions.
This paper first explores the requirements for placing the user in the ambient intelligence loop. Then it describes our approach aimed at answering the requirements, which originality sets in the use of the model-driven engineering paradigm. It reports on the prototype that has been developed, and analyzes the current status of our work towards the different research questions that we have identified.

Keywords: user in the loop, ambient intelligence, service composition, software components, emergence of services, presentation of services, model-driven engineering, model transformation

1 Introduction

Ambient and mobile systems consist of fixed or mobile devices connected by one or several communication networks. These devices host services specified by interfaces and implemented by independently developed, installed, and activated software components. Components therefore provide services and, in turn, may require other services. They are blocks that can be assembled to build more complex services. For example, hardware or software interaction components (*e.g.*, buttons, sliders, screens) and functional components like a Polling Station and a Report Generator can be assembled if their interfaces match and provide a complete distributed "voting service".

Due to the high mobility of current devices and users, the environment is open and highly unstable: devices and software components, which are independently managed, may appear and disappear without this dynamics necessarily being foreseen. Human users are plunged into these dynamic systems and can use the services at their disposal. Ambient intelligence aims at offering them a personalized environment, adapted to the current situation, anticipating their

needs and providing them the right services at the right time, with as little effort as possible.

We are currently developing a solution in which services (in fact, microservices) are dynamically and automatically composed in order to build composite services and so customize the environment at runtime. Here, unlike the traditional "top-down mode" for building applications, services are built on the fly in "bottom-up mode" from the components that are present and available at runtime. This is supported by an *assembly engine* in line with the principles of autonomic computing and the MAPE-K model [11]: it senses the existing components, decides of the connections (it may connect a required service and a provided one if their interfaces are compatible) without using a pre-established plan (or not necessarily), and commands them. The heart of this engine is a distributed multi-agent system where agents, close to the software components, cooperate and decide on the connections between their services. Composite services (realized by assemblies of components) continuously emerge from the environment, taking advantage of opportunities as they arise. And to make the right decisions and offer the relevant services, the engine (*i.e.*, the agents) learns at runtime by reinforcement. The main advantages are proactivity and runtime adaptation in the context of openness, dynamics and unpredictability [15].

The user is at the core of ambient or cyber-physical systems. Here, unlike the traditional SOA paradigm, she/he does not necessarily demand or search for services (in "pull mode"); on the contrary, services adapted to the context and operational are supplied in "push mode". In this context of automation based on artificial intelligence, the sharing of decision-making responsibilities between the assembly engine and the user is in question. Anyway she/he must be kept "in the loop". On the one hand, it is essential to assist the user in the appropriation and control of the pushed services: she/he must be informed but also must keep some control over her/his ambient environment, or possibly be able to contribute herself/himself to the construction of personalized services. On the other hand, to make the right decisions, the assembly engine must rely on a model of the user in her/his environment. This model, which is unknown a priori, must be built at runtime and evolve dynamically.

Keeping the user in the loop therefore demands a number of requirements to be met. The objective of this work is to experiment and evaluate a solution based on model-driven engineering and model transformations in order to put the user in the control loop. The purpose of this paper is to explain and justify the interest of such an approach, to describe the main architectural principles, and to report on the development of a prototype solution (the design of the smart engine itself is out of the scope of this paper). The conducted experimentation allows us to conclude positively on the advantages of such an approach.

The paper is organized as follows. Section 2 describes in more details the problem through a use case. The concrete issues raised by the specifics of the domain, listed as requirements. Section 3 analyses the current state of the art and concludes that there is no current solution that fully addresses the requirements. Section 4 presents our initial ideas to address the research questions identified.

Section 5 presents the prototype we have developed and experimented in order to validate our approach. Finally, a conclusion is given in Section 6 as well as the perspectives of this work.

2 Use Case and Requirements

2.1 Use case

In order to illustrate the problem and motivate the requirements, we propose the following use case, divided into two phases: the first one describes an opportunistic adaptive service composition and the second one the emergence of an unanticipated service.

MissJane is a student at the university. This morning, she has a formative assessment: the teacher asks some questions and the students answer using a *Remote Control* device lent by the university for the year. The answers are collected by the teacher who makes comments in return. For that, the teacher activates a *Quiz* service implemented by three software components: a *Polling Station* available on the university network, a *Report Generator* and a *Remote Control* installed on his laptop. Then, the services provided by the students' remote controls connect with the required service of the *Polling Station* component. Unfortunately, MissJane has forgotten her remote control at home and is unable to answer. However, the ICE (Interactive Control Environment) interface which is at her disposal in order to control her smart environment suggests the use of a vertical slider currently available on her smartphone instead of the remote control. Even though it was not originally designed to be used with the *Quiz* service but as it matches the required service of the *Polling Station*, MissJane can use it, at least if she agrees, and therefore answer. In fact, the ICE interface could have suggested several other compatible interaction components (as an horizontal slider or a dimmer switch) also available in the environment. Then, MissJane would have chosen her favorite one.

Here, several available components have opportunistically been assembled by the smart engine. Then the resulting *Quiz* service that is adapted to the context has been presented to MissJane, who used it after acceptance. The corresponding assembly is depicted on the right side of Fig. 1 (we voluntarily use an informal notation of components and connections). Note that, in this example, we do not consider how the quiz questions are displayed to the students.

The course in now terminated. MissJane frequently goes to her favorite pub in the afternoon. To book a table and order drinks, she uses an *Order* service (see the left side of Fig. 1) implemented by three components (*Customer Input Interface*, *Menu Presentation*, *Order Generator*) provided by the pub and installed on her smartphone. As today it's her birthday, she would like to invite the other students to have a drink. But she doesn't want to enter all of the orders manually. Thus, she deactivates her *Customer Input Interface*. Then, in such a context, the assembly engine proposes to bind the *Order Generator* component with the *Polling Station* still available in the environment, instead of the *Customer Input Interface*. Now, the new *Pub4.0* service allows each student to order

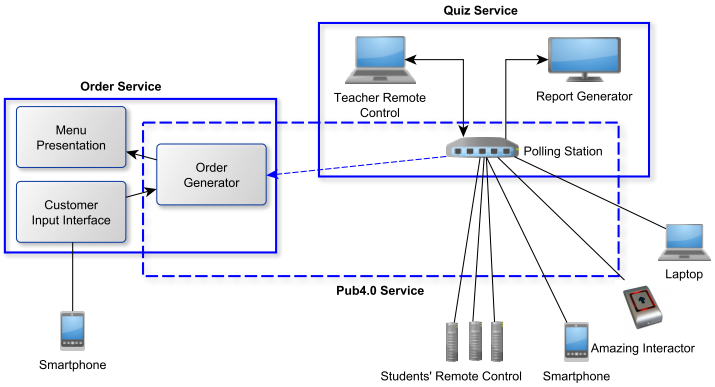

Fig. 1. Emerging Composite Services

her/his own drink with her/his remote control, and sends the global order to the pub. This service really emerges from the ambient environment as it was not designed beforehand and it is built from non-dedicated components provided by different authorities. The *Pub4.0* service is in the dotted frame of Fig. 1.

2.2 Requirements

Our goal in this project is to put the user in the loop. To achieve this goal, we have identified several requirements listed below. In a general way, the user must be aware of the emergence of new services that are pushed by the assembly engine, have the privilege to control this emergence, and be able to appropriate the services. On the other side, to improve its decisions, the intelligent assembly engine needs to learn from the user's actions and reactions to the situation and the proposals of services. We have organized the requirements in main concerns.

Presentation: An emerging service must be presented to the user. As unanticipated services may appear, the user must be informed of their availability. For example, in our use case, MissJane would receive a notification on her smartphone that she can use the vertical slider as a voting device. This implies that she has accepted to receive such a notification. This also raises some requirements related to acceptability and intelligibility. As a result, the research questions we are interested in are:
PRE 1. How to present an emerging service to a human user in an intelligible and personalized way?
PRE 2. When and how often must the emerging services be presented?

Acceptation: The user must accept or reject an emerging service. After it passes the presentation phase, user acceptation determines if the proposed

service is relevant and has to be deployed or not. In our use case, MissJane would accept the use of the proposed vertical slider. This raises usability requirements: it demands an easy way for the user to accept or reject the emerging service (*e.g.*, MissJane would simply click on the accept button attached to the notification). Some related research questions are:
ACC 1. How the user must be notified that acceptation is required?
ACC 2. How the user could accept or reject an emerging service?

Modification: An emerging service should be modifiable by the user. The user should be able to remove or replace any component in the proposed service, more widely to modify the emerging service. For example, MissJane should be able to change from a vertical slider to an horizontal one. So, the user must have the necessary tools and services to modify the emerging service: alternative components and/or assemblies should be presented, editing should be easy and the user assisted in this task. This concerns the usability and ergonomics of the editing tools. In addition, the permission to use and bind a component,*i.e.*, security concerns, might be considered. Some related research questions are:
MOD 1. How the user can be assisted and tools be helpful?
MOD 2. How to insure, on the spot, that the modified service is still correct?
MOD 3. Which components of the ambient environment should be usable and presented to the user and how?

Creation: A composite service can be created by the user. Like the engine but without using its proposals, the user should be able to create her/his own composite service *i.e.*, an assembly from scratch, out of available components. For example, MissJane should be able to build by herself the *Pub4.0* service instead of the engine. To do this, the user must visualize the available components and be able to bind the ones she/he selects. Besides usability, user assistance, service correctness and relevance, another problem -partly related to scalability- concerns the identification of the available and useful components. This part of the requirements does not bring any new particular research question in addition to those related to the modification concern.

Feedback generation: The assembly engine must receive feedback from the user's actions. When a service is created, modified, accepted or rejected, positive or negative feedback must be generated for the engine, that could help to increase the quality of its decisions and fit to the user's behavior, practices and preferences (this means that a user profile is implicitly built). For instance, breaking a connection between services could trigger negative feedback for the engine in order to decrease the estimated value of the binding. In the same way, setting up a new connection could generate a positive feedback increasing consequently the estimated value of the binding. Thus, for example, after several times MissJane has modified an emerging service by choosing the horizontal slider, the engine would have finally learned her preference and proposed the

service with the horizontal slider as a priority. In addition, when using a graphical editor, the user's actions such as swipe or pinch-spread may give information for the assembly engine. This way, the swipe of a service could mean that this one is interesting, and reinforce the interest of the component which implements this service. Concerning feedback and learning, some research questions are:
FBK 1. How to capture user's intentions from her/his manipulations?
FBK 2. How to translate the observed actions into useful information for the engine?

3 State of the Art

According to [9], as self-adaptive systems (*e.g.*, implementing the MAPE-K model) can behave in unexpected ways, humans must be involved in the adaptation process: they can help in conflict resolution and improve the adaptation strategy by giving feedback, even when they have limited attention or cognition. Transparency, intelligibility, trust to users, controlability, and management of user attention are major requirements. In [8], authors propose a solution to integrate the user in the self-adaptation loop, while usability and preference modeling are the main requirements. Adaptation relies on variability models built at design-time and user-level preferences. In addition, for acceptability and to avoid user trouble, "user focus" components (*i.e.*, components that are in the actual user focus, in opposition to "background" components) are kept out of dynamic adaptation. User contribution can be more or less explicit: she/he can select and adjust an application, accept or reject an application, change her/his preference, or even put off the adaptive behavior.

In order to succeed, putting the user in the loop must meet usability requirements. For that, End-User Development (EUD) aims to enable non-specialists in software development to create or modify applications. Common approaches consists in providing software elements to be customized and composed. According to [13], which reviews different projects in particular concerning mobile applications, a motivation is that "regular development cycles are too slow to meet the users' fast changing requirements". In [6], authors propose an EUD environment designed for home control as an alternative to artificial intelligence. Additionally, they report on their "lived-with" experiences with EUD at home. They conclude that if EUD and machine learning are competing approaches, "it should be possible to augment EUD with machine learning".

In [10], the emphasis is put on feedback and machine learning in adaptive smart homes. Authors argue that user preferences and profile can be learned (by semi-supervised reinforcement learning algorithms), associated to activity recognition that transforms raw data into sharp information about the user situation.

In the domain of human-computer interaction, several solutions for interface plasticity (*i.e.*, dynamic adaptation to changing environments) rely on component or service dynamic composition [7]: automation is demanded to overcome complexity (in number, dynamics, composability...), but keeping the user in

the loop is imperative both to observe and to control the interactive ambient environment. The concept of Meta-UI (User Interface) [5] has been introduced as "the set of functions that are necessary and sufficient to control and evaluate the state of interactive ambient spaces". In [7], we have proposed the Meta-UI to present emerging user interfaces and allow for user's choice in the context of ambient systems.

Regarding the requirements analyzed in Section 2.2, the existing solutions are only partially satisfactory. They are *ad hoc* (EUD environments or Meta-UI), and none of them can support the description and edition of unanticipated emerging services. The next section introduces the principles of our approach, and Section 5 overviews our solution and details the prototype we have realized as a proof of concept.

4 Our approach

From the previous section we can conclude that the problem we address requires to match and master links between concepts. It can be between a service and an assembly of components, between an intent and a set of model manipulations, etc. The key concerns here are: (i) the presentation/manipulation of services, which implies some form of editor, and (ii) the navigation/transformation between concepts. We have hence naturally explored the use of the recent Model-Driven Engineering (MDE) approaches to help in this concern.

Our team has a long experience in providing modeling and language engineering tools and approaches [4]. One of the most recent activity addresses the benefit of having, in the context of Cyber-Physical Systems (CPS) models directly manipulable by the final user in order to pilot and adapt their behavior [3]. Such manipulations are now possible thanks to the progress of language engineering environments such as GEMOC[1] that allow the definition of Domain-Specific Modeling Languages (DSML) and the automated generation of the language workbench that goes with it (graphical and textual editors, transformation languages, etc.).

In order for a human to manipulate concretely a model, a set of elements are required: (i) some tooling (viewers, editors, debuggers, interpreters, ...); (ii) some representations (concrete and abstract syntax, ...); (iii) some interpretations and rules (semantics, grammar, ...). This is the purpose of MDE approaches to provide such environments (see Fig. 2). In our context, we have to extract information from an ambient systems technical world (made of components, bindings, services, etc.) and present them from a user point of view (made of goals, expectations, required services, etc.). MDE will help to make the connections between the two domains by providing: (i) a detailed organization of the concepts of each domain (called metamodels), (ii) a mapping between those concepts, (iii) the required environment to manipulate and navigate between those concepts. The detailed use of MDE to help solving the research questions we have listed in section 2.2 will be given in Section 5.1.

[1] `http://gemoc.org/`

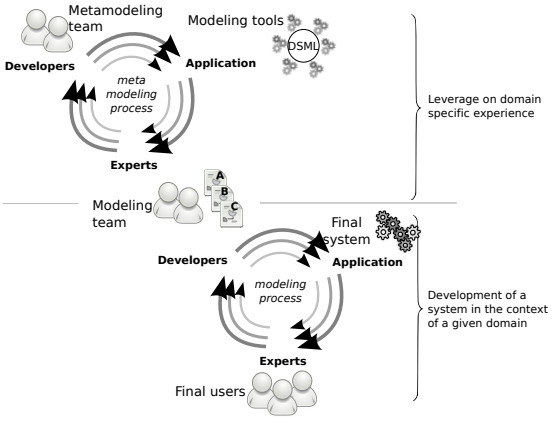

Fig. 2. Model-Driven Engineering in action (taken from [4])

Apart from our own efforts (*e.g.*, [1, 3]) towards putting the final user in the loop of the monitoring and management of his/her own applications, we can cite several other approaches. In [14], the authors use MDE to control user interface adaptation according to explicit usability criteria. They focus on the generation of those interfaces and hence address more the variability concerns that the user interactions themselves. Let us also mention the work from [2], where the authors apply knowledge (inferred from large volumes of information, artificial intelligence or collective intelligence) to boost the performance and impact of a process. They nevertheless do not focus in user interaction. The following section provides details on the way we have implemented MDE techniques to answer the requirements identified in Section 2.

5 Proof of Concept

In order to experiment the base ideas of our approach, we have developed a solution that consists of a specialized model editor for user manipulations and tools to link the models with (an emulated version of) the ambient system. The full source code of our prototype can be found on Github[2]. The first tool generates the model of a service from the output of the assembly engine. The second tool allows the models that are created, modified, or accepted by the user to be deployed in the ambient environment.

Several technologies and frameworks support the implementation. They are used to define a metamodel from which models can be edited using a graphical editing framework, and to transform models by model-to-text transformation into codes that realize the deployment. In practice, we have used the Eclipse Modeling Framework (EMF[3]) which is a basic plugin for metamodeling on

[2] `https://github.com/marounkoussaifi/MDE_Prototype_User_In_The_Loop`
[3] `https://www.eclipse.org/[modeling/emf|sirius|acceleo]`

Eclipse, Ecore to define and create the metamodel, Sirius[3] to define the editor's resources, and Acceleo[3] which is a model-to-text transformation tool, to generate deployment code.

In the following, we present an overview of the implemented approach, and provide some more technical details.

5.1 Overview of the prototype solution

Fig. 3 shows an overview of our prototype solution that is structured in three parts: an *editor*, a *service presenter*, and a *service deployer*.

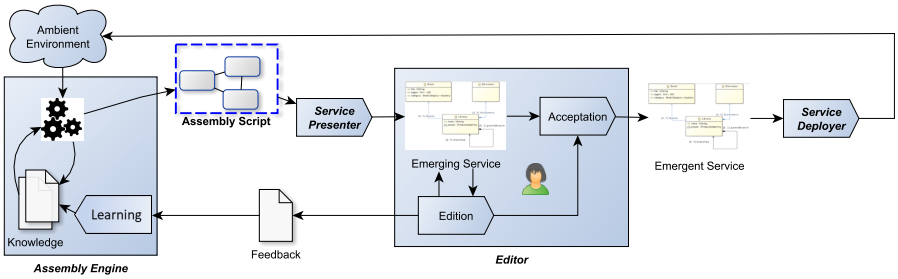

Fig. 3. Implementation of the complete loop

At first, the engine monitors the ambient environment to detect the available components and produces composite services in the form of scripts, *i.e.*, text files defining executable bindings of components. Fig. 4 shows an example of such a script in Java, where the comments have been added by hand for a better understanding.

```
// **** Emergence of the new Pub4.0 service ****
try {
    //Connecting the Order Generator to the Polling Station
    ambientEnv.bind(I2ofC8, I1ofC1, C8, C1);
    //Connecting the Vertical Slider to the Polling Station
    ambientEnv.bind(I2ofC1, I1ofC5, C1, C5);
    //Connecting 3 students remote controls to the Polling Station
    ambientEnv.bind(I3ofC1, I1ofC4, C1, C4);
    ambientEnv.bind(I3ofC1, I1ofC4bis, C1, C4bis);
    ambientEnv.bind(I3ofC1, I1ofC4beta, C1, C4beta);

} catch (BindingFailure e) {
    e.printStackTrace();
}
```

Fig. 4. Script for the assembly of Pub4.0 service

Then the *service presenter* transforms the script into an editable model of the emerging service to be presented to the user. This model conforms to the metamodel we have defined for this purpose (see Section 5.2). Via the *editor*, the service model can be manipulated either in the form of a text (for example for experimented users) or in a graphical form (possibly for non-specialists). Actually, this form can be adapted to the user thanks to the separation between the model and its representation, *i.e.*, the model can be represented in a domain-specific language (DSL). Fig. 5 shows the graphical representation by the *editor* of the emerging *Pub4.0* service proposed by the engine (see Section 2.1). It consists of different components connected together. The students' remote controls are connected to the *Polling Station* by binding the *Vote* services together. Also, MissJane's *Vertical Slider* is used as the master remote control of *Pub4.0*: it's connected to the *Polling Station* by binding the *Master Control* service to the *Value* service. The *Value* service represents a generic type of service which is compatible with different other types, such as the *Master Control* service. In the same way, the *Polling Station* is connected to the *Order Generator* by binding the *Report* service to the *Order* service. Additionally, the *editor* may display several non-connected components which are available for connection if necessary. Once the emerging service is uploaded in the *editor*, the user can accept or reject it. She/he can also modify it, that is to say remove or change any binding between the components and use available components if one exists, or even define a new service by creating a new assembly.

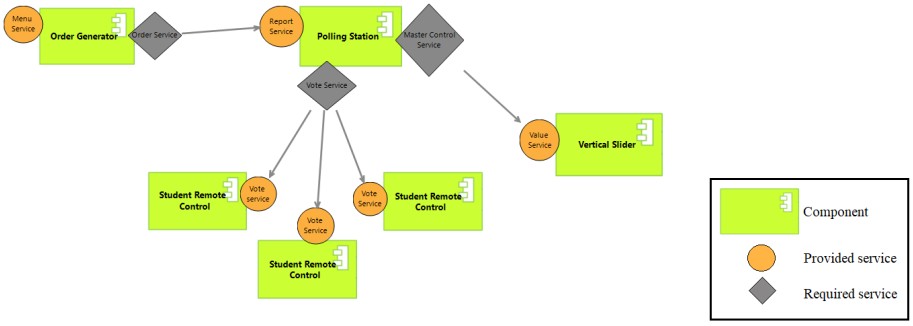

Fig. 5. Presentation of the Pub4.0 service

When editing, in order to generate feedback for the engine to enrich the agents' learning process, user's actions on bindings (in general on the interactive interface) are captured. The engine knowledge hence increases and therefore the engine future decisions will be more in line with the user expectations and profile.

At last, the emergent[4] service is transformed by the *service deployer* into a script to be executed in the ambient environment.

5.2 Service edition

The graphical *editor* is the core element of the answer to the identified requirements listed in Section 2.2. It realizes the ICE interface introduced in Section 2.1: basically, it allows for visualization of emerging service models, service acceptation, and modification, deletion or creation of links between services. In addition, as a graphical editor, it enables to drag and drop any displayed component.

The *editor* relies on a metamodel that frames the definition of component assemblies as a service. The metamodel is classically defined by a class diagram relating together the metamodeling concepts (see Fig. 6). The figure was automatically generated by the Sirius Ecore Editor, a tool that allows the graphical representation and edition of an Ecore model (metamodel). It consists of three main classes. The *Service* abstract class is extended by two child classes, the *ProvidedService* and the *RequiredService* classes. The ambient environment (*ambientEnv* class) is composed of components (*Component* class). Components are composed themselves of at least one service (*Service* class). Bindings between component services are made to build the emerging service. Additionally, we have implemented Object Constraint Language (OCL) [16] rules to constrain the service models (*e.g.*, to control that a service does not exceed a maximal number of connections).

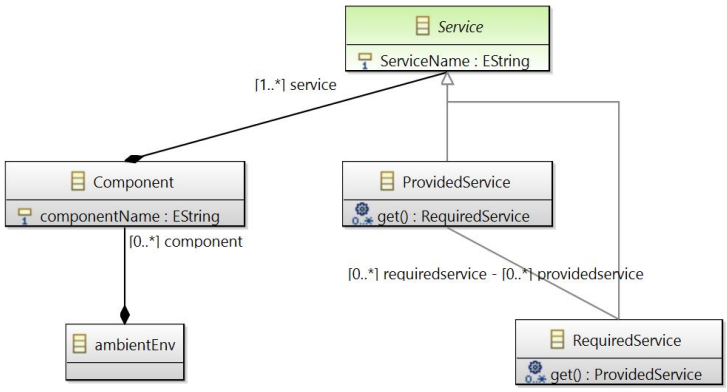

Fig. 6. Our Service Metamodel

[4] We deliberately use *emerging* to qualify services that are dynamically appearing. We reserve the use of *emergent* for emerging services that have been accepted by the user.

To develop the *editor*, we have used the GEMOC Studio, and more precisely Sirius, a technology for designing customized graphic modeling tools. Sirius allows to define editors in a completely graphical way, without having to write any code. This is where the use of an MDE approach takes all its sense. Indeed, with such a strong coupling between the tool and the concepts, it is important that we take into account, in advance, the future evolutions of our metamodel. As the editor is completely automated from the metamodel itself, the metamodel evolutions have no impact on the editor from a workload point of view. At this point of our work, feedback generation has not yet been implemented. The GEMOC monitoring capabilities will be used in order to generate feedback for the learning process of the engine.

The *editor* should be integrated into the whole system and the ambient environment for example to present a short list of available components that are not connected but relevant for use (*e.g.*, an horizontal slider). At this stage, our *editor* is not fully integrated. Nevertheless, in order to test our prototype and simulate the arrival of new components, we have added to the editor a side panel that allows the user to design any component with its services.

5.3 Service presentation

The *service presenter* is the element of our solution that transforms an emerging service into an editable model (which in turn will be presented to the user via the *editor*, as described in the previous section). Unlike the *editor*, the *service presenter* is not a user-manipulated tool.

The *service presenter* relies on the same metamodel as the *editor* to generate the model of the emerging service. Nevertheless, unlike for the editor but for fast prototyping concerns, we have not yet used an MDE approach to implement the *service presenter*. For the moment, we have developed a Java program that records all the bindings between different components, then generates the XML source of the graphical model while respecting the *editor* metamodel concepts. Whenever the metamodel changes, the *service presenter* must be rebuilt in order to become compatible with these changes. Using a MDE approach will be one of the major evolution of our future work.

However, in its current form, the *service presenter* is fully operational and is able to generate the model of an emerging service in the form of an assembly of components.

5.4 Service deployment

The *service deployer* is the element of our solution that generates the bindings commands to be executed for the actual deployment of an emergent service in the ambient environment. It is also a non-user manipulated tool.

Likewise the *editor*, the *service deployer* must rely on the same metamodel while executing model-to-text transformation in order to properly generate the binding script. At this stage, this part suffers from the same limitation than the presenter described previously.

The *service deployer* consists of an Acceleo program that performs model-to-text transformation. Acceleo is an open source code generator from the Eclipse foundation. It allows the design of code generation modules that can generate outputs in a language chosen by the developer from one or more models as inputs. Currently, the *service deployer* performs model-to-Java code transformation.

At this stage of our work, the generated Java code implements the emergent service model to be injected in the ambient environment. This is enough for rapid prototyping and test.

6 Conclusion and Future Work

Infrastructure automation, commonly based on continuous integration, automated testing and deployment, helps in microservices management [12]. Our project aims to go a step further in this direction by automating the assembly of services that are available in the environment and operational. In such a context, the user must nevertheless be put into the loop to be informed of emerging services, to be able to edit, modify, validate them, and to give implicit feedback to the automatic system.

In this paper, we have proposed an MDE-based approach intended to answer the requirements to place the user in the ambient loop. The solution consists of an *editor* that enables the user to visualize an emerging service provided by a *service presenter*. Also, it enables her/him to accept or edit the service, before deployment by a *service deployer*. In such a way, the user is a full actor in the ambient system, especially as her/his actions may produce feedback for the intelligent system. At this stage of our work, tools for service presentation (*service presenter*) and deployment (*service deployer*) are working but should be consolidated *via* a full MDE-based development.

In the following, we discuss the current status of our solution towards the nine research questions we have identified. This discussion is summarized in Table 1 where the status regarding research questions are rated from none to three +.

Research Question	Current Status
PRE 1 (How to present)	+
PRE 2 (When to present)	+
ACC 1 (How to notify user)	
ACC 2 (How the user accept)	++
MOD 1 (Help in manipulation)	++
MOD 2 (Correctness)	+++
MOD 3 (What to present)	+
FBK 1 (How to capture intentions)	+
FBK 2 (Feedback for the engine)	

Table 1. Current status of our solution towards the identified research questions

The first group of research questions is directly related to the MDE-based approach we adopted in order to put the user in the loop: PRE 1, MOD 2, FBK 2. The experience presented in this paper shows that MDE meets the requirements of service presentation and editing, whereas the services are correct by construction since they conform to the metamodel. In addition, as the concrete service representation is separated from the service model itself, any dedicated language that is familiar to the user can be used (DSL). So, we do not expect any particular service manipulation abilities from the user; in the contrary we consider that it is up to ICE to adapt to the user. On the other hand, the view is currently only structural but does not present the function of the emerging service (neither of the components). Likewise, if a certain number of user actions can be observed, they still need to be interpreted in a way that is useful for learning. These points are fundamental, so we aim for a +++ level of response. To meet this objective, and fulfill intelligibility requirements both for the user and the engine, important work remains to be done concerning the enhancement of the metamodel and the transformation rules.

A second group of research questions concerns problems related to Human-Computer Interactions (HCI): PRE 2, ACC 1, ACC 2. They mainly concern acceptability, usability, and ergonomics. For the moment, the ambient environment and its changes are sensed periodically; at the same frequency, new emerging services are presented if there exist. We still have to deal with problems related to environment instability, awareness of user preferences, obtrusiveness or ergonomics in order to reach a solution rated between + and ++. Our proposal will rely on solutions elaborated in the HCI domain, and we do not really aim for a major contribution to the state of the art.

The last questions are strongly related to Artificial Intelligence issues: MOD 1, MOD 3, FBK 2. Currently, the *editor* supports the presentation of emergent services proposed by the intelligent system. We should go further in the choice of relevant services and components to present according to the context (user profile, situation...), and in the assistance to the user. Another challenge sets in the translation of user actions into learning knowledge useful to the engine. As these aspects are essential, we aim for a level response rated from ++ to +++. The further development of the engine's intelligence and its coupling with ICE will provide answers.

References

1. Bruel, J.M., Combemale, B., Ober, I., Raynal, H.: MDE in Practice for Computational Science. In: Int. Conf. on Computational Science. Reykjavík, Iceland (Jun 2015), https://hal.inria.fr/hal-01141393
2. Cabot, J., Clarisó, R., Brambilla, M., Gérard, S.: Cognifying Model-Driven Software Engineering. In: Seidl, M., Zschaler, S. (eds.) Software Technologies: Applications and Foundations. pp. 154–160. Springer International Publishing, Cham (2018), https://link.springer.com/chapter/10.1007/978-3-319-74730-9_13
3. Combemale, B., Cheng, B.H., Moreira, A., Bruel, J.M., Gray, J.: Modeling for Sustainability. In: Modeling in Software Engineering 2016 (MiSE'16). ACM, Austin, USA (2016), https://hal.inria.fr/hal-01185800

4. Combemale, B., France, R., Jézéquel, J.M., Rumpe, B., Steel, J.R., Vojtisek, D.: Engineering Modeling Languages. Chapman and Hall/CRC (Nov 2016), https://hal.inria.fr/hal-01355374
5. Coutaz, J.: Meta-user Interfaces for Ambient Spaces. In: Proc. of the 5th Int. Conf. on Task Models and Diagrams for Users Interface Design. pp. 1–15. TAMODIA'06, Springer-Verlag, Berlin, Heidelberg (2007), http://dl.acm.org/citation.cfm?id=1756988.1756990
6. Coutaz, J., Crowley, J.L.: A First-Person Experience with End-User Development for Smart Homes. IEEE Pervasive Computing 15, 26 – 39 (May 2016), https://doi.org/10.1109/MPRV.2016.24
7. Degas, A., Trouilhet, S., Arcangeli, J.P., Calvary, G., Coutaz, J., Lavirotte, S., Tigli, J.Y.: Opportunistic Composition of Human-Computer Interactions in Ambient Spaces. In: Workshop on Smart and Sustainable City (Smart World Congress 2016 & Int. Conf. IEEE UIC 2016). pp. 998–1005. IEEE Computer Society (2016), http://oatao.univ-toulouse.fr/18769/
8. Evers, C., Kniewel, R., Geihs, K., Schmidt, L.: The user in the loop: Enabling user participation for self-adaptive applications. Future Generation Computer Systems 34, 110–123 (May 2014), https://doi.org/10.1016/j.future.2013.12.010
9. Gil, M., Pelechano, V., Fons, J., Albert, M.: Designing the Human in the Loop of Self-Adaptive Systems. In: García, C.R., Caballero-Gil, P., Burmester, M., Quesada-Arencibia, A. (eds.) 10th Int. Conf. on Ubiquitous Computing and Ambient Intelligence. pp. 437–449. Springer International Publishing (2016), https://link.springer.com/chapter/10.1007/978-3-319-48746-5_45
10. Karami, A.B., Fleury, A., Boonaert, J., Lecoeuche, S.: User in the Loop: Adaptive Smart Homes Exploiting User Feedback—State of the Art and Future Directions. Information 7(2) (Jun 2016), https://doi.org/10.3390/info7020035
11. Kephart, J.O., Chess, D.M.: The vision of autonomic computing. Computer 36(1), 41–50 (Jan 2003), https://doi.org/10.1109/MC.2003.1160055
12. Lewis, J., Fowler, M.: Microservices (2014), https://martinfowler.com/articles/microservices.html
13. Paternò, F.: End User Development: Survey of an Emerging Field for Empowering People. ISRN Software Engineering 2013 (Apr 2013), https://doi.org/10.1155/2013/532659
14. Sottet, J.S., Calvary, G., Coutaz, J., Favre, J.M.: A model-driven engineering approach for the usability of plastic user interfaces. In: Gulliksen, J., Harning, M.B., Palanque, P., van der Veer, G.C., Wesson, J. (eds.) Engineering Interactive Systems. pp. 140–157. Springer, Berlin, Heidelberg (2008), https://link.springer.com/chapter/10.1007/978-3-540-92698-6_9
15. Triboulot, C., Trouilhet, S., Arcangeli, J.P., Robert, F.: Opportunistic software composition: benefits and requirements. In: Lorenz, P., Maciaszek, L.A. (eds.) Int. Conf. on Software Engineering and Applications (ICSOFT-EA). pp. 426–431. INSTICC (Jul 2015), http://oatao.univ-toulouse.fr/15305/
16. Warmer, J., Kleppe, A.: The Object Constraint Language: Getting Your Models Ready for MDA. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2 edn. (2003), https://dl.acm.org/citation.cfm?id=861416