Opportunistic Composition of Human-Computer Interactions in Ambient Spaces

Augustin Degas, Jean-Paul Arcangeli, Sylvie Trouilhet
IRIT, University of Toulouse, UPS, France
Firstname.Lastname@irit.fr

Gaëlle Calvary, Joëlle Coutaz
Univ. Grenoble Alpes, LIG, F-38000 Grenoble, France
CNRS, LIG, F-38000 Grenoble, France
Firstname.Lastname@imag.fr

Stéphane Lavirotte, Jean-Yves Tigli
University Nice Sophia Antipolis
CNRS (UMR 7271), I3S, Sophia Antipolis, France
Firstname.Lastname@unice.fr

***Abstract*—We propose an approach based on Adaptive Multi-Agent Systems, using the principles of Meta-User Interfaces and Opportunism in order to solve Human-Computer Interaction Composition in Ambient interactive spaces. The idea of this approach is to see every component as an agent able to interact with other components to compose autonomously in order to opportunistically suggest to users smart compositions of his interactive ambient environment. We present the notions of component, composition, and human-computer interaction composition. We chose mainly two aspects of the composition of human-computer interaction which are the controllability and finality of the composition. Finally, we illustrate our approach with use cases taken from the neoCampus project.**

Keywords—ubiquitous computing; ambient interactive spaces; human-computer interaction; opportunistic component composition; meta-user interface; adaptive multi-agent system theory

I. Introduction

Ambient intelligence aims at offering an “intelligent” space in everyday life to access numeric information and services by giving each user suitable, natural and user-friendly way to interact with their own interactive environment. By “interactive environment”, we mean all devices, every way of interacting with them, but also every application (*i.e.* software) available to the user. The origin of the concept of “Ambient intelligence” is from Mark Weiser's idea called “Ubiquitous Computing” [22]. In a few words, it is the simple idea of computer technology so profoundly integrated into our daily life that we don't pay attention to it anymore. To picture this, he used the example of literature which is everywhere in our daily life (from street signs to newspapers, books and more) and which we use every day. An important point in this example shows that the limit between the two worlds, that of literature and the human world, is blurred, which means literature is naturally part of our life. The other point is the important number of occurrences of objects that belong to the world of literature, (*i.e.* written words) in our daily life.

Ambient intelligence can be directly applied to Smart Cities in which Cities around the world are becoming connected cities and the use of Information and Communication Technologies is growing every day. Nowadays, interactive systems are everywhere, new sort of devices and more devices are meant to be available in the interactive environment of every user. Devices are meant to appear and disappear in the interactive environment, because the user is moving or is acting on this interactive environment, because other users are moving or are acting on this interactive environment, or because the interactive environment is changing on his own (for example because the batteries of the device are low).

In this constantly changing environment, human must be maintained in the loop, be able to interact with the ambient and interactive environment. This means that Human-Computer Interaction (HCI) must evolve dynamically with this changing environment (the ability of HCI to evolve with the environment is called plasticity [18]). In other words, HCI must be a smart assembly of what is currently available in the interactive and ambient environment, which means we need to be able to compose HCI from what is available, from any available component, but this also means that this composition must evolve with new availabilities and the disappearances of old ones.

In this particular domain of HCI, we believe that the users are not able to compose their interactive environment every minute to deal with each change, and that the system could assist them by suggesting a part or an entire composition from what is available in the environment.

This paper introduces the HCI problem in the context of ambient environment and presents a work in progress approach to tackle this problem by assisting the user.

Section II introduces what a component, a composition, and a HCI composition are. Section III is oriented around two aspects of the Composition of HCI that we chose because they are particularly related to composition of HCI in ambient environment. Each of the two aspects is firstly introduced, then we look how the different approaches have dealt with them. The first aspect is the Controllability of the Composition, *i.e.* the degree of control a user has on it. The second aspect is the Finality of the composition, the goal aimed at when using the composition. Section IV introduces our response proposition (which is the start of our work in progress) to the two aspects presented in section III. This proposition is based on Meta-User Interface, Opportunistic Composition and Adaptive Multi-Agent Systems. The last section presents some Use Cases to illustrate the problem, our work in progress approach and some points that need to be taken into account in the future.

II. Component, Composition and Human computer Interaction

In this section we present the notion of component and composition in the general context of computer science, then in the context of HCI.

A. *Component, Composition*

To explain the notion of component, we can start by looking at the etymology of the noun: Component means "put together" (*componere*, from *com-* 'together' and *ponere-* 'put'), and with this we basically have an idea of what a component is in computer science and in HCI: an entity that we can assemble, put together, to create other entities.

Component and composition in computer science are a software conception approach that wants to be modular, reliable, reusable, and wants to decrease interdependences between modules. This approach was mostly used in order to decrease development cost and time to market. A widely quoted definition of a software component is that of Szyperski [17]:

«A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties.»

In a way, the software component concept is the evolution of the object concept in which the required interface (*i.e.* the set of required services – dependencies on other components) is exposed at the same level as the provided interface. Components are black boxes, software bricks, we can assemble through these interfaces (a required interface can match with an appropriate provided interface), and thus create new components (which means final applications that result from this kind of assemblage are also components). This assemblage is called "composition". Fig.1. shows an example of composition (match of required and provided interfaces) [19]. This example is constituted of a set of component embedded on a computer, and three other components each embedded on their own device (in a way, they are drivers) all present in a lab. A component "Experiment leader" (embedded on the computer) provides its controls to the component "Robot Arm" (a driver). This experiment leader is an interface running on the computer, allowing a user to perform a set of experiments. The Robot Arm provides some electrical signals which are used by an oscilloscope ("Oscillo 3.0"), and the results of the analyze of the current signals by the oscilloscope are stocked in a component "Stock" (embedded on the computer). The set of analyzed signals are finally analyzed as a whole by the "Analysis" component (embedded on the computer) which use all the data in the component "Stock".

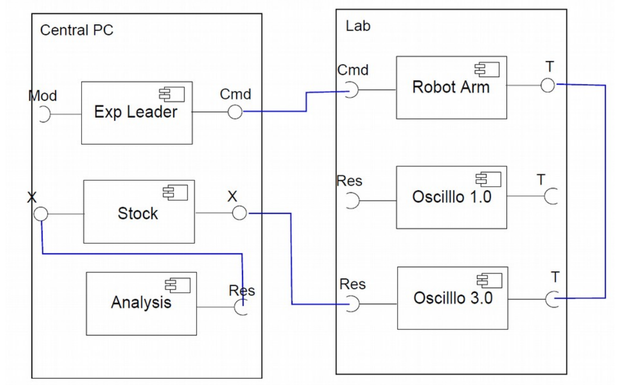

Fig. 1. An example of composition [19]

B. *Composition of HCI*

Even if the notion of component has been introduced almost fifty years ago by Douglas McIllroy during the first International Conference on Software Engineering [15], most research has been done in the last twenty years [21], and this research "remains general, and has still not been applied to HCI" [2].

In the particular case of HCI, we will from now on separate components into two categories, functional components and user interaction components, based on the functional decomposition Functional Core (FC) – User Interface (UI). As we will see later, this distinction makes sense. For now, lets just give some examples:

- UI components can be numeric UI components, like widgets (button, slider, track-bar …), a part of a numeric UI (a set of sliders and a picture box), or an entire numeric UI (*i.e.* the interface of an application). In the same order of ideas, UI components can be physical, like a physical button, a part of a physical UI (like the numeric pad of a keyboard), or an entire physical UI (like a keyboard). In Fig.1. "Exp leader" is a UI component.
- FC components are the functions manipulating the data model. It can be an action triggered by a user's action on the UI (like pushing a button) or simply a function not shown by any mean to the user (like an eye-tracker stocking the movements of the point of gaze in a hidden database). In Fig.1. "Stock" is a FC component.

The following section presents both aspects of Composition of HCI in ambient environment that we want to discuss because it seems particular to this problem: the Controllability and the Finality of the Composition.

III. Controllability and Finality of the Composition

A. *Controllability of the Composition*

By the controllability of the composition, we mean that the composition can be done with various degrees of control from the user, or the designer. In the different approaches of Composition of HCI we saw different degrees of control balancing varying between total automation to total control from the user/designer. As stated, the system can lead the composition, such as *Compose* [10] for which the initiative of the composition comes from the user that expresses a need, but the composition is entirely done by the system. However the composing system can manage slightly less things, e.g. *Task Tree Merge* [14] and *Alias* [12] in which the user/designer decides what will be used for the composition (in these particular cases, the system processes fusion of applications), so he has the initiative of the composition, but the system performs the composition, and only lets the user deal with conflicts s/he doesn't manage. On the other hand, the composition can be entirely managed by the designer as in *ComposiXML* [13], or by the user as in *On-the-fly-services* [23], from the beginning to the end. But the composition system can be between these two extremes and instead of composing almost entirely automatically, or letting the user assume entirely the composition, the composition system can be present at each step of the composition, by extending each user action (*i.e.* to get the idea in another context consider the auto-completion that helps the user to finish typing). An example of such hybridization is *ONTOCOMPO* [1], which helps the designer who chooses a component to keep or suppress from an application by managing the dependencies of the said component, assisting the designer in their task.

In these different approaches, the first point to note is that the composition system tends to exclude the user from the composition process, or on the contrary it tends to leave them do the entire process. The second point, is that the applications are created from what is available on the workstation and not from what is available in the interactive ambient environment. This is contradictory to the idea of ambient environment: all devices must be able to interact with each other, thus a composition cannot composed of known entities all the time, nor restraint to a single workstation. Furthermore, the system must take into account the dynamic of the ambient environment, but we will talk more about it in the following subsection.

B. Finality of the Composition

Concerning our findings on HCI composition, there are two different goals when using composition of HCI. The first is based more on the reuse of component in order to facilitate and accelerate the conception of applications. The objective in this case is to fuse existing applications or parts of applications (*i.e.* to make from two different applications a single one containing both characteristics), or it is to compose static applications (*i.e.* applications not meant to change with the context of use *i.e.* the triplet user, platform, environment [3]) from components. It is the case of *ComposiXML, Task Tree Merge*, *ONTOCOMPO* and *Alias*. This kind of composition is related to the first use of component discussed in section II, which is to decrease development cost and time to market.

The second goal is to use composition to make adaptable applications, more precisely application that can adapt dynamically to the context of use and user task. It can be a fission, in other words a distribution of an interface across different available devices, or the development of applications by the end-user (*On-the-fly service composition*, *SOAUI* [20]), or the automatic development of applications to fulfill the task of a user (*Compose*).

Our aim is to make interfaces capable of evolving with the environment, called plastic interfaces. In an ambient environment, it is necessary to use whatever is available, but also to adapt to the evolution of the availability. In the context of ambient intelligence, component are meant to appear and disappear. It is better to use what is available in the environment than being unable to function because what we need is not available.

Another point concerning the finality of the composition is that in the different approaches, there is a known need when composing: in on case the designer or the user is controlling the composition entirely (*ComposiXML*, *ONTOCOMPO, On-the-fly service composition*, *SOAUI*), thus they are following a need. In another case a system is composing automatically (*Task Tree Merge, Alias*) from what a user has decided to fuse, which also means he is following a need (in this case it is to stop redundancy of action or information). In the last case a system is composing from a need expressed by a user (*Compose*). Our position regarding this point is that contrary to designers that may be able to collect the requirements, users may have trouble formulating their needs, or to do it sufficiently precisely to be used in the construction on an HCI, thus we want to help them in the process.

C. Synthesis

From the two aspects of the composition of HCI we described in this section, we have shown three needs, requirements that the composition system must fulfill when composing HCI in ambient environment. The first need emerges from the aspect of controllability of the composition: it is required to keep the users in the loop and enable them to both observe and control their interactive ambient environment. The second need comes from the aspects of controllability and finality of the composition: we equally want a system aware of the context of use (as a reminder, the triplet user, platform and environment), but also a system able to evolve with this context of use. Last need emerges from aspect of finality of the composition: we want to help the user with the composition of their environment.

The need to maintain the user in the loop and the need to help them in the process may seem contradictory at first, but we need to keep in mind that exactly as in the case of controllability, one does not exclude the other, in other words we want a middle ground between these two needs.

In the following section, we will address answers to all these needs, which we want to use in our work in progress approach. As we will describe in the following section, these answers are Meta-User Interface, Opportunism and Adaptative Multi-Agent System.

IV. A proposition of Composition of HCI in ambient environment

In this section, we begin to describe in the following sub section general principles of our work in progress approach, Meta-User Interface and Opportunism. Then in another subsection, we discuss briefly of an Adaptive Multi-Agent System with which we want to deal with these principles.

A. Meta-User Interface and Opportunism

1) Meta-User Interface

Meta-User Interface is an answer to the need to enable the user to both observe and control their interactive ambient environment. Let us present the concept as it was introduced in [6]. The starting point of the Meta-UI concept is the same as the one we used earlier: in ambient environment, "users are not limited to the system and applications of a single computer […] [. U]sers, services, and resources discover other users, services and resources, and integrate them into an ambient interactive space" [6]. This interactive environment is meant to evolve dynamically and is not limited to a single workstation, thus in this ambient interactive environment, usual solutions to control the interactive environment are not appropriate anymore (like shells for a single station), and it is required to keep the user in the loop, in other words it is required to find a way to enable them to control their interactive ambient environment. As defined in [6], "The concept of Meta-User Interface, as the set of functions (along with their user interfaces) that are necessary and sufficient to control and evaluate the state of interactive ambient spaces". To resume, the concept gathers which entities can be present in an interactive environment, how we can manipulate them, and what we can do with them.

Meta-UI is necessary in this ambient context, as it fulfills the need to keep users in the loop and enables them to both observe and control their interactive ambient environment. In our work in progress approach, we want to include such concept because it could enable the user to observe the interactive ambient environment (*i.e.* every available component), but in our approach the control (*i.e.* the control of the composition) would not be total, as we said earlier, but a middle ground between automation and total control.

2) Opportunism

In traditional software development, software are usually developed to respond to explicit and pre-established needs and stakeholder's requirements. Nevertheless, the evolution of the context of use is an important and difficult challenge for developers, because the needs and stakeholder's requirements are evolving with. The first response to this challenge is to accelerate the time to market of applications, and as you may have noticed, it was the major using of components. The second is to make applications that evolve with the context of use. To evolve with the context of use, there are two solutions, the first one is to make applications able to evolve by themselves with the context of use. The problem of this approach is that in ambient environment, it is impossible to predict every possible situation. From this final assessment, the concept of opportunistic composition has emerged: if it is impossible to describe every possible situation and to adapt applications to all these possible situations, let us make applications from every situation, in other words let us compose applications from what is available in the environment, and recompose when this environment evolves. This idea is reversing the traditional software development process. While traditional software development process starts with requirement analysis, the opportunist approach is triggered with what is available in the environment (*i.e.* components) to create application (*i.e.* a composition), because there is an opportunity, and then consider the interest of such application. This bottom-up approach (by opposition to the traditional top-down approach) makes application that emerge from the environment and evolve afterward by dynamic (re)composition based on new opportunities. This bottom-up approach has been used in [7]. In this approach the components are included in a container in a Service Lightweight Component Architecture (SLCA). A component is a software component or a proxy to services. An application is described with a set of rules which may be dynamically implemented at run-time. These rules are managed by a *weaver of Aspect of Assembly*. Their approach is opportunistic, by selecting the most appropriate set of aspect of assembly according to the context. In our approach, we would like to be independent of such rules or models, in order to be able to react to unpredicted situations for which no rules have been provided for. Such principle fulfills the need of a system aware of the context of use and able to evolve with this context of use.

B. Adaptative Multi-Agent System for Composition

Our approach is based on Adaptative Multi-Agent Systems (AMAS) [4], that is what we introduce here. In computer science, Ant Colony Algorithm (ACO) [5], is a well known algorithm used originally to find the shortest path on graphs. This ACO is based on what ants are doing in real life to bring more food to the colony: every worker ant explores the environment, and when it finds a food source, it takes what it can carry and leaves pheromones on its return path to the colony. Every worker ant is also exploring the environment, and when it finds pheromone trace it may follow it to find a food source, and on its return path it will also leave pheromones. Nevertheless every ant will not take each time the path of the other ant, and so leave pheromones on another path. Pheromones evaporate with time, and shortest paths being used more than longer paths on average, so at the end of multiple ant passages, the shortest path should emerge: almost every ant that will go to this food source will use the shortest path.

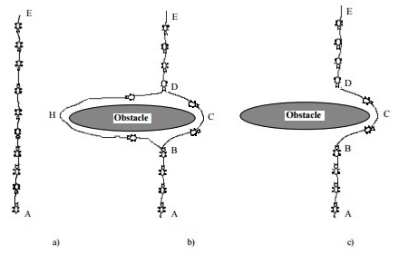

Fig. 2. Discovery of the new path by ant after an obstacle has been placed on their previous path.[5]

This example illustrates the paradigm of Adaptative Multi-Agent Sytem. An agent is an autonomous computer program. Every worker ant can be seen as an autonomous entity, that evolves in an environment composed of other ants and food sources. Every worker ant evolves in its environment, acts on the environment by leaving pheromones, and accomplishes its goal by bringing food to the colony. Every worker ant only sees its surroundings, in other words every entity has a local view of its environment. Basically we have here the notion of what an agent is: an autonomous program that has a local goals, perceives its environment locally, decides of actions accordingly to its goals, and acts. A set of agents can be considered as a Multi Agent System (MAS).

For the adaptive element, we need to go back to the pheromone part: every ant has a cooperative attitude, it leaves pheromones on its return path in order to help other ants to find food sources. It may help itself, not necessary, but it will surely help other ants. In some cases, an agent may be non-cooperative, which means it may bother other agents in their task, or it cannot fulfill its goal and thus cannot help the group at all. The AMAS theory identifies seven generic non cooperative situations [4]. In such situations, the agent will change its nominal behavior, in other words its usual behavior, and behave differently to reach a cooperative situation. If we go back to the example of the ant colony, the entrance of the colony may be blocked, and thus ants that want to enter are unable to do so: they are unable to accomplish their goal, and other ants passing by may help them to unblock the entrance even if they don't want to enter themselves.

Components are independent units that can be assembled through their required and provided interfaces, in other words components can be organized in order to obtain new components. Each component can be seen as an autonomous program, in other words as an agent. This idea may seem odd, nevertheless we can already justify the autonomous part by reconsidering ambient environment: every entity may be on its own, and so must be autonomous. Components are naturally decentralized entities, thus using AMAS to represent them and to make them organize (compose) autonomously is appropriate. These agents may interact with each other, they may communicate with their surrounding to require and to provide what they require and provide with their interfaces in order to assemble and compose autonomously. Such approach has been made in the area of functional programing [11], and we want to have such approach in ambient interactive environment. It may have been noticed that with this AMAS approach we are dealing with the opportunism principle: components are composing one with another autonomously and applications may emerge from these compositions.
We want to enable the user to observe his interactive ambient environment (*i.e.* every available component), but the control (*i.e.* the control of the composition) would not be total, but a middle ground between automation and total control. Such position is based on the idea that the main default of most automatic approaches is to let the user out of the loop [6], but that we also believe that every user is not able to compose his interactive environment every minute to deal with every changes. In our work in progress approach, we want to help the user in the composition, and to provide such help we want to suggest composition to the user. These suggestions can be small steps, *i.e.* part of a final composition, coupling two components for example, or it can be suggestions of final applications. Applications may emerge from autonomous behavior of components, such suggestions would result from the composition of numerous components. We want to enable the user to choose trough an interface (a Meta-UI) what he wants to have, to choose among different smart suggestions. Such suggestions might be numerous in order to suggest a lot of different possibilities, or can be restraint but more appropriate in the context because the system is also learning. We are dealing here with the second need (section III.C), the need to help the user to compose in the interactive ambient environment. Ideally, explanations about the compositions would be given to explain what are these compositions and why this compositions have been chosen among others. In this approach, user goals are not directly taken into account: the context of the system take into account previous behavior of the user and his/her habits, but the user could also require compositions based around a component (in the following use case for example he may have solicit a composition with the camera driver).

In the following subsection we show some examples of how would look like this kind of approach. The use case represents some components that may be able to compose in a pedagogic context in university with students.

V. Use Case

In this section we will show a use case relating HCI Composition in an ambient interactive environment. The context of this use case is a class taking place in a classroom, but also retransmitted in another class and possibly in other places (like in some students' home). We will first show possible compositions, showing a bit our work in progress approach, and then from these possible compositions we will illustrate some earlier points, and discuss of some new points.

In this example, numerous interactive components may be present. For example the professor may have multiple cameras pointed at him or at the board, a smart board, a tablet, a personal computer and a microphone. Students on

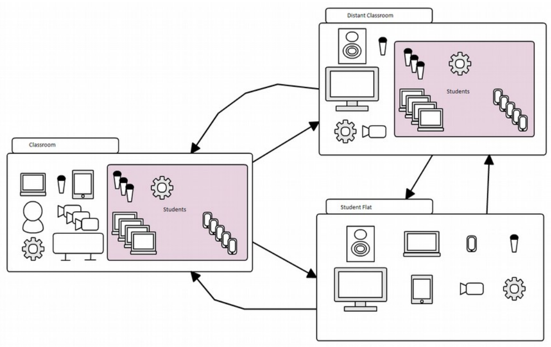

Fig. 3. A class taking place in a classroom, retransmitted in two other locations down right a student's room with a student.

their side may also have microphones, personal computer and smart phones, but also speakers and screens. Fig.3 represents three locations, distant (three rectangles), but interconnected (arrows symbolizes this interconnection): on the left a classroom with the professor and some students, up right another classroom only with students, and in this ambient environment, a lot of examples could have been taken to explain what we see in our opportunistic composition of HCI, but as you may imagine the number of possible compositions is quite large, so to simplify we only use a tiny example with a camera and a tablet. The choice of these two elements may also be seen as suggestions of final composition pre-selected by the system among other possibilities. Scalability is not an objective in this paper, the goal here is to make the best with what is available in the environment around the user (or what is considered close). The problem of scalability stays open for now.

In our example, the driver of the camera is a component that requires the management of three controls: its angles of rotations (ɵ for the yaw angle, ȹ for the pitch angle-see camera driver draw of Fig.4) which are floats, and its zoom which is also a float. This same driver provides a video stream (what the camera is filming currently). This component will autonomously ask in its environment if other components provides some means to manage its controls and/or requires a video stream.

The components from the tablet being the only ones in our example, they will also be the only ones to answer to the driver component. For the example, let us simplify a bit more and say that the available components of the tablet are only sliders and a component able to display a video stream. And to simplify even more we are going to say that the sliders have the same appearance, except for the orientation which can be vertical or horizontal. In this very simplified use case we already have 54 possible compositions (3 required controls with 3 possible states *vertical, horizontal* or *none*, and a provided stream video that maybe be matched or not), or 8 if we consider that every required interface of the camera will have a match. We will only consider 4 examples of resulting composition(see Fig.4), that may be seen as suggestions proposed to the user on his Meta-UI:

- Two horizontal sliders can manage ȹ and the zoom and one vertical slider can manage ɵ. (1)
- Three horizontal sliders can manage ɵ, ȹ and the zoom (2)
- Two horizontal sliders can manage ɵ and the zoom and one vertical slider can manage ȹ. (3)
- One horizontal slider can manage ɵ and two vertical sliders can manage ȹ and the zoom. (4)

And in all these suggestions, a component of the tablet may display the video.

In these different suggestions one is particularly bad for the user, one is more acceptable but suboptimal and two are acceptable. By acceptable, we mean here that the suggestion is ergonomic, helps the user to know how the application works, does not induce them in error and try to be easy to use.

The suggestion (1) is particularly bad because it induces the user in error: while one would expect the "logical" choice with ȹ managed with a vertical slider because the angle is in a vertical plan, the vertical slider manages an angle in the horizontal plan (ɵ). The second one (2) is more acceptable because instead of inducing the user in error, it just helps them to determine what each slider does. The two last suggestions (3)(4) are acceptable because they don't induce the user in error and they try to help the user to determine which slider manages what. We may say that the last example (4) is better because it respects the habits of users (zoom is usually managed with a vertical UI component).

What we are showing here is that in HCI, we must add other criterion than these already defined in [19]: decentralization, dynamic adaptation, combinatorial optimization, re-composition, learning and context awareness, utility of the result, and silence of user needs. And by this we are implying that in the composition, the simple matching of interfaces (a required interface with an appropriate provided interface) is not sufficient, and thus the user must be an active part of the composition. For example in our case, from a functional point of view, both sliders have the same quality, but from a user point of view, use the vertical slider is more appropriate for ȹ, in this situation, the vertical slider is more qualified. Contrary to criterion we can quantify and to which we can associate a value, criterion that may distinguish two elements, some criterion are dependent on the appreciation from a user point of view. For example, we can dissociate two screens of the same size with their resolution. We may say that choosing a vertical slider for ȹ is logical because in both we find the idea of verticality, but the choice of a vertical slider for the zoom is based on users habit and not on semantics. It is because the user is used to this representation that we continue to use it.

Lets now say we had the possibility to use sliders, but also other numeric objects. For example we could associate the zoom to a pinch/spread gesture, ȹ to vertical slide and ɵ to horizontal slide. If we allow the designer to use tactical gestures, they will use them, but why? Is it by habit or

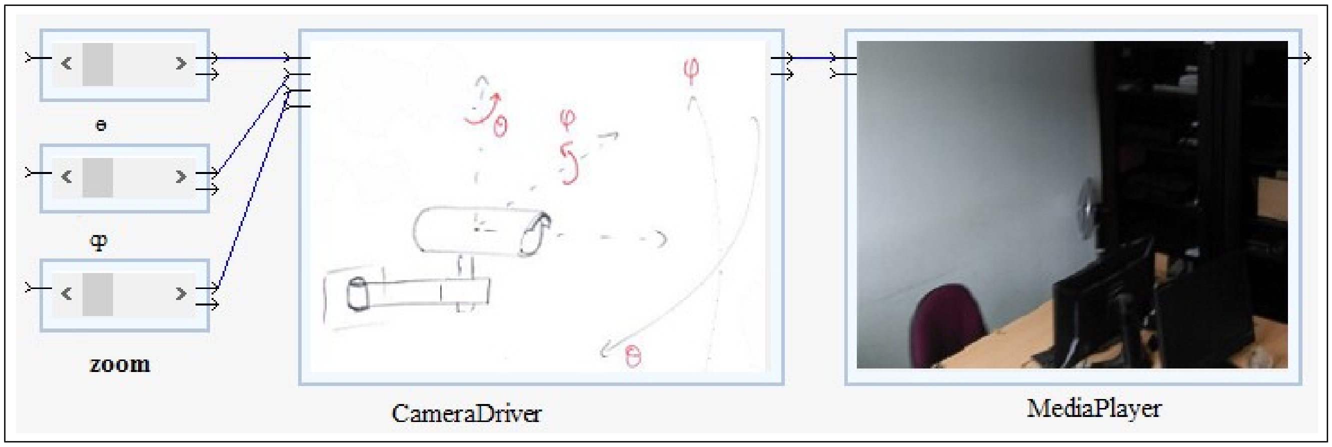

Fig. 4. Illustration of Composition (1): - > are required interfaces and > - are provided interfaces

because it uses less space on small screens? If it is by habit why didn't we keep old habits and so sliders from the computer, and if it was to use less space, why didn't we use the tablet's gyroscopes? Beside the fact that not every tablet has efficient enough gyroscopes, there is also a practical question: forcing the user to lean the tablet will degrade their vision on the screen. Once again we find criterion user oriented, designers wanted to save some space on these small screens and they chose what was most appropriate for both sides (human and computer). In a way, it is understandable given the fact that HCI is the interface between humans and machines.

In the HCI composition, the simple matching of interfaces is not sufficient, and thus the user must be an active actor of the composition. Nevertheless, as you may also have noticed, we simplified a lot the example, and even with these conditions, a lot of suggestions can be made, and some criterion may be taken into account by machine learning: if a user whom we suggest different choices of HCI, chooses the same option every time, the system can learn it. These learned preferences may be used to restraint the number of suggestions, providing a better help to the user to control their interactive and ambient environment and contribute to the requirement of combinatorial optimization.

We have also briefly introduced a notion we want to use in our composition of interfaces, which is the abstraction of UI component by what they can manage, for example a slider can manage a float. This idea is not really new, we can find it in [9] for example, which is more than thirty years old, nevertheless it is an interesting idea for what we want to do because it opens a lot of possibilities: a new component that is able to manage a float or an integer is more easily included in new composition than a component providing a "slider" or a "track bar". It also opens some unexpected possibilities. For example, we previously used sliders, these sliders can be abstracted as able to manage a one dimensional float in a range, but are also able to manage other dimensional variables, like different boolean (if the area of the slider is clicked or not, if the mouse is in the area of the slider…). This abstraction can also be used to represent non conventional user interfaces, we can for example say that a door has the ability to manage a one dimensional float in function to the angle that the door makes with the wall (the opening angle of the door) but also boolean (if the door is opened or not …). Rolling shutter can be represented the same way, for example the state of the shutter can be seen as a one dimensional float (for example a half closed shutter corresponds to a 0,5 on a [0;1] range). And if we allow such abstractions, someone that doesn't have any other way to interact with the camera, like a student with no smart-phone nor personal computer, may be able to manage θ with the door, and φ with the rolling shutter.

VI. Conclusion

From ambient interactive environment emerge different needs. Such environment by nature dynamic and thus applications based on this interactive environment must evolve with it. Nonetheless such evolution cannot exclude the user from the loop, but to include the user in the process we must make it accessible and the process should not require too much time from the user. Based on the natural independence of every entity in such environment, the use of component and composition is appropriate. However classical component based software engineering cannot be used directly (user's requirements are evolving, the environment is evolving, and user must be implicated in the composition process…), and thus composition must be adapted to the problem.

To the adaptive need we addressed a proposition based on opportunism: in this approach application emerges from the composition of what is available in the environment, and thus will evolve with it dynamically. For the necessity to implicate the user, we proposed to enable the composition system to make suggestions to the user. These propositions could be small steps, part of a final composition when the user is able to express his requirements, or final compositions if he is unable to express it.

On an experimental point of view, our work is still in progress. We want to use in our work AMAS and re-enforcement learning in order to compose opportunistically. A first composition engine based on AMAS technology has already been implemented, we want to make another version more evolved, and we are currently working with WComp [8] and UPnP devices. On a AMAS architecture, which will be the composition system, we have a three level architecture: service agent level, component agent level and type agent level. For every required or provided interfaces there is a service agent associated. Component agents will coordinate the behavior of its services agents, for example the camera driver is a component agent and its three controls (required interfaces) and its video stream (provided interface) will be managed each by a service agent (so there is four service agents). Type agents manage component agents of its type, and will gather information and learn from it in order to suggest better composition to the user. The behavior of these different agents is currently being discussed, for service agents, the life cycle (perception, decision, action) and its non cooperative situations are identified. These service agents will be the one using the abstraction by type (int, float..) (section V). However, such alignment is not sufficient in real conditions, for example a slider able to handle the zoom (float) of the camera (Fig.4) should know the range of the zoom (0 to 80), and the initial zoom, and the encoding can be different from the camera to the slider. We find equivalent concerns in WoT (Web of Things) [16], and their solutions may be adapted to our problem.

Acknowledgment

This work is supported by the neoCampus project, a project that gather ten laboratories of the university Paul Sabatier (Toulouse) and see campuses as small connected cities. This work is also a collaboration between three laboratories, I3S (Nice), LIG (Grenoble) and IRIT (Toulouse).

References

[1] Brel, C., Gonin, P. R., Giboin, A., Riveill, M., & Dery, A. M. (2014, September). Reusing and Combining UI, Task and Software Component Models to Compose New Applications. In *Proceedings of the 28th International BCS Human Computer Interaction Conference on HCI 2014-Sand, Sea and Sky-Holiday HCI*(pp. 1-10). BCS.

[2] Calvary, G., Dery-Pinna, A. M., Occello, A., Renevier, P., & Gabillon, Y. (2013). Composition of User Interfaces. *Computer Science and Ambient Intelligence*, 203-224.

[3] Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., & Vanderdonckt, J. (2003). A unifying reference framework for multi-target user interfaces. *Interacting with computers*, *15*(3), 289-308.

[4] Capera, D., Georgé, J. P., Gleizes, M. P., & Glize, P. (2003, June). The AMAS theory for complex problem solving based on self-organizing cooperative agents. In *Enabling Technologies: Infrastructure for Collaborative Enterprises, 2003. WET ICE 2003. Proceedings. Twelfth IEEE International Workshops on* (pp. 383-388). IEEE.

[5] Colorni, A., Dorigo, M., & Maniezzo, V. (1991, December). Distributed optimization by ant colonies. In *Proceedings of the first European conference on artificial life* (Vol. 142, pp. 134-142).

[6] Coutaz, J. (2006). Meta-user interfaces for ambient spaces. In *Task Models and Diagrams for Users Interface Design* (pp. 1-15). Springer Berlin Heidelberg.

[7] Ferry, N., Hourdin, V., Lavirotte, S., Rey, G., Tigli, J. Y., & Riveill, M. (2010). Models at runtime: service for device composition and adaptation.

[8] Ferry, N., Hourdin, V., Lavirotte, S., Rey, G., & Tigli, J. Y. (2013). Wcomp, middleware for ubiquitous computing and system focused adaptation.*Computer Science and Ambient Intelligence*, 89-120.

[9] Foley, J. D., & Wallace, V. L. (1974). The art of natural graphic man—Machine conversation. *Proceedings of the IEEE*, *62*(4), 462-471.

[10] Gabillon, Y., Petit, M., Calvary, G., & Fiorino, H. (2011, February). Automated planning for user interface composition. In *IUI 2011-International Conference on Intelligent User Interfaces* (p. 5p).

[11] Georgé, J. P., & Gleizes, M. P. (2005). Experiments in emergent programming using self-organizing multi-agent systems. In *Multi-Agent Systems and Applications IV* (pp. 450-459). Springer Berlin Heidelberg.

[12] Joffroy, C., Caramel, B., Dery-Pinna, A. M., & Riveill, M. (2011, June). When the functional composition drives the user interfaces composition: process and formalization. In *Proceedings of the 3rd ACM SIGCHI symposium on Engineering interactive computing systems* (pp. 207-216). ACM.

[13] Lepreux, S., Vanderdonckt, J., & Michotte, B. (2006). Visual design of user interfaces by (de) composition. In *Interactive Systems. Design, Specification, and Verification* (pp. 157-170). Springer Berlin Heidelberg.

[14] Lewandowski, A., Lepreux, S., & Bourguin, G. (2007). Tasks models merging for high-level component composition. In *Human-Computer Interaction. Interaction Design and Usability* (pp. 1129-1138). Springer Berlin Heidelberg.

[15] McIlroy, M. D., Buxton, J. M., Naur, P., & Randell, B.(1968, October). Mass-produced software components. In *Proceedings of the 1st International Conference on Software Engineering, Garmisch Pattenkirchen, Germany* (pp. 88-98). sn.

[16] Rocher, G., Tigli, J. Y., Lavirotte, S., & Daikhi, R. (2015, October). Run-time knowledge model enrichment in SWoT: A step toward ambient services selection relevancy. In *Internet of Things (IOT), 2015 5th International Conference on the* (pp. 62-69). IEEE.

[17] Szyperski, C., Bosch, J., & Weck, W. (1999, June). Component-oriented programming. In *Object-oriented technology ecoop'99 workshop reader* (pp. 184-192). Springer Berlin Heidelberg.

[18] Thevenin, D., & Coutaz, J. (1999, August). Plasticity of user interfaces: Framework and research agenda. In *Proceedings of INTERACT* (Vol. 99, pp. 110-117).

[19] Triboulot, C., Trouilhet, S., Arcangeli, J-P., & Robert, F(2015). *Opportunistic software composition: benefits and requirements*. In *Int. Conf. on Software Engineering and Applications (ICSOFT-EA)*, INSTICC, p. 426-431.

[20] Tsai, W. T., Huang, Q., Elston, J., & Chen, Y. (2008, October). Service-oriented user interface modeling and composition. In *e-Business Engineering, 2008. ICEBE'08. IEEE International Conference on* (pp. 21-28). IEEE.

[21] Vale, T., Crnkovic, I., de Almeida, E. S., Neto, P. A. D. M. S., Cavalcanti, Y. C., & de Lemos Meira, S. R. (2016). Twenty-eight years of component-based software engineering. *Journal of Systems and Software*, *111*, 128-148.

[22] Weiser, M. (1991). The computer for the 21st century. *Scientific american*,*265*(3), 94-104.

[23] Zhao, Q., Huang, G., Huang, J., Liu, X., & Mei, H. (2008, December). A web-based mashup environment for on-the-fly service composition. In *Service-Oriented System Engineering, 2008. SOSE'08. IEEE International Symposium on* (pp. 32-37). IEEE.