Input interactions and context component based modelisations: differences and similarities.

Diane Lingrand and Michel Riveill
RAINBOW team
I3S UMR 6070 - Univ. Nice - Sophia Antipolis / CNRS
B.P. 145 - F 06903 Sophia Antipolis Cedex - France
Diane.Lingrand@unice.fr

ABSTRACT

Since several years, ubiquitous computing and pervasive computing has emerged and, in particular, context-aware computing. Using mobile devices, the context is perpetually evolving, more than with standard workstation. In such environment, software must modify its behavior dynamically. An emerging way of programming an adaptive software is component programming.

The focus of this paper is to review existing component approaches for input devices and for context especially those that consider component modeling. We aim at determining the similarities and differences between context and input devices in order to propose further a common component model architecture that will help building such intelligent applications.

Categories and Subject Descriptors

H.5.2 [**Information Interfaces and Presentation**]: User-centered design

Keywords

Context, human computer interaction

1. INTRODUCTION

This paper deals with software applications on mobiles devices. One of the major challenges of developing such applications concerns the interaction with these devices[29]. Traditional interaction using mouse and keyboard as input devices are no more adapted: they may be not available (how to type on a keyboard while walking) and interactions with an application may not be the main focus of the user (interacting with a computer is obviously not the main task of a taxi driver while driving).

Adapting the software to the evolving situations, to the task to be performed and to the user could decrease the need of explicit interactions between the user and the application. This is part of ubiquitous computing approach and is context-aware computing aim: building applications that are aware of the context.

In order to build software that can dynamically adapt to a context that is constantly evolving, the component modeling approach is the most appropriate. This approach has emerged since several years, in the continuity of modular coding and objects programming. After focusing on standalone software running on one workstation, we are now in a world of complex software applications that are distributed and that run on different hardware platforms. The complete rewriting of an application is no more thinkable. Portions of code must be reused. Software must dynamically adapt to evolving situations.

However, the notion of component is not clearly defined. We can say that it has inputs, outputs that are computed from the inputs. An interesting property is that the composition of two components is also a component. But composition is also not really clearly defined.

The compositional adaptation is defined in [26] as it "enables software to modify its structure and behavior dynamically in response to changes in its execution environment".

The separation of concerns between business components and technical components has been studied by several authors and is now clearly well established in the component programming community.

However, the software part that is concerned by interactions needs to be modeled by components in order to take into account the plasticity of interfaces. Few works has been done considering this GUI components approach, except [27].

It is also necessary that the context acquisition and context interpretation itself dynamically adapt to evolving sensors, needs or tasks ... which means that the context processing need to be context-aware. For this matter, a component modeling approach for the context seems to be well adapted.

There are two different aspects concerning, context and composition: (i) the composition of context components for context acquisition and context processing and (ii) the adaptation of components to the evolving context. However, the availability of a sensor may also be part of the context: context acquisition is thus depending on the context ! Therefore, we will clarify the notion of context later in the paper.

The focus of this paper is to review existing component approaches for input devices and for context. The question is: what can we learn from context-aware modeling for input devices management and vice versa ? We aim at determining the similarities and differences between context and input devices in order to propose a common component

CAI AVI '06 Venezia

model architecture that we help building such intelligent applications.

We will first review component approach for input devices. As we will see, they are not widely studied. Then, we will focus on context component architectures that have been more widely studied.

2. INPUT DEVICES AND TAXONOMIES

Input devices are devices used by users to enter commands to computers such as mouse and keyboard. There exist a large variety of input devices from keyboards to 3D mouse, speech recognition, handwriting recognition, video analysis and so on.

2.1 Input devices taxonomies

Since several years, several authors have proposed input devices taxonomies in order to classify these devices. Foley et Wallace [15] classify the input devices in four families: grab, localization, button and value. In [16], they propose to associate input devices to actions to be performed.

In other hand, Buxton[5] classify input devices on a bi-dimensional graph depending on the degrees of freedom and measured data (motion, pressure, ...) and the type of touch with the user. Card and colleagues[6] have extended this taxonomy by distinguishing the absolute values from relative values and in separating translation components from rotation components.

Jacob et Sibert [23] began to observe that the missing dimension concerns the cognitive perception. Their taxonomy makes the distinction between integral devices (considering all degrees of freedom as a whole) and separable devices (consider each degree of freedom individually). This classification has been used by [22, 2].

Beside these almost theoretical different taxonomies, we also find experimental comparison based on measures such as the Fitts law [14] or the steering law [1] as for example in [22].

2.2 Mackinlay and colleagues approach

The first work that present input devices closer to the component approach is presented in [25] . They observed that "neither taxonomy deals with the combination of individual input devices into complex input controls." which consist of a premise of component composition.

They represent an input device by a sextuplet composed by M, the manipulation operator (e.g. z-axis rotation for a rotary plot), In the input domain (e.g. values between 0^o and 270^o, S the current state (e.g. θ), R the resolution function, Out the output domain and W works or rules (e.g. what to do if a joystick is released).

The authors propose three composition operators for Input Devices: connection composition, layout composition and merge composition. The connection composition connects the output of a component to the input of another component using a "resolution function" for data transformations. However, the use of the resolution function has been restricted to constant scale factor and clipping. More complex resolution function could be studied in order to have more complex interactions.

The layout composition consist of a space repartition of devices (e.g. horizontally aligned). The merge composition concerns different input devices merged into one (e.g. two 1D sensors merged in a 2D sensor): the merge composition of two input devices is still an input devices.

2.3 Component models

In [28], authors review methods of bricks composition, from composition for kids based on LEGO and LOGO to Electronic Bricks and to Programmable Bricks. One step further, the phidgets[19] (physical widgets) are programmable components representing physical devices. Authors argue that "just as widgets make GUIs easy to develop, so could phidgets make the new generation of physical user interfaces easy to develop". It is easy to build new devices by physical assembly of individual phidgets.

In other hand, the ICON (Input Configurator)[13] approach is a toolbox for easy creating Post-WIMP input devices. They propose a new input model based on input configurations. The ICON input toolkit is composed by three different types of component: *system devices* (input devices) that can be linked to *library devices* (transformations providing higher level informations) and *application devices* that allow users to build interactive applications and to configure their inputs in order to use alternative input device and techniques taking into account the use of impoverished or enriched input. More recently, the WCOMP[8] approach consider input devices as *Java Bean* components. Both approaches allow the programmer to graphically choose in the composition schema the component management he wants for the input devices. These two approaches provide mechanisms to allow the programmer to compose components.

2.4 Conclusion on input devices

The Mackinlay approach is a good basis for an input device component. However, it is not enough expressive for composition. The composition proposed by the Phidgets approach is interesting but not sufficient for devices of higher level. The ICON approach is the closest to what we are looking for because all devices existing today are considered and the component architecture separating the acquisition (devices) from analysis and presentation to the application. However, it is at the user to program the adaptation.

3. CONTEXT-AWARE APPLICATIONS

We will now focus our interest on context-aware applications. These applications use informations from their environment such as it were a kind of input device. This field of research is really younger than the field of input interactions and is subject of a lot of different ideas on components based architecture developed by different teams.

3.1 What is context ?

A definition that is general and well cited by authors in the context-aware computing community is the definition by Dey [11]:"Any information that can be used to characterize the situation of an entity, where an entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and the application themselves".

Several authors [7, 10] have made a classification of the different types of context in several families: **environmental context** such as nearby people or objects, luminosity, temperature, noise or weather, **user context** such as user location, personal interest, preferences, activity, emotion, muscle contraction, blood pressure or handicap, **computer context** such as URL, nearby displays, network bandwidth,

memory available and **time context** such as history of actions, history of locations or current time and date.

These informations have very different natures. Some are easily measurable using sensors and can be compared each others. Some are more difficult to measure such as emotion or weather because of the subjectivity of their nature. They need to be quantified with respect to the usage that will be done by applications. Different levels are also needed (e.g. space coordinate, name of the city or office number). Some other information such as preferences should be entered by users or inferred from other informations on the user.

Therefor, some authors differentiate the **sensed context** from other type of context informations. In [18], authors defined sensed context as referring to that part of context that is accessible via sensors. They define the sensed context as "properties that characterize a phenomenon, are sensed and that are potentially relevant to the tasks supported by an application".

Henricksen [21] consider sensed, static, user-supplied (profiled) and derived information as the most useful. She also mention that since context comes from a wide variety of sources, the quality and persistence of context information is really heterogeneous. These four categories of context present distinct characteristics.

Context information may be provided by sensors that have different quality attributes such as accuracy, robustness, precision. They also could fail. Context information may also be provided by users. In this case, information is likely to be correct but may be not up-to-date if the user neglected to update them. Henricksen [20] classified the properties in four classes: unknown, ambiguous (2 sensors give different responses), imprecise and erroneous. Then, context information is not only the information itself but needs meta-information such as accuracy, dynamic, and also cost function in order to compare different context each other [24].

3.2 What are context-aware applications ?

We need to distinguish parameterized applications, context-aware applications and context-dependent applications. Parameterized applications used informations as parameters without adaptation. Context-dependent applications cannot run without context information but may have adaptation to fluctuation of context (e.g. a GPS on board of a car). Context-aware applications are enhance by the presence of context information but can still run without information (e.g. a context-aware mobile phone will automatically switch the ring off entering a meeting and augment the level of ring in a noisy room in absence of a meeting but will still react as a usual mobile phone in absence of informations). In [12], three type of context-aware functions are defined: presenting information and services, automatically executing a service and attaching context information for later retrieval.

In the following, we focus on the modelisation of context in components. Our need concern the adaptation of the system to the addition of a context component, the deletion of a context component and the replacement of one component by another.

Replacing a localization system using a GPS by an indoor positioning system when the user enter a building induce new problem: the resolution is different, the precision is different and the information we will use may be different. Making measurements [24] will need to have a semantic continuity in case of sensor missing or new sensor connecting.

4. COMPONENT MODELS OF CONTEXT

4.1 The Context Toolkit and further approaches

The **Context Toolkit** (CTK) [12] provides applications with access to context information from their environment using software components that have distinct functions. Context widgets are responsible of the acquisition of context information. Four other components are responsible of context-specific operations: interpreters, aggregators, services and discoverers. Interpreters produce higher-level context information from informations given by context widgets. Aggregators product an easy access to all the context informations related to a given entity. Then, services execute behaviors on the environment related to the context obtained and discoverers allow applications to know the context informations available in order to take advantages of them.

The CTK aims at hiding the complexity of sensors and data acquisition from sensors. The context information is abstracted to higher level in order to separate the acquisition concern to how context information is used. This toolkit is make from components that are reusable and customizable.

However, it is not really clear how to merge informations from different sensors, how to maintain different levels of abstractions and how to use informations about sensors such as resolution, accuracy that are accessible.

The **Solar System** [7] is a graph based approach following [12] that proposes four types of operators: Transformer, Filter, Merger and Aggregator that can be combined. The differentiation of the operators is interesting but no information is given on how to compose these operators.

4.2 The Context-Handling and The Contextor

The **Context-Handling Component**[18] is responsible of transformations on context data using three flows of informations: data, meta-data (informations on data) and control. Each component can connect its outputs to the inputs of another component.

The **Contextor**[10] is computational model for context that is an enhancement of the context-handling component. Contrarily to the context-handling component, the contextor uses the data and meta-data in the same channel in order to insure synchronism. It is a "software abstraction that models a relation between variables of an Observed System Context". The authors present different contextor types: the elementary contextor that does not have data and meta-data entries because this contextor represents sensors and the non elementary contextors: history contextor, threshold contextor, translation contextor, fusion contextor and abstraction contextor.

The composition of contextors is defined as "a kind of history function that involves the emergence of new relations and peripheral state variables as well as the destruction of old ones" but without explained how it is performed. They explicit two types of composition, the "Data Channels Connection" that is dynamically done at the execution and the "Encapsulation" that is decided by the software writer.

4.3 Other models

In the **Strathclyde Context Infrastructure** (SCI)[17], authors imagine a context entity as a software component. They consider the problem of discovering and composing ap-

propriate context entities to be a special case of the more general problem of discovering and composing components in software engineering. They also abstract the concrete context entities and configurations to basic context elements and context operators, in order to build higher-level contexts. They address several open issues such as the dynamic composition of context entities, control over the quality and structure of context composition, and adaptivity to environmental changes.

The **sentient object programming model** (CORTEX project) [4] is a component-oriented reflective middleware based on OpenCOM aiming at sensor fusion, context extraction and reasoning. It is composed by three different categories of software entities: sensors, sentient objects and actuators. Sentient objects are intelligent software components that are able to act autonomously based on acquisition information and to cooperate with each others. The communication is event-based: sensors produce software events that are consumed by sentient objects who produce in response software events that are consumed by actuators.

The **iQueue framework** [9] aims at making software creation easier by facilitating the mechanics of composition. In this event-based framework, programmers choose events which are of interest for the context, specify fusion services using Bayesian networks and specify rules using an embedded inference engine. This should allow the programmer to focus only on the semantic of context use and composition.

4.4 Missing requirements

Some of the previously presented architectures allow the programmer to easily compose dynamically context components: addition of a new sensor, deletion of a sensor or replacement of a sensor by another. However, the automatic adaptation of an application to missing context informations or new context informations is not completely addressed.

In order to evaluate and to compare different contextual situations one from another, we need a measure of a context. In [24], we have introduce a notion of cost function. But cost composition can be done only on data that represent the same physical quantity. As for example, a cost in term of time, in term of money. In these cases, an addition of all the elementary costs will be done. Then, for one sensor, we need different elementary cost functions depending on the usage that will be done in the applications. Could it be foreseen? Accuracy on the data must also be taken into account. We then need to imagine more complex rules defined by the user, such as a user profile.

5. DISCUSSION

Sensed context informations and user input interactions are both acquired using different sensors. Depending of the situation, some of them have a great confidence (e.g. push button or information entered by a user in a comfortable position) while others are sensitive to noise (e.g. speech recognition, space localization or push button in transportation).

Input devices are generally intentionally used by the user in order to interact with an application while context acquisition is passive. The borderline is tiny: if a user knows how the system reacts to context modifications, he could use this information in order to modify the system: users always want to divert systems !

However, input devices are usually used for one application at a time. Since context-aware use needs to be transparent to the user, one context information can be used simultaneously by different applications, making it more difficult to be diverted by the user. The part of the system that is responsible of context management is independent of applications as it is for input devices since a long time!

The previously presented architectures for context or component models for input devices (2.3) always separate the acquisition step that is common to all applications from the interpretation one that depends on the use of the informations that will be done by the application.

The future challenges concern essentially the integration of input devices and context interpretation components at the heart of the application architecture in order to adapt them during application reconfiguration. The study briefly presented in this paper, but concerning more largely software architectures, shows that different software architectures are interesting in solving the reconfiguration problem. As pointed out by [3], there is a need to precise the semantic of composition in order to dynamically associate software components that constitute the business part of the application and to allow the application to interact with its environment.

6. REFERENCES

[1] J. Accot and S. Zhai. In ACM CHI'97, pp 295–302.
[2] R. Balakrishnan, T. Baudel *et al.* In ACM CHI'97.
[3] M. Baldauf, S. Dustdar *et al.* J. of Ad Hoc UbiComp 2006.
[4] G. Biegel and V. Cahill. In IEEE PerCom'04.
[5] W. Buxton. *ACM SIGGRAPH CG*, 17(1):31–37, 1983.
[6] S. K. Card, J. D. Mackinlay, and G. G. Robertson. *ACM Trans. on Information Systems*, 9(2):99–122, 1991.
[7] G. Chen and D. Kotz. In *IEEE Work. on Mobile Computing Systems and Applications*, 2002.
[8] D. Cheung, G. Joulie, F. Grillon *et al.* In *IEEE Int. Conf. on Systems, Man and Cybernetics*, vol 5, 2003.
[9] N. H. Cohen, A. Purakayastha, L. Wong *et al.* In *IEEE Int. Conf. on Mobile Data Management (MDM)*, 2002.
[10] J. Coutaz and G. Rey. In *ACM CADUI*, pp 283–302, 2002.
[11] A. K. Dey. *Pattern Recognition Letters*, 5(1):4–7, 2001.
[12] A. K. Dey, D. Salber, and G. D. Abowd. *HCI*, 16(2-4):97–166, 2001.
[13] P. Dragicevic and J.-D. Fekete. In *IHM-HCI*, 2001.
[14] P. Fitts. *J. of Experimental Psychology*, 46:381–391, 1954.
[15] J. Foley and V. Wallace. *Proc. IEEE*, 62(4):462–470, 1974.
[16] J. Foley, V. Wallace, and P. Chan. *IEEE Comp. Graph. and Appl.*, 4(11), nov 1984.
[17] R. Glassey, G. Stevenson *et al.* In *Work. on Middleware for Perv. and Ad Hoc Comp., Middleware*, 2003.
[18] P. D. Gray and D. Salber. In IFIP Int. Conf. on Engineering for HCI, LNCS 2254, 2001.
[19] S. Greenberg and C. Fitchett. In ACM UIST 2001.
[20] K. Henricksen and J. Indulska. In *IEEE Work. CoMoRea*, pp 33–37, mar 2004.
[21] K. Henricksen and J. Indulska. *J.of Pervasive and Mobile Computing*, 2(1):37–64, 2006.
[22] K. Hinckley, J. Tullio, R. F. Pausch *et al.* In ACM UIST'97.
[23] R. J. Jacob and L. E. Sibert. In ACM CHI, 1992.
[24] D. Lingrand, S. Lavirotte, and J.-Y. Tigli. In Workshop CAMS (OTM), LNCS 3762, 2005.
[25] J. D. Mackinlay, S. K. Card *et al.* HCI, 5:145–190, 1990.
[26] P. K. McKinley, S. M. Sadjadi, E. P. Kasten *et al. IEEE Computer*, 37(7):56–64, jul 2004.
[27] A.-M. Pinna-Déry and J. Fierstone. In Mobile HCI 2003.
[28] M. Resnick. *Comm. of the ACM*, 36(7):64–71, jul 1993.
[29] T. Starner IEEE Micro, 2001.