Traces using aspect oriented programming and interactive agent-based architecture for early usability evaluation: basic principles and comparison

Jean-Claude Tarby, Houcine Ezzedine, José Rouillard, Chi Dung Tran, Philippe Laporte and Christophe Kolski

1 Laboratoire LIFL-Trigone, University of Lille 1, F-59655 Villeneuve d’Ascq Cedex, France
{jean-claude.tarby, jose.rouillard, philippe.laporte}@univ-lille1.fr
2 LAMIH – UMR8530, University of Valenciennes and Hainaut-Cambrésis, Le Mont Houy, F-59313 Valenciennes Cedex 9, France
{houcine.ezzedine, chidung.tran, christophe.kolski}@univ-valenciennes.fr

Abstract. Early evaluation of interactive systems is currently the subject of numerous researches. Some of them aim at explicitly coupling design and evaluation by various software mechanisms. We describe in this paper two approaches of early evaluation exploiting new technologies and paradigms. The first approach is based on aspect oriented programming; the second one proposes an explicit coupling between agent-oriented architecture and evaluation agents. These two approaches are globally compared in this paper.

Keywords: Human-computer interaction, Early evaluation, Usability, Traces, Agent-based architecture, Aspect oriented programming.

1 Introduction

Interactive systems evaluation is a very rich research and application domain since the seventies. To evaluate how people use the interactive applications, many techniques and methods can be applied [18,19,22]. Some of them are widely used in companies and universities. New methods or variants of methods appear and are tested progressively, according with new needs and specificities coming from the emergence of information and communication sciences and technologies. Among them, automatic and semi-automatic methods and tools are considered as promising (see [16,17]). This paper concerns early usability evaluation, which is also currently particularly studied in the HCI literature.

We present in this paper two complementary approaches contributing to early evaluation during the first stages of a project. It is a question of explicitly coupling design and evaluation by exploiting innovative technologies and paradigms. This coupling aims to produce traces concerning how is used the interactive applications to evaluate their utility and utilisability, for example by comparing prescribed and effective tasks. The first approach exploits the paradigm of aspect oriented programming to integrate mechanisms of trace in interactive applications. The
concept of trace was the subject of various studies in HCI [16]. The second approach proposes an explicit coupling between agents constitutive of an agent based architecture, and several evaluation agents. These two approaches are first described; then, they are compared.

2 First approach for early usability evaluation: injection of mechanism of traces by aspects

2.1 Aspect-Oriented Programming

New paradigm of programming appeared in the middle of the Nineties, Aspect-Oriented Programming (AOP) results from the Xerox PARC. AOP must be perceived like an extension of Object-Oriented Programming: indeed, complementary generic mechanisms significantly come to improve separation of the concerns within the applications [14].

In a traditional approach, the business objects locally manage their technical constraints (identification/authentication, security, transactions, data integrity...). The duplication of these crosscutting elements in methods of classes leads to a phenomenon of dispersion and interlacing of the level system concerns and increases the complexity of the code. AOP allows the modularization of these elements by the addition of a new dimension of modularity, the aspect. The scope of the crosscutting concerns supported by AOP exceeds that of the current solutions like the EJB.

Join point, advice, aspect, pointcut, are the principal concepts introduced by AOP:

- A join point represents a particular location in the flow of the program instructions (beginning or end of method execution, field’s read or write access ...).
- Advices are methods which are activated when precise join points are reached: the mechanism of weaving inserts in the initial code the advices calls either in a static way (at compile-time) or in a dynamic way (during execution). Advice can execute before, after, or around the join point.
- An aspect is a module which allows the association between advices and join points by means of pointcuts.
- Pointcuts are used to define a set of join points on which will have to activate an advice. Furthermore, a pointcut allows capturing the execution context of join points. For a method call, this context includes the target object, the arguments of the method and the reference of the returned object, as many information of most useful for the injection of mechanism of traces.

Based on the principle of inversion of control (IOC), AOP thus extracts from the business code the dependences with the technical concerns by locating them in the aspects and by managing them from outside by the mechanism of weaving. It becomes consequently possible to be focused on business logic.

Moreover, AOP proposes the mechanism of introduction. This last allows the modification of classes, interfaces or even of existing aspects: it is possible to inject a
method or an attribute in a class, to add a relation of heritage, to specify that a class implements a new interface. For example, in the objective to automatically sort a collection of Java class instances, an aspect will declare that the latter implements the interface `Comparable` and inject the required method `compareTo` to it.

### 2.2 Traces by aspects

Thanks to the principle of separation of concerns, AOP can inject traces mechanisms in existing applications (cf. Figure 1, step ①) by writing aspects (step ②) which on the one hand listen user actions, method calls, changes in data values, etc., and on the other hand produce the traces. These aspects are then weaved with the initial code (step ③) which remains intact. The code produced by weaving contains then the initial code and code of aspects (step ④). The initial application can be used completely normally without the aspects or be traced with them (step ⑤). The mechanism of trace is thus disengageable without any effect on the initial code.

![Diagram](image)

**Fig. 1.** Injection of mechanism of traces by aspects

To produce a trace we need three types of information: data to be traced, when to produce the trace and where to store it. Traced data mainly relate to the functional core (and consequently the associated tasks) and the user interface (actions from the user, but also displayed data…). For example it is possible to trace the beginning, the end or the interruption of a task, the opening of a window, the selection in a drop-down list, etc. Because our work is use-oriented, it is easier to trace the actions of the user when the functional core and the user interface are built from a task oriented design method. Thus, if the application is designed with an evaluation-oriented approach as presented in [23], it is easy to recover other data such as the context of execution of the tasks, the role of the user (in CSCW for example), etc.
Most of the time, the traces are produced when a method is called or at the end of the execution of the method, and these methods may be associated to tasks. AOP provides us all the requested services for the production of traces (cf. before and after keywords present in AOP). Moreover, it is very easy to parameterize the productions of traces, for example to produce them by a dedicated thread, or only if a condition is true.

Today the traces are generated in XML files (step ①) whose contents are parameterized by a set of formats also written in XML (step ②). This allows us to generate traces in different formats while emitting same information from the traced application. Although we privilege traces in XML format, the external definition of formats will make it possible to generate very compact textual files (not XML).

With our approach the exploitation of traces is facilitated because we choose data that we want to trace, as well as the format for the result, contrary to approaches based on log files. The analysis of traces (step ③) produce statistics, task models (step ④), filtered information, etc. This side of our work is not presented in this paper. At the moment this analysis is done after the production of traces, but we plan to realise real time analysis in the future (for an adaptation of the application, to advise the user, etc.).

Our work is similar to works such as [2,5,6,9,10,11,24]. It uses AspectJ [4] but it could be made with other languages supporting AOP such as [3,21,25].

3 Second Approach for early usability evaluation: interactive agent-based architecture and evaluation module

3.1 Agent-oriented architecture for interactive systems

Several architecture models have been put forward by researchers over the past twenty years. Two main types of architecture can be distinguished: architectures with functional components (Langage, Seeheim, Arch and their derived models) and architectures with structural components (PAC and its derived models [7], the MVC model (Model-View-Controller ; from Smalltalk) and its recent evolutions, AMF and its variants [20], H4[8]. . .). The classic models of interactive systems distinguish three essential functions (presentation, control and application). Some models (such as the Seeheim and ARCH models) consider these three functions as being three distinct functional units. Other approaches using structural components, and in particular those said to be distributed or agent approaches, suggest grouping the three functions together into one unit, the agent.

These architecture models propose the same principle based on separation between the system (application) and interface. Thus, an architecture must separate the application and the interface, define a distribution of the services of the interface, and define a protocol of exchange. The interest to separate the interface from the application is to facilitate the modifications to be made on the interface without touching with the application. Figure 2 proposes a comprehensive framework for architecture [12,15], showing a separation in three functional components, called
respectively: interface with the application (connected to the application), controller of dialogue, presentation (this component being in direct relation with the user).

Fig. 2. An agent oriented architecture for interactive systems

These three components group together agents:
- the application agents which handle the field concepts and cannot be directly accessed by the user. One of their roles is to ensure the correct functioning of the application and the real time dispatch of the information necessary for the other agents to perform their task,
- the dialogue control agents which are also called mixed agents; these provide services for both the application and the user. They are intended to guarantee coherency in the exchanges emanating from the application towards the user, and vice versa,
- the interactive agents (or interface agents), unlike the application agents, are in direct contact with the user (they can be seen by the user). These agents coordinate between themselves in order to intercept the user commands and to form a presentation which allows the user to gain an overall understanding of the current state of the application. In this way, a window may be considered as being an interactive agent in its own right; its specification describes its presentation and the services it is to perform.

3.2 Principle of coupling between architecture based on agents and evaluation agents

Our starting objective was to propose a tool for collecting objective data, adapted to agent based interactive systems. This tool corresponds to an electronic informer; it consists of a program, invisible for the user (of the system to be evaluated), which transmits and records all the interactions (actions of the operator and reactions of the system) in a data base. The exploitation of this data base has the aim of then providing the evaluator with data and statistics enabling him/her to draw conclusions with regard to various aspects of utility and utilisability.
This informer being dedicated to the evaluation of agent-based interactive systems, it must be closely related to the architecture of the system to evaluate [13,26]. We are interested particularly in the interactive agents. This electronic informer, figure 3, consists of several informer agents deduced starting from architecture from the system to evaluate and more particularly starting from the multi-agent system concerning presentation. It is based primarily on the acquisition of information and specific data of the system to be evaluated (actions of the user and reactions of the system). Those will make it possible to rebuild the tasks really carried out by the user (a posteriori mode) and to confront them with the model of tasks to be carried out (a priori mode), according to confrontation principles described in [1].

Let us suppose a module of presentation made up of 6 interactive agents (each one being able to interact with the user), 6 evaluation agents will be instanced and connected to the interactive agents. During the interactions with the user, the 6 evaluation agents memorize in real time the data concerning interaction between the user and the 6 interactive agents. After the realization of the tasks, these data are analyzed automatically; using a specific user interface dedicated to the evaluator, these data are presented in time differed at this one. They can go from a bottom level, corresponding to simple user or system events, to higher levels (for example concerning task level). Examples are available in [26].
4 Comparison between the two approaches

A comparison of the two approaches is given in Table 1. The two approaches have common objectives: to gather data to compare predicted tasks and activity, and to highlight utility and usability problems. The ways used to obtain these data differ according to the approaches.

From the point of view of integration in the software engineering, the two approaches require particular specifications. Approach 1 (AOP approach) needs to know the methods and data that can be traced, as well as the formats of trace; this information can be collected during the specifications or after the implementation. Approach 2 (agent approach) requires the specification of the elements of the interactive system, and the evaluation agents. No particular architectural design is requested for the AOP approach, but agent approach requires that the design of the interactive system architecture must be based on interface agents, as well as the establishment of connections between the interactive agents and the evaluation agents. About the implementation, AOP approach automatically generates the code of the aspects and the weaving with the initial code of the application to be traced; agent approach requires programming the services of the interactive system agents and the evaluation agents.

From the user centred evaluation point of view, in addition to the fact that the two approaches can be coupled with other techniques such as interviews, eye tracking, etc., they use different modes to gather data: with AOP approach, data are automatically collected by the execution of the code issued from the aspect weaving on the initial application code; with agent approach, data are collected from the evaluation agents by observing the interactions between the interface agents and the user. To be collected with AOP approach, data must be accessible by a method (with the meaning of the object-oriented programming); this method can be public, inherited, etc. Time is accessible in the same way. Data collected with agent approach are potentially multiple (cf. Table 1).

In their current version the approaches use different languages. AOP approach uses Java and AspectJ; agent approach is based on C++. In the future, it is expected that AOP approach will be extended to other languages supporting AOP such as PHP, C++, etc., and that agent approach will uses Java.

Concerning the types of application, AOP approach currently can trace any application written in Java and supporting AspectJ. However, the traced applications are today mainly interactive applications (WIMP\(^1\) applications). In the future, it is planned that AOP approach will be applied to information systems, distance learning applications, and mobile applications. Agent approach is currently applied to information systems used in a context of supervision of network of bus and tramway. In the future, it should aim any type of information system.

\(^1\) Window, Icon, Mouse, Pull-down menu.
Table 1. Comparison between the two approaches.

<table>
<thead>
<tr>
<th>Traditional stages of software engineering</th>
<th>AOP approach</th>
<th>Agent approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preliminary or feasibility study</td>
<td>Injection of mechanism of traces by aspects</td>
<td>Explicit consideration of early evaluation in the project</td>
</tr>
<tr>
<td>Specification</td>
<td>Specification of: interactive system, parameters to be traced, formats of traces</td>
<td>Specification of: interactive system agents, evaluation agents</td>
</tr>
<tr>
<td>Architectural design</td>
<td>(empty)</td>
<td>Design of the interactive system architecture based on interface agents; connections between interactive system agents and evaluation agents</td>
</tr>
<tr>
<td>Coding</td>
<td>Generation of the code of the aspects and weaving between the code to be traced and the aspects</td>
<td>Coding of the services of the interactive system agents and evaluation agents</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>User-centred evaluation</th>
<th>Interaction data gathering</th>
<th>Execution of the weaved code</th>
</tr>
</thead>
<tbody>
<tr>
<td>(simultaneously with other possible methods: interviews, eye tracking, questionnaire, etc.)</td>
<td>Espionage by the evaluation agents of the interactions between interface agents and the user</td>
<td></td>
</tr>
<tr>
<td>Collected data</td>
<td>Any data accessible by a method (in the meaning of object-oriented programming) + Time</td>
<td>User and system events, errors, time of tasks execution, unused objects, number of help requests…</td>
</tr>
</tbody>
</table>

Goals
- Depends on how traces are exploited:
  - gathering data to compare predicted tasks and real activities, highlighting problems of utility and usability…

Languages
- Current: Java with AspectJ
- Intended: Any language supporting AOP
- C++
- Java

Types of application
- Current: WIMP applications
- Intended: Information systems
  - Distance learning applications
  - Mobile applications
  - Information systems used in a context of supervision of network of bus and tramway

The **advantages** of these two approaches are that they provide principles and mechanisms facilitating and prompting early evaluation. In addition AOP approach allows keeping intact the initial code and thus leading in parallel and/or serially the realization of the application and the realization of the mechanisms of traces.

The **disadvantages** are as follows. With agent approach, it is difficult to define for the moment the optimal number of evaluation agents (the first version contained an evaluation agent by interaction agent, and the new version will contain only one for the user interface; this aspect is under study). In addition, agent approach lets consider the need for new design methods of user interface envisaging a coupling between interface agents and evaluation agents. To be more effective, AOP approach needs design methods integrating aspects for the evaluation. That means for example that any potentially traceable data must be accessible by object methods.
5 Conclusion

The early evaluation field is the subject of active researches in the HCI community. For our part, we work on two complementary approaches. The first is based on aspect oriented programming; it allows the injection of mechanisms of traces in existing applications. The second is based on new possibilities offered by agent based approaches; it aims at ensuring a coupling between agent based architectures and evaluation agents. Although turned towards same objectives in term of evaluation, these two approaches have different characteristics, advantages and disadvantages which were compared in the paper.

For these two approaches, the research perspectives are numerous: it is important to study adapted design methods, to improve the current mechanisms, to test them in various application domains.

Acknowledgments. The present research work has been supported by the “Ministère de l'Education Nationale, de la Recherche et de la Technologie », the « Région Nord Pas-de-Calais » and the FEDER (Fonds Européen de Développement Régional) during the projects SART, MIAOU and EUCUE. The authors gratefully acknowledge the support of these institutions.

References