CREATIVE MATH. & INF.
16 (2007), 151 - 158

Dedicated to Professor Ioan A. RUS on the occasion of his 70^{th} anniversary

An agent based user interface evaluation using aspect oriented programming

ADRIANA M. TARŢA, GRIGORETA S. MOLDOVAN and GABRIELA ŞERBAN

ABSTRACT. Human-computer interaction design has an essential role in the success or failure of a software product. The user interface reflects this aspect of the system. In this paper we propose a new alternative for evaluating user interfaces using an agent-based approach. The Intelligent Agents domain is an important research and development area in the field of Computer Science and of Artificial Intelligence, particularly [16]. It provides a new mechanism for problem solving and a new user-computer interaction method. In our proposal, based on task models (task trees), agents are used for monitoring and assisting users in interaction with the system. Task models [17] are used in the user centered design context in order to give valuable information about the sequence of actions the user must perform to accomplish his/her goals. In order to separate the agent from the evaluated software system, we use a recently developed programming paradigm, Aspect Oriented Programming [6].

1. INTRODUCTION

1.1. **Human-Computer Interaction.** Nowadays, computers are used more and more in performing current tasks. The widespread of computers has lead to a situation where users with different expertise level (novice, experts) have to perform various tasks. The problem that arose was that systems having a good functionality and remarkable performances were avoided to be used by their users. Many researches have been developed in order to determine why systems having a good internal quality were considered not usable. The source of this attitude was located at the user interface level, including the man-machine interaction design. These problems have been considered subject of research in a recent discipline called Human-Computer Interaction (HCI) [2]. HCI is a multidisciplinary field of study, including knowledge from psychology, sociology, ethnography. A solution proposed in order to solve the above mentioned problems is a new design paradigm called User Centered Design. In this approach, the system design starts from tasks the users should perform. Task analysis techniques have been taken from psychology and adapted to system design. The task models built after the task analysis process are a valuable source of information in the design process.

1.2. **Agents.** An **agent** ([12]) is anything that can be viewed as **perceiving** its environment through *sensors* and **acting** upon that environment through *actions*.

One of the task of an agent is to assist the user, achieving tasks in his place, or teaching the user what he should do. An *agent* is characterized by:

Received: 15.09.2006. In revised form: 10.11.2006
2000 *Mathematics Subject Classification.* 68N99, 68P99, 68T99.
Key words and phrases. *User interface evaluation, agents, aspect oriented programming.*

- the *architecture part*, or the agent's behavior - the action performed after any given sequence of percepts;
- the *program part*, or the agent's built-in part - the internal functionality of the agent.

An *artificial intelligent agent* should be endowed with an *initial (built-in) knowledge* and with the capability of *learning*. The learning capability ensures the agent's *autonomy* - the capability of deducing his behavior from its own experience.

The aim of Artificial Intelligence is to design the agent program: a function that implements the agent mapping from percepts to actions, the *intelligence* of an agent being included in his program part. At a given moment, the agent will choose the best way of action, as he was programmed to do it. In situations in which the program has incomplete information (knowledge) about the environment in which the agent acts and lives, *learning* is the only way for the agent to acquire the knowledge he needs in order to achieve his task.

So, an important task is to design the program part of an intelligent agent, and even more, to implement the capability of learning.

1.2.1. *Software Agents.* Intelligent **software agents** are a new class of software that act on behalf of the user to find and filter information, negotiate for services, easily automate complex tasks, or collaborate with other software agents to solve complex problems. Software agents ([10]) are a powerful abstraction for visualizing and structuring complex software. Procedures, functions, methods, and objects are familiar software abstractions that software developers use every day. Software agents, however, are a fundamentally new paradigm unfamiliar to many software developers. The central idea underlying software agents is that of delegation. The owner or user of a software agent delegates a task to the agent and the agent autonomously performs that task on behalf of the user. The agent must be able to communicate with the user to receive its instructions and provide the user with the results of its activities. Finally, an agent must be able to monitor the state of its own execution environment and make the decisions necessary for it to carry out its delegated tasks. There are two approaches to building agent-based systems: the developer can utilize a single stand-alone agent or implement a multi-agent system. A stand-alone agent communicates only with the user and provides all of the functionality required to implement an agent-based program. Multiagent systems are computational systems in which several agents cooperate to achieve some task that would otherwise be difficult or impossible for a single agent to achieve. Agents within a multiagent system communicate, cooperate, and negotiate with each other to find a solution to a particular problem.

1.2.2. *Agent-Based Systems.* An **agent-based system** is one in which the key abstraction used is that of an agent. An agent is a system that enjoys the following properties ([18]):

- autonomy: agents encapsulates some state and make decisions about what to do based on this state, without the direct intervention of human or others;
- reactivity: agents are situated in an environment (a physical world, a user via a graphical user interface, a collection of other agents, the INTERNET),

are able to perceive this environment (through the use of potentially imperfect sensors), and are able to respond in a timely fashion to changes that occur in it;
- pro-activeness: agents do not simply act in response to their environment, they are able to exhibit goal-directed behavior by taking the initiative;
- social ability: agents interact with other agents via some kind of agent-communication language ([3]).

Even if the discipline of intelligent agents has emerged largely from research in Artificial Intelligence, the only intelligence requirement we generally make for the agents is that they can make an acceptable decision about what action to perform next in their environment, in time for this decision to be useful ([19]). Other requirements for intelligence will be determined by the domain in which the agent is applied: not all agents will need to be capable of learning, for example. Thus, the application and exploitation of agent technology can be viewed, primarily, as a computer science problem. Agents are simply software components that must be designed and implemented in much the same way that other software components are.

1.3. **User Interface Evaluation.** In order to provide useful systems, the companies are conducting evaluation sessions for their products. There are many methods of evaluating user interfaces (focus groups, guidelines review, user testing, etc.), most of them being expensive. An alternative to these methods is considered the automatic evaluation of the user interface, based on usability metrics, pattern matching or task performance ([5]). Literature on user interface design frequently uses the term **usability**. Several usability definitions exists, some of them being similar and some of them being very different. ISO ([4]) defines usabilty as the extent to which a computer system enables users, in a given context of use, to achieve specified goals effectively and efficiently while promoting feelings of satisfaction. **Usability evaluation (UE)** consists of methodologies for measuring the usability aspects of a system's user interface (UI) and identifying specific problems [2, 9]. Usability evaluation is an important part of the overall user interface design process, which consists of iterative cycles of designing, prototyping, and evaluating [2, 9]. Usability evaluation is a process that entails many activities. Common activities include:

- *Capture* collecting usability data, such as task completion time, errors, guideline violations, and subjective ratings.
- *Analysis* interpreting usability data to identify usability problems in the interface.
- *Critique*: suggesting solutions or improvements to mitigate problems [5].

Usability testing with real participants is a fundamental usability evaluation method ([9, 13]). It provides an evaluator with direct information about how people use computers and which are the problems of the interface being tested. During usability testing, participants use the system or a prototype to complete a predetermined set of tasks while the tester records the results of the participants work. The tester then uses these results to determine how well the interface supports users task completion as well as other measures, such as number of errors and task completion time. Automation has been mainly used in two ways within

usability testing: automated capture of user data and automated analysis of these data according to some metrics or a model.

Many usability testing methods require the recording of the actions a user makes while exercising an interface. This can be done by an evaluator taking notes while the participant uses the system, either live or by repeatedly viewing a videotape of the session: both are time-consuming activities. As an alternative, automated capture techniques can automatically log user activity. An important distinction can be made between information that is easy to record but difficult to interpret (e.g., keystrokes) and information that is meaningful but difficult to automatically label, such as task completion. Automated capture approaches vary with respect to the granularity of information captured.

Task-based approaches analyze discrepancies between the designer's anticipation of the user's task model and what a user actually does while using the system. Evaluators can use this approach to compare user and designer behavior on specific tasks and to recognize patterns of inefficient or incorrect behaviors during task completion. Task based approach can be used to detect additional patterns, including immediate task cancellation, shifts in direction during task completion, and discrepancies between task completion and task model.

1.4. **Aspect Oriented Programming.** Aspect oriented programming (AOP) is a new programming paradigm that addresses the issues of *crosscutting concerns* [6]. A crosscutting concern is a feature of a software system whose implementation is spread all over the system. Well-known examples of crosscutting concerns are logging and security. In order to implement a crosscutting concern, AOP introduces four new notions: *joinpoint*, *pointcut*, *advice* and *aspect* ([6]).

The aspects are integrated into the system using a special tool called *weaver*. Nowadays, there are extensions that support AOP for well-known programming languages (i.e., AspectJ for Java [1]) which are used in industry, too.

This paper proposes a new agent based user interface evaluation approach, based on task models and aspect oriented programming.

The paper is structured as follows. The task analysis basic concepts and methods are discussed in Section 2. Our new approach in user interface evaluation using agents and Aspect Oriented Programming is described in Section 3. Section 4 presents a small case study of our approach. Conclusions and future work are given in Section 5.

2. Task Analysis and Modeling

Before discussing the task analysis and modeling techniques, we have to understand the meaning of *task* concept. A *task* is an activity that should be performed in order to reach a goal. A goal is a desired modification of state or an inquiry to obtain information on the current state of an object (system) [8].

Task analysis is the process of gathering data about the tasks people perform. The process of structuring this data and gaining insight into the data is called *task modeling*.

In the design of interactive systems the task analysis can be used with different goals:

- requirement analysis - when through a task analysis designers identify requirements that should be satisfied in order to obtain an useful system;
- design of interactive applications - the information from task models is used to better identify the interaction techniques and the presentations of the application (in this case the modeling technique should provide temporal information about the logical order of tasks);
- usability evaluation - the system task model and the user task model are compared in order to get information about the matching between these models; also, having a structured task model some techniques like KLM (Keystroke Level Model) can be applied to get information about the time needed to perform a task - this kind of approximation may be also used to compare different task models addressing the same problem.

Task models are built after task analysis is performed. Task analysis aims to identify the relevant tasks and how activities are performed currently.

The goal of task models is to identify useful abstractions highlighting the main aspects that should be considered when designing interactive systems. The main advantage of task models is that they represent the logical activities that an application must support.

A task model in the design of an interactive system describes a set of activities that users intend to perform while interacting with the system. Two types of task models have been also identified: *the system task model* that provide information about how the designed system requires tasks to be performed, and *user task model* which is how users expect to perform their activities. It is desirable that these two models be very similar, otherwise some usability problems will be present.

Task models are built for many different situations: usually, a task model is built when designing a new application with the goal of obtaining precise information about: the order of task performance, objects from the domain manipulated in the task performance process, agents and roles responsible for the task performance, events triggering task performance, preconditions and postconditions for task performance. Also, a task model can be built for an existing application, in this case the goal is to understand the underlying design, to analyze its limitations, and solutions to overcome them. Task model can address the problem of designing an entire application, or just a part of it.

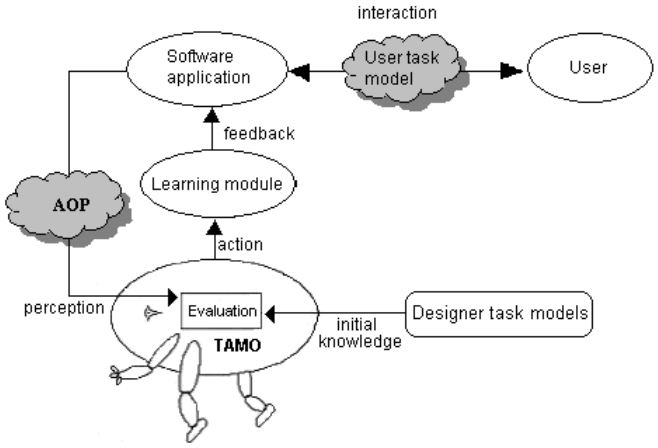
Task models describe the semantic and temporal relations between the identified tasks. Task models are usually represented as *task trees*, describing the hierarchical structure of tasks ([8, 17]).

3. An Agent Based Approach for Usability Evaluation

In this section we present an agent based usability evaluation approach using Aspect Oriented Programming (AOP). The proposed approach uses **TAMO** agent in order to compare the user task model with the designer task model.

The environment in which **TAMO** agent is situated is a software environment. The agent uses AOP in order to gather information about its environment.

In Figure 1 we propose the overall architecture of an agent-based system for usability evaluation.



As Figure 1 shows, the component parts of the proposed system are:

- The *software application* (**SA**). It is a software system for which we intend to evaluate its usability. The application has an associated *task tree* (**TT**) that was developed by the UI designer of the application.
- The *user* (**U**). It is a person that uses **SA** in order to perform a predetermined set of tasks.
- The *agent* (**TAMO**). It is a software agent whose goal is to monitor the user's actions and to compare them with **TT** in order to be able to assist the user. The initial knowledge of the agent is the **TT** associated with **SA**. The *perceptions* of **TAMO** are the actions that the user **U** performs on **SA**. The program part of **TAMO** consists in comparing the two task trees. In the general case, the action of the agent consists in sending the results of the comparison to the *Learning module*. **TAMO** can be endowed with the capability of *learning*, in which case **TAMO** agent becomes an interface agent ([7]), a kind of personal assistant of the user.
- The Learning module (**LM**). It is the module of the software system that, in a general architecture, learns from the users' behavior and sends a feedback to the UI evaluator. It can be a learning agent.

In the current version of our evaluation system, the *Learning module* is not included, yet. The action that **TAMO** performs is to save the results of the comparison on a storage device. We are currently working on adding the *Learning module*, that will grow the accuracy of the evaluation results.

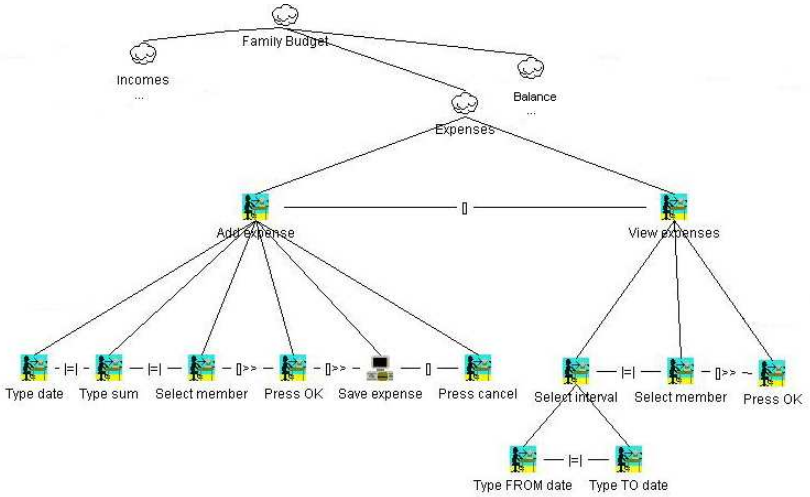
4. A Case Study

In order to test our system we have developed a small application for family budget management having the following functionalities: adding an expense/ income, viewing expenses/incomes for a period of time and computing the balance for a period of time. An income/expense has a date, a value, a category, and a family member. The functionalities are available using menus or toolbar buttons.

The architecture of the agent based system that we propose for usability evaluation is the one described in Figure 1. The *Learning module* is not included in the current version of our system.

The AOP module is used for capturing user's events: mouse clicking, text entering, menu choosing, etc. These events are received by **TAMO** agent, that will rebuild from them the task the user has performed, creating the *User Task Tree* (**UTT**).

TAMO also has an initial knowledge, that is the *Designer's Task Tree* (**DTT**), as shown in Figure 2. In our implementation, the initial knowledge is stored in an XML file.



The *evaluation* module of the agent, that gives its behavior, makes a comparison between **UTT** and **DTT**. It verifies if **UTT** is a subtree of **DTT** and it saves the results of the comparison on a storage device.

5. Conclusions and Further Work

We have presented in this paper a new *agent based* approach for usability evaluation and we have proposed the architecture of an usability evaluation system. The described system contains an agent, called **TAMO**, that has an evaluation module for comparing the user and designer's task models.

Further work can be done in the following directions:

- To add the learning module to our current version of the system. This learning module will grow the accuracy of the usability evaluation system.
- To add the learning capability to **TAMO** agent, in order to assist the user in performing his/her tasks (if needed).
- To define quality measures for evaluating the obtained usability results.
- To apply our usability evaluation system to complex software systems.
- To compare the results obtained by our system with the results of other usability evaluation approaches: **SUS** ([15]), **SUMI** ([14]).

REFERENCES

[1] "AspectJ Project," http://eclipse.org/aspectj/
[2] Dix A., Finlay J., Abowd G.D. and Beale R., *Human-Computer Interaction*, 3^{th} *edition*, Pearson, Prentice Hall, 2004
[3] Genesereth M. R. and Ketchpel S. P., *Software agents*, Communications of the ACM, 37(7), 1994, pp. 48-53
[4] ISO 9126, Software product evaluation - Quality characteristics and guidelines for their use
[5] Ivory M. and Hearst M., *The State of the Art in Automating Usability Evaluation of User Interfaces*, ACM Computing Surveys, Vol. 33, No. 4, December 2001, pp. 470–516
[6] Kiczales G., Lamping J., Menhdhekar A., Maeda C., Lopes C., Loingtier J.M. and Irwin J., Aspect-Oriented Programming, *Proceedings European Conference on Object-Oriented Programming*, Vol. 1241, Springer-Verlag, 1997, pp. 220–242
[7] Maes P., *Social interface agents: Acquiring competence by learning from users and other agents*, (Technical Report SS-94-03), AAAI Press, 1994, pp. 71–78
[8] Mori G., Paternò F. and Santoro C., *CTTE: Support for developing and Analyzing Task Models for Interactive System Design*, IEEE Transactions on Software Engineering, Vol. 28, No. 9, 2002
[9] Nielsen J., *Usability Engineering*, Boston, MA: Academic Press, 1993
[10] Nwana H. S., *Software Agents: An Overview*, Knowledge Engineering Review, 1996
[11] Paternò F., *Model-based Tools for Pervasive Usability*, Technical Report, University of Pisa (Italy), 2004
[12] Russell S.J. and Norvig P., *Artificial intelligence. A modern approach*, Prentice-Hall International, 1995
[13] Shneiderman B., *Designing the user interface: strategies for effective human-computer interaction*, Addison-Wesley Longman Publishing Co., Inc., 1986
[14] SUMI, sumi.ucc.ie.
[15] SUS, www.usability.serco.com/trump/documents/Suschapt.doc.
[16] Weiss G. (Ed.), *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, MIT Press, 1999
[17] van Welie M., *Task-based User Interface Design.*, PhD Thesis, Vrije Universiteit Amsterdam, 2001
[18] Wooldridge M. and Jennings N. R., *Intelligent Agents. Theory and practice*, The Knowledge Engineering Review, 10(2), 1995, pp. 115–152
[19] Wooldridge M., *Agent-Based Software Engineering*, Mitsubishi Electric Digital Library Group, London, 1997

BABEŞ-BOLYAI UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE
M. KOGALNICEANU 1
400084 CLUJ-NAPOCA, ROMANIA
E-mail address: adriana@cs.ubbcluj.ro
E-mail address: grigo@cs.ubbcluj.ro
E-mail address: gabis@cs.ubbcluj.ro