Journal of Information Systems Engineering & Management, 1:4 (2016), 51
ISSN: 2468-4376

http://www.lectitojournals.com

Supporting GUI Exploration through USS Tool

José Luís Silva[1*], J. D. Ornelas[2], J. C. Silva[3]

[1] Instituto Universitário de Lisboa (ISCTE-IUL), PORTUGAL & Madeira-ITI, PORTUGAL
[2] Universidade da Madeira, PORTUGAL
[3] Instituto Politécnico do Cávado e do Ave, (EST-DIGARC), PORTUGAL

*__Corresponding Author__: jose.luis.silva@iscte.pt

Citation: Silva, J. L., Ornelas D., J., Silva, J. C., (2016) Supporting GUI Exploration through USS Tool, *Journal of Information Systems Engineering & Management, 1*:4 (2016), 46.

doi: http://dx.doi.org/10.20897/lectito.201651

Received: June 13, 2016; **Accepted**: October 19, 2016; **Published**: November 8, 2016

ABSTRACT

Advances in usability and design techniques (e.g. user-centered design) try to facilitate the use of interactive systems. However, users still have to adapt to interactive systems, i.e. they have to learn the steps required to accomplish a task either by trial and error or by obtaining help. While advanced users are usually able to adapt without much effort this is far from being the case with beginners. Some interactive systems offer different interaction styles in an attempt to meet the needs of all types of user but this is not the case with all interactive systems. In this sense, we present an approach to support the use of any interactive system making use of enriched models and picture-driven computing to achieve tasks automation. The USS tool (User Support System) is the basis to the adaptation of interactive systems accordingly to the users' needs. The approach provides the foundation for the addition of help (based on demonstration) to any graphical user interfaces (GUI) facilitating learning and use. The work is illustrated by a case study and completed with a preliminary user evaluation which provides insights about the validity of the approach.

Keywords: Interactive systems, Picture-driven computing, Automation, Task Modeling, User support system

INTRODUCTION

The concept of interactivity is recent, it comes from physics and have been incorporated by other fields of knowledge. In the computer field, interactivity comes against a new conversational dimension of information, translated by a bi-directionality. In this sense, there is a separation between who sends and who receives the message. In the case of an interactive system, a dialogue communication is established in the exchange.

Interactive systems have undergone significant developments, much due to the constant advances that technology has been targeted. Progress has taken place to explore new forms of interaction that become increasingly

simple/easy to use by users in general (Knabe K., 1995). Interactive systems can be described as systems that allow continuous communication (transfer of information) between a computer and a user (Piyawadee Sukaviriya et al., 1990).

There are many ways and means of interaction between man and machine that have been applied, through the traditional command line interface, graphical user interfaces, the style interaction question-answer, keyboard, mouse, voice recognition, touch screen, etc. With regard to interaction styles, the most common is the menu, while the question-answer method, although simple and self-explanatory, is suitable only for users with low level of expertise. However, despite continuous progress both in terms of usability and intuitiveness, users (especially those with low skills) of interactive systems continue to encounter many challenges. One of them is due the fact that several systems do not take into account the needs of each user profile and the context in which they are inserted to adapt to them. To facilitate the use of interactive systems new solutions have emerged, such as context-sensitive help, tutorials, and new paradigms such as 3DUI (3 Dimension User Interface) that uses for instance a Kinect as input and a CAVE as output. However, none of the solutions proves fully adequate. It is true that users learn faster through tutorials illustrated by screenshots than to just read the text (Guy A. Boy, 1998), but users have many difficulties in locating the interface element in tutorials (O. Al-Shara and A. Dix., 2004). Users find themselves forced to adapt to different systems in order to meet their needs. They have to search for information (help manuals) or use the method of trial and error to learn how to interact with the system or to know what steps must follow to complete a given task. Thus, any improvement considering user support in the use of interactive systems lead to important contributions in this area.

The work presented here provides users with an innovative method of interacting with any existing interactive system. The approach is based on enriched tasks models and picture-driven computing and aims to solve usability problems and improve efficiency while interacting with GUIs (using automated tasks). At this point, due to the interest in knowing the behavior of the interactive systems, task models, identifying the interaction between the user and the system, represent an ideal starting point to achieve the goal of this project.

The approach applies to different contexts, and provides a tool for developers, which easily allows the automatic creation of scripts. The results of the approach permits, even users with little or no knowledge of Information Technology (IT), to perform the required tasks. Typically, users have to learn how to perform tasks by reading information (e.g. help and support) or by exploration (trial and error). The work presented in this paper aims to provide the basis for an innovative method for users to interact with any developed interactive system, independently of the Operating System and without requiring access to the application source code. The main goal is to enable an effective adaptation of interactive systems for users (beginners in particular) that would result in an increase in the system's usability and a decrease in the prior user knowledge required.

This paper presents USS, a tool that provides the basis for improving user interaction with GUIs, solving several usability and efficiency problems via task automation, enriched task models and picture-driven computing. An initial version of this paper was published in (Ornelas, J. D. et al., 2016b). The current paper extends the previous paper by articulating the User Support System approach more thoroughly. It describes and illustrates the approach in more detail by means of a more complex case study. Additionally, an evaluation and corresponding results are also presented and discussed.

The article is structured as follows: Section 2 describes background concepts and presents some related studies; Section 3 presents the implemented approach; Section 4 presents a simple application of the approach used to simplify the explanation of the implementation and makes it easier to understand. Section 5 illustrates the work by means of a case study and a user evaluation with respective results. Finally, discussion and conclusions are presented in Sections 6 and 7.

BACKGROUND

A brief description of some concepts is presented in this section to provide the reader with the basis for understanding the developed work. Related studies are described in the last subsection.

A. Task Automation

In Human-Computer Interaction the automation aims to simplify task execution by, for instance, reducing the number of steps/actions the user has to perform. Task automation might always appear to be beneficial but too much automation might lead to bad results (Guy A. Boy., 1998). To avoid such situation, the understanding of what should or should not be automated is fundamental. Célia Martinie et al., (2011b) presented an approach to this end. Their work is based on task models to identify which tasks a user can perform in an interactive system and

consequently the best automation level. Due to the complexity of frequent interactions these advances are not enough and users still need to have some previous knowledge to perform a task efficiently in an interactive system.

Ideally tasks are identified and consequently modeled in the early phases of the development process. However, in systems already developed and/or without access to their source code other approaches must be considered (e.g. picture-driven computing). Parasuraman et al. (2000) states that the automation can be performed at various stages of the interaction and on different classes of functions (i.e. acquisition, analysis, decision and execution). The automation of these classes might lead to varying implications in terms of performance, trust and cost. In regard to implementation the automation can be implemented with scripts or macros and can vary in its degree of complexity from no automation to full automation (where user participation is not required).

Task models are hierarchical representations of the tasks that a user can perform in an interactive system and describe how each task can be performed. The two most recognized notations and therefore most frequently used to model tasks are: HAMSTERS (Racim et al., 2014) (Human-centered Assessment and Modeling to Support Task Engineering for Resilient Systems) and CTT (ConcurTaskTrees) (Li, Jiao et al. 2010). On one hand the HAMSTERS tool is an open source solution dealing with both small case studies (laboratory) and real situations (business) and requiring very little training/learning time. It is inspired by existing notations and tools, combining the advantages of all of them. The tool allows:

1) the addition of conditions for the execution of tasks;
2) the representation of information flow between tasks;
3) the simulation of task sequence execution.

On the other hand, the CTT notation is the most common approach to task analysis, being widely used both at the academic and industrial level. The CTTE (ConcurTaskTrees Environment) tool allows for the introduction of informal textual description of a user case/scenario and supports simultaneous tasks, metric calculation, task performance evaluation and interactive simulation.

With regard to the stated features and after a trial period with both tools, CTT was chosen mainly because of its compatibility with complementary tools (in particular the MARIAE) that can be useful at a later stage of the work (as will be seen later).

B. Picture-Driven Computing

Before the application of automation to a system the identification of the tasks to automate is required. This is a procedure that can be done at different stages of the development process of the target software, as well as in legacy systems. In the latter case, where it is not possible to have access to application's source code, there is a need for an alternative way to carry out automation tasks. In those situations Picture-Driven Computing is seen as a viable alternative (Chang, Tsung-Hsiang et al., 2010). This new paradigm offers software developers a fully logical alternative. The approach uses computer vision algorithms to analyze instantly the content and evolution of a graphical user interface (Silva, J. C. et al., 2014).

As stated by Kourousias et al. (2010), the picture-driven computing paradigm can be a good ally for accessibility design (refers to the design of products or services for people with disabilities) of computer systems. This ensures access without help and indirect access with technologies to support people (Phillips, Betsy, and Hongxin Zhao. 1993).

Sikuli (Tom Yeh et al. 2009) is a picture-driven computing tool that uses image recognition to identify and control GUI components. The software allows users to develop scripts that work on graphical user interfaces, through the use of screenshots of these GUIs, and allowing visual scripting and automation of actions on any graphical interface. Alternative solutions to Sikuli are Automa, RIATest and Eggplant Functional.

Automa (Lund, A.M., 2001)) is a tool for Windows that automates repetitive tasks on interactive systems. The tool allows for workstation control using simple commands such as "start", "click" and "write". The sequence of input commands can be stored in a file which can be reproduced in different ways, by clicking on a button within a specified range or from a test management tool.

RIATest (Automate testing of web applications, 2016) is an automation tool for GUI tests. It is capable of automating any item on the screen, which is accessible through the Windows UI Automation API. The tool identifies an application's GUI elements through the Object Inspector, using simple but powerful location features. The tool allows the user to pause the execution of a script, edit it, and then continue, without restarting the execution.

Finally, Eggplant Functional tool (eggPlant range, Test automation tools, 2016) executes functional test automation, using an approach based on patterns of pictures. The tool enables the writing of tests in a very intuitive way. It uses sophisticated recognition algorithms to locate objects on the screen and thus control the device and perform the interaction in the same way that a user does. This approach allows the test of any technology on any platform to be made from the user perspective. For example, to click on the "OK" button the program analyzes the

screen through image recognition algorithms, finds the button and then triggers an event at the system level to click on the button.

C. Related Work

The work of Eagan et al. (2011) enables the reduction of the gap between what is provided by an application and what is expected by the users. Their approach allows the modification of the interface and behavior of an application at run time without source code access. Another related work is that of Yeh et al. (2011) that provides help to GUI users at run time. Célia Martinie et al. (2011a) presented an approach to provide contextual help about sequences of actions to be performed in order to make a given task available. Pangoli et al. (1995) developed an approach to obtain task-oriented help from the user interface specification. Other bodies of work (e.g. (Piyawadee Sukaviriya and James D. Foley., 1990)) have looked at the automation of the creation of graphical illustrations (as it can be time-consuming and expensive) for teaching users about a software application. Users learn more quickly by following tutorials illustrated by screenshots than by reading only text (Harrison, S. M., 1995) but they often have difficulty in locating the interface element present in the tutorial (Knabe, K., 1995). Some works address this difficulty (e.g. (Bergman, L. et al. 2005) but none execute the help in the interface; in other words, they do not provide help in automating the execution of the desired task.

Palanque et al. (1993) looked at the generation of contextual help by adding annotation to the Petri net model used to build the GUI. His work addressed the generation of context-sensitive help from a model of Petri nets dialogue, by adding network notes. This method allows the analysis and verification of the dialogue specification in order to verify the interface behavior, and even automatically generate a contextual help system. Paternó et al. (1995) describes another approach, i.e. a task-oriented approach that supports the automatic generation of help. The help information is structured according to user tasks (which are associated with interaction objects). The UI drawing is made, based on the specification of tasks involving the user view of the system functionality. This is used to produce the design of the UI and help.

Several approaches providing contextual help to users were presented however, in this work we are interested in a different kind of help. Our focus is to provide automated/assistive execution of tasks instead of providing contextual help. The next sections describe the approach, its application to a case study and its preliminary evaluation with end users.

THE APPROACH

The manual creation of Sikuli script for each interactive system is not a viable solution (time-consuming process) for adapting GUIs to user needs. Task models are usually available from the early phases of development. Using this fact, we have developed a tool (designed to be used by developers) enabling the automatic generation of parameterized Sikuli scripts from enriched task models. The automatic creation of Sikuli scripts is the main challenge for the adaptation of interactive systems to users. In this section a description of each phase of the approach is presented:

A. the presentation of the rules for the enrichment of the task model required for the automatic script creation;

B. the steps to create the scenarios that complement the script creation process;

C. the algorithm used for the script creation process.

A. Task Model Enrichement

The original task models do not possess enough information for the script creation to be accomplished. Part of the missing information is the screenshots required by Sikuli scripts. For this purpose a notation was developed for inclusion of the screenshot location in the *description* field of each task (see Figure 1). The information was added to this field because it does not interfere with the interpretation and execution of the model. Another purpose of the description field is to include additional information of the Sikuli function to be used in the resulting script (e.g. waiting times, dialog messages, etc.).

Besides the inclusion of additional information in the description field of relevant tasks the names of the tasks must be adjusted according to the Sikuli script. For example, to show a popup window, the name of the task must start with the *popup* reserved word followed by the name of the task. In the same way a set of rules have been developed for every possible situation and have to be satisfied for a correct enrichment of the task model. Those rules are presented in the remaining text of this subsection.

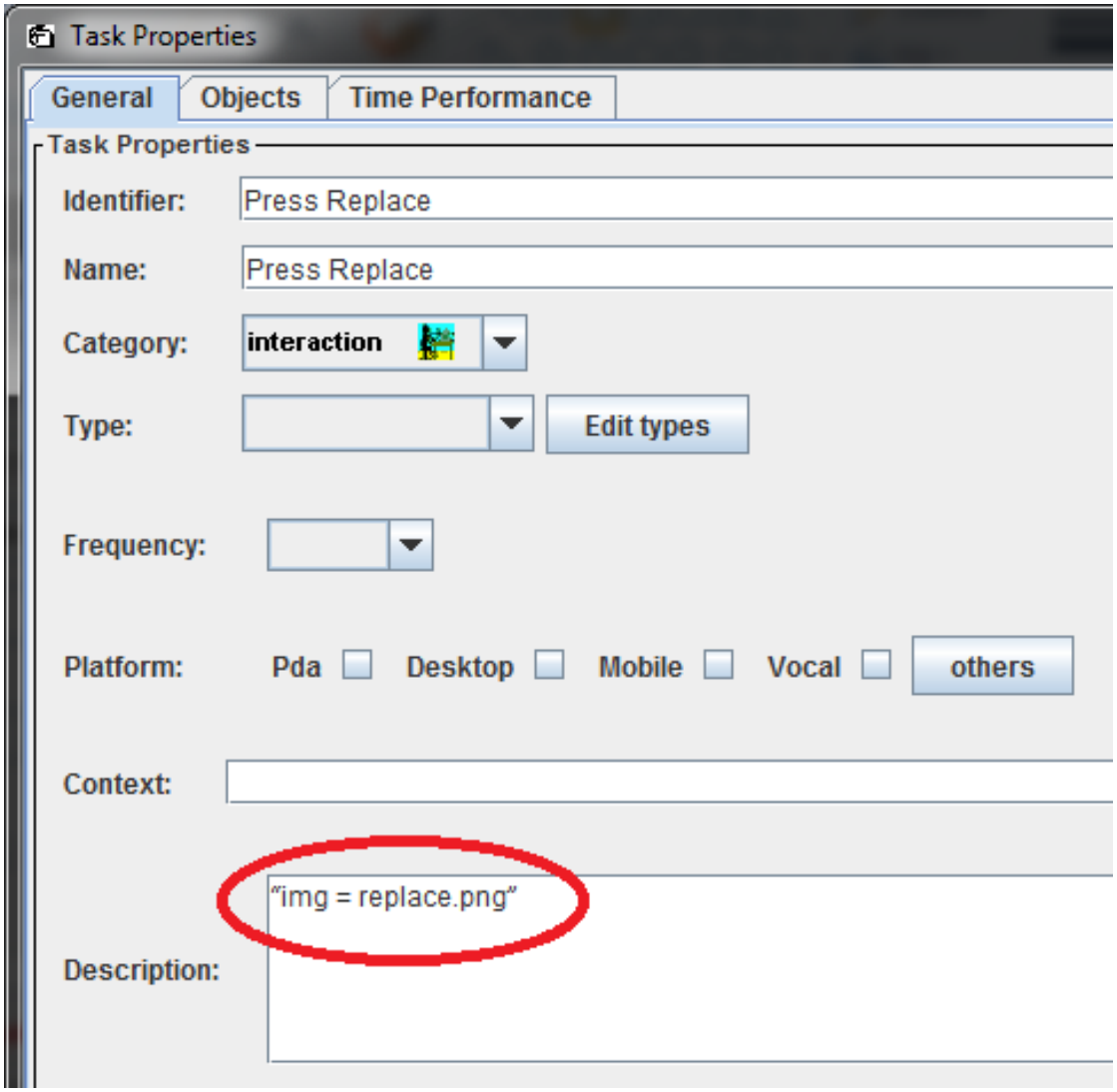

Figure 1 - CTTE task properties (description field)

The following description will present the rules to be used for:

i) the attribution of the names of the tasks (*Name* field);
ii) the information added to the tasks (*Description* field).

The Sikuli function associated to each rule and a description of when/how the rule should be used is also presented.

Table 1. Click functions

Rule	Sikuli function	Description	Value of the *Description* field
Press <task>	click(*img*)	Used for a click (left mouse button) on an element (e.g. *img* parameter)	The name of the image e.g: "img = name.png"
PressR <task>	rightClick(*img*)	Used for a click (right mouse button) on an element (e.g. *img* parameter)	The name of the image e.g: "img = name.png"
PressD <task>	doubleClick(*img*)	Used for a double click on an element (e.g. *img* parameter)	The name of the image e.g: "img = name.png"

Table 1. Function for data introduction/edition

Rule	Sikuli function	Description	Value of the *Description* field
Enter <task >	text = input(); paste(*img*, text);	Used to ask the user to manually insert data (*text* parameter) in a specific field (e.g. *img* parameter)	The name of the image e.g.: "img = name.png"
EnterPassw <task>	passwd = input("Introduce password", hidden=True); paste(*img*, passwd);	Used to ask the user to manually insert a password in a specific field (*img* parameter)	The name of the image e.g.: "img = name.png"
EnterSemiAuto <task>	paste(input(text))	Used to ask the user to manually insert data indicating what is expected (*text* parameter)	The text to be inserted in the dialog window to ask the information (*text*)
EnterAuto <task>	paste(text + Key.ENTER)	Used for automatic data introduction provided (*text*)	The text to be introduced (*text*)
EnterKey <task>	type(key)	Used to automatically press a key or manual text introduction	The key, e.g. : Key.F11
EnterCopy <task>	Type("c", KeyModifier.CTRL)	Used to copy text selected	The name of the image e.g.: "img = name.png"
TextPaste <task>	Paste(img, Env.getClipboard())	Used to paste to a specific field (*img*) a copied text	The name of the image e.g.: "img = name.png"

Table 2. Waiting functions

Rule	Sikuli function	Description	Value of the *Description* field
WaitAppear <task>	Wait(img, 10)	Used to wait 10 seconds until an image (*img*) appears on the screen	Name of the image: "img = nome.png"
WaitT <task>	Wait(time)	Used to wait a specified time (t*ime*)	Waiting time, e.g. : "10"
WaitDisappear <task>	waitVanish(img)	Used to wait until something disappear from the screen	The name of the image: "img = nome.png"

Table 3. Modal window functions

Rule	Sikuli function	Description	Value of the *Description* field
Popup <task>	Popup(msg)	Used to show a modal window and specify it content (msg)	The information to show, e.g.: "Invalid operation"
PopAsk <task>	popAsk(msg)	Used to show a modal window with a question (msg) with Yes/No answer	The question to ask. "message"

Table 4. While cicle

Rule	Sikuli function	Description	Value of the *Description* field
FindW <task>	While not exists (img)	Used to inspect the screen until the identification of an element	The name of the image: "imag = nome.png"

Table 5. Conditional expression (if/else)

Rule	Sikuli function	Description	Value of the *Description* field
YesNo_IF <task>	If(text):	Used to verify if the answer (using the popAsk function) of the user is true or false	The name of the Boolean variable: text
ShowM <task>	If exists(img):	Used to verify if an element (img) is visible in the screen	The name of the image: "img = nome.png"
_IF <task>	-	Used to indicate the task to-execute if the condition is true	
_LastIF <task>	-	Used to indicate the last task-to be executed within the condition	
_firstELSE <task>	Else:	Used to indicate the task to-execute if the condition is false	
_ELSE <task>	-	Used to indicate the task to be-executed if the condition is false	
_LastELSE <task>	-	Used to indicate the last task-to be executed within the condition	

Table 6. Functions to open/close an application

Rule	Sikuli function	Description	Value of the *Description* field
OpenApp <task>	App.open(path)	Open an application from a provided location (path)	The path: Path
CloseApp <task>	App.close(path)	Close an application from a provided path	The path: Path
FocusApp <task>	App.focus(title)	Focus an application by indication the title of it	Title of the application to focus: title

The application of the described rules enables the enrichment of the task models for the purpose of the automatic Sikuli script creation. After the enrichment of the task model the second phase of the script creation process (i.e. scenario selection) can start. In the following subsection the description of the second phase of the process is described.

B. Scenario Selection

From a task model, alternative sequences of steps might be executed to perform a task. There is consequently a need to identify the sequence to be used in the automation. Two methods were identified to solve this issue. The first is based on the use of a functionality of the CTTE, i.e. the generation of Presentation Task Set List (can be seen as a finite state machine) from the task model. The second is based on a functionality of MARIAE (simulation of task execution) that enables the simulation of task models and the selection of scenarios (see Figure 2). As the first method was rejected for technical reasons (inconsistency of the system states generated) the second method was the one selected. This method consists of selecting the scenarios by performing a manual simulation of the task model to be automated. The simulation generates an XML file with the ordered sequence of steps accomplished. An example of a generated XML file (for the task of finding a word in the Notepad) is presented in the Figure 3. This file together with the enriched task model is the input required for the task automation to take shape via the execution of the script created.

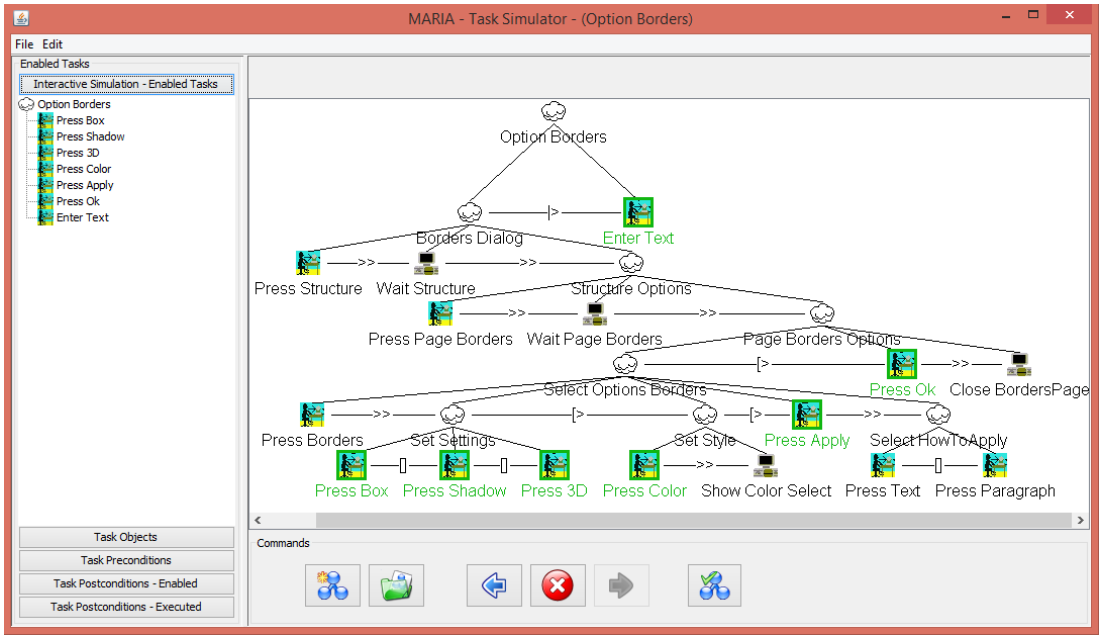

Figure 2 MARIAE (task simulator interface)

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<scenarioType xmlns="http://giove.isti.cnr.it/ctt/scenario" modelId="Option Find">
    <step executed_task="Press Edit">
        <completed_task name="Press Edit" level="2"/>
    </step>
    <step executed_task="Press Find">
        <completed_task name="Press Find" level="2"/>
    </step>
    <step executed_task="EnterSemiAuto wordFind">
        <completed_task name="Set Options Find" level="3"/>
        <completed_task name="EnterSemiAuto wordFind" level="4"/>
    </step>
    <step executed_task="Press findNext">
        <completed_task name="Press findNext" level="3"/>
    </step>
    <step executed_task="ShowM MsgAckCantFindWord">
        <completed_task name="ShowM MsgAckCantFindWord" level="5"/>
    </step>
    <step executed_task="Press Ok MsgAckCantFindWord">
        <completed_task name="Process ModalWindow" level="5"/>
        <completed_task name="Press Ok MsgAckCantFindWord" level="6"/>
    </step>
    <step executed_task="Close MsgAckCantFindWord">
        <completed_task name="Scan Find Attributes" level="2"/>
        <completed_task name="Check Replace Result" level="3"/>
        <completed_task name="MsgAckCantFindWord" level="4"/>
        <completed_task name="Close MsgAckCantFindWord" level="5"/>
    </step>
    <step executed_task="Press Cancel Replace">
        <completed_task name="Finish Replace" level="2"/>
        <completed_task name="Press Cancel Replace" level="3"/>
    </step>
    <step executed_task="Close Replace">
        <completed_task name="Option Find" level="0"/>
        <completed_task name="Start Find Dialog" level="1"/>
        <completed_task name="Close Replace" level="2"/>
    </step>
</scenarioType>
```

Figure 3 MARIE task execution simulation (generated XML file)

C. Sikuli Script Creation Process

As stated previously the process requires an enriched task model together with the scenario of the task to be automated. The process is comprised of the following phases:

1. Extraction of relevant task names and associated images from the task model;

2. Extraction, from the scenario, of the order of steps required to execute the task;
3. Automatic creation of the Sikuli script based on the extracted information from phases 1 and 2.

Since both task model and the scenario are represented by XML files, a library (i.e. XPath Parser) was used to extract the relevant information. After this step an algorithm was developed to manipulate the extracted information and generate the desired script.

Before the development of the algorithm the information present in both files was analyzed and the useful one for our goal identified. Consequently, from the task model (first file) only the elements *Name* and *Description* (see Figure 4) were considered. As explained in the *Task model enrichment* subsection the content of the field <Name> will produce the respective Sikuli function. Alternatively, the field <Description> can have different types of data which will produce different results depending of the rule used.

```
<Task Identifier="OpenApp Skype" Category="application" Iterative="false" >
    <Name>OpenApp Skype</Name>
    <Description>C:/Program Files (x86)/Skype/Phone/Skype</Description>
    <TemporalOperator name="SequentialEnabling"/>
    <Parent name="Skype makeCall"/>
    <SiblingRight name="WaitT SkypeWindow"/>
</Task>
```

Figure 4. Excerpt of a file representing an enriched task model

In the second file (see Figure 5), containing the sequence of executed steps to complete the chosen task, the element required was the <step> element (in particular its attribute "executed_task"). This attribute contains the name of each step and should be compared against the names present in the task model to verify their compatibility.

```
<step executed_task="Enter ContactName">
    <completed_task name="Enter ContactName" level="2"/>
</step>
<step executed_task="Press Contact">
    <completed_task name="Option Find" level="1"/>
    <completed_task name="Press Contact" level="2"/>
</step>
```

Figure 5. Excerpt of a file representing a scenario

To put in practice the stated analysis and proceed with the extraction of the relevant information present in the XML files an approach based on the XPath Parser and JAVA was used. The XPath (XML Path Language) enables the extraction of information present in a XML file. For this purpose, a set of XPath expressions were used.

After getting the required data for the Sikuli script creation, an algorithm able to manipulate this information and generate the Sikuli script was implemented. Firstly, the algorithm compares the names of the tasks model against the names of the scenario and save them in the order the tasks are executed. Secondly, an iterative reading of the names is made to i) verify which of the rules is being used and ii) obtain the corresponding Sikuli functions. This process is used to create the script. Additionally, some lines of code which are in common for all scripts (e.g. error treatments) are also automatically added.

In Figure 6, an overview of the generic process applied in the context of a small example is presented. An automation script (i.e. Sikuli script) is the result of the process (right side of the figure). A partial description of the concrete CTT model was used and the resulting script is described in the following section.

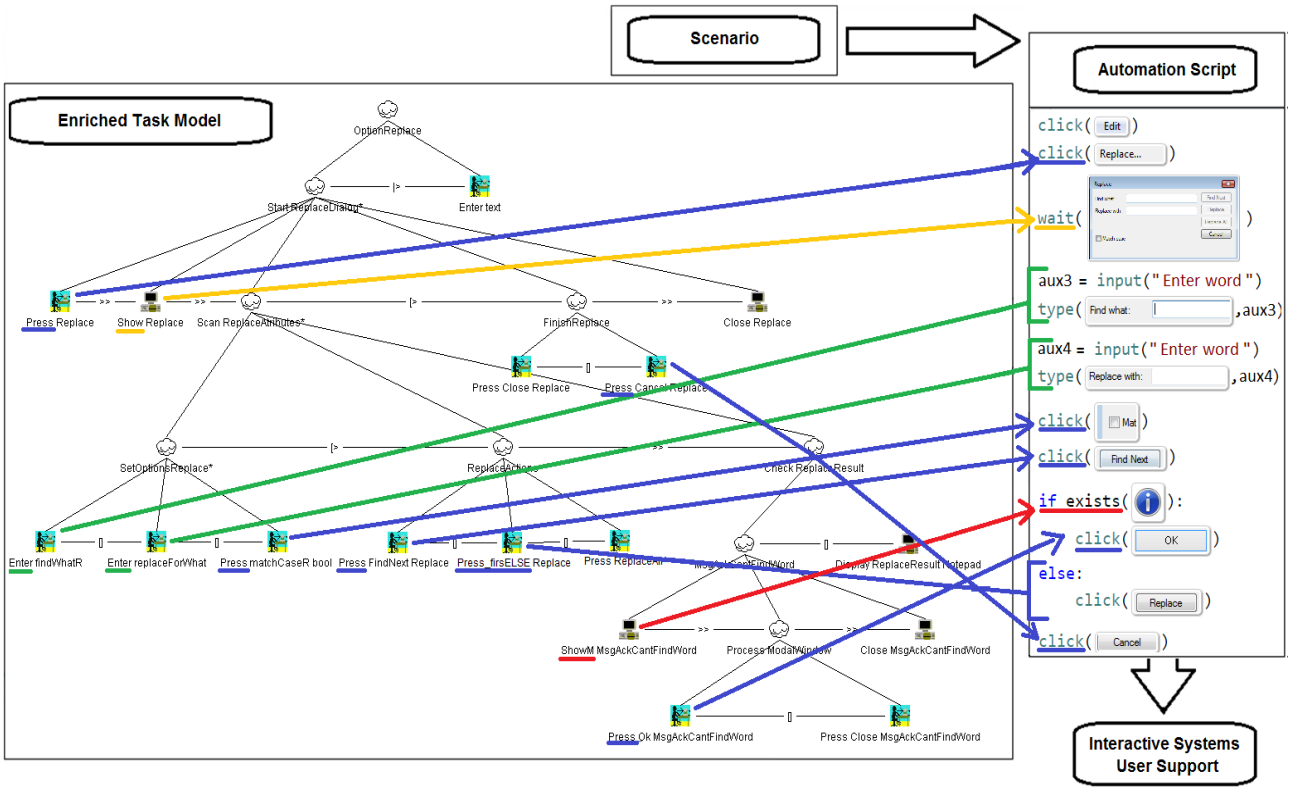

Figure 6. Overview of the script creation process applied to the Replace task of the Notepad example

ILUSTRATION OF THE APPROACH – THE NOTEPAD EXAMPLE

This section is intended to illustrate the application of the approach with a small example to facilitate the understanding of the process. The Notepad editor is used world-wide and is relatively easy to use for the vast majority of users. However, newcomers to Information Technology systems might find it difficult to accomplish some tasks, mainly because they have not done something similar in their previous experience. The replace task was the one used to illustrate the application of the approach and understanding of the process.

A. The Replace Task

The CTT model of Figure 6 represents the steps that must be performed to accomplish the task of replacing words in the Notepad. A brief explanation of the task modeled is described by considering a part (see Figure 7) of the whole CTT model of Figure 6.

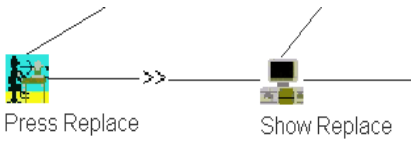

Figure 6. task of the Replace task model

An Interaction Task (*Press Replace*) that represents the interaction of the user with the application (the user has to press the *Replace* button) and an Application Task (*Show Replace*), representing the realization of a task by the application (the application shows the result of interaction made by the user), are represented. These tasks are connected by a CTT *sequential enabling* temporal operator (>>) meaning that the task on the right of the operator starts when the execution of the task on the left is finished. This is represented in the resulting script (see right side of Figure 6) with a sequence of two commands (*Click* and *Wait*). The *Click* command performs a click on the element represented by the image passed as argument. The *Wait* command waits for the image passed as argument to appear. In this case the image is the Replace modal window (see right side of Figure 6) that appears as a consequence of the user click on the *Replace* button. Other operators and types of task are used in the task model. These are translated to the script in a similar way.

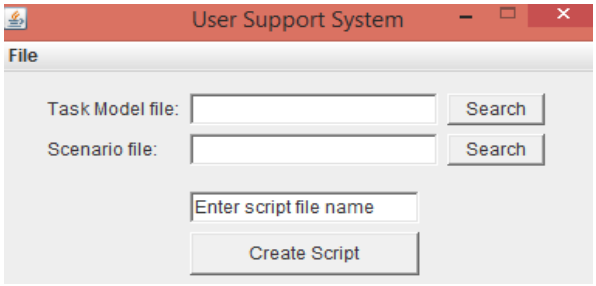

Figure 7. USS tool GUI

B. USS Graphical User Interface

The GUI of the USS tool is presented in Figure 8. It has two text boxes that must be filled with the paths to the enriched task model and scenario files respectively. Then on pressing the *Create Script* button, the Sikuli script is generated (with the name specified in the respective textbox). The generated script can then be used (just by executing it) for the task automation of any interactive system. Developers can use the USS tool to create the scripts associated with the widgets of a simplified version of any (or set of) existent GUIs. A discussion about the use of USS is covered in the discussion section.

C. cript Execution

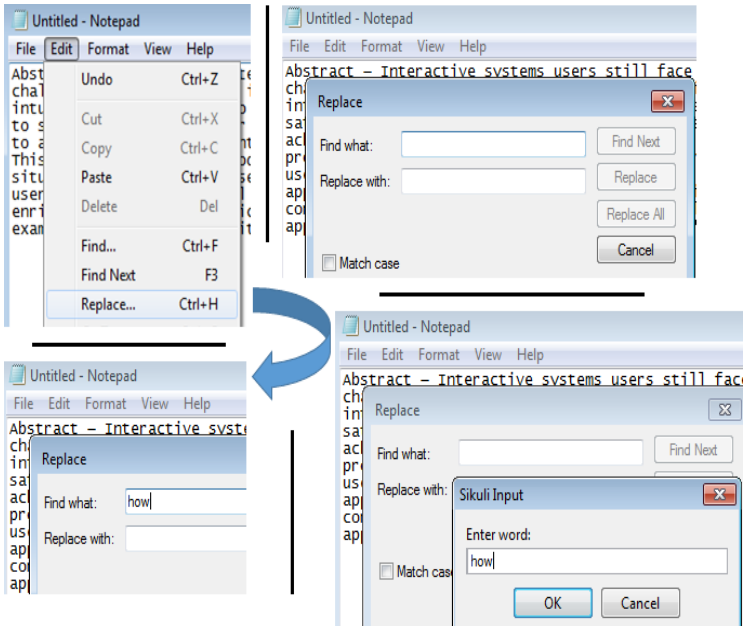

Figure 9. Initial steps of the script execution

Once Sikuli scripts have been generated, task automation can be performed. Figure 9 illustrates the initial execution of the script automatically generated in this example. The execution of the first two *click* commands of the script (see right side of Figure 6) corresponds to the steps of the first image (upper left hand side) of the process represented in Figure 9. The *wait* command matches the second image (upper right hand side). The instruction *aux3 = input("Enter word")* corresponds to the third image where Sikuli asks the user about the word to be replaced. After the introduction of the word by the user (*how* in this example – see last image of Figure 9) the next Sikuli script instruction is executed (*type*) with the word provided as a parameter, as well as a screenshot of the location where the input has to be introduced (Figure 6). This means that the word to be replaced is inserted in the replace modal window. The execution of the script continues following a similar process resulting in the interpretation of the script instructions. At the end, the task is automatically executed. However, as the execution of this task depends on concrete values provided by the user (the word to be found and the word to be used in the substitution), the user must provide them during the execution. Nevertheless, the advantages of the approach remain as the system adapts to the user needs, helping them to complete the tasks. The example used is very simple, but we believe it facilitate the understanding of the process. In the next section a case study and it evaluation are used to demonstrate the benefits of the approach.

CASE STUDY AND EVALUATION RESULTS

In the previous section a small example was presented mainly to facilitate the understanding of the process. This section aims to present the benefits of this approach by means of a case study and its evaluation. Indeed, Blender (http://www.blender.org), a 3D computer graphics software product used for creating animated films, visual effects, art, 3D printed models, interactive 3D applications and video games was used for this purpose. As happened to us, other Blender's beginners might have difficulties even to realize the most basic tasks (e.g. to select and move an object). To perform the tasks, besides having consulted the online software manual, some obstacles remained (e.g. to locate elements in the interface) that made them difficult to execute. Based in this situation, a Sikuli script was created using the USS tool. The use of the script aims to improve the user learning process and efficiency to accomplish a task. The automatically generated script was then added to a developed Support System (Ornelas, J.D. et al., 2016a) that help users to learn how to perform a task by watching it being automatically executed. As a side note, the generated scripts are also used in the ISI tool (Silva, José Luís et al., 2016), enabling the creation of a new UI abstraction aiming at simplified user interaction.

In Figure 10 both Support System in the left hand side and Blender in the right hand side are presented to the users. To perform or to learn how to perform a task, the user just has to select it from the Support System (double click on the respective item) and the associated Sikuli script is then automatically executed. Therefore, this helps users both in efficiency (the task is automatically executed) and learning (the user automatically learn how to perform the task by watching it being executed).

In the evaluation, the task the users were asked to perform was to apply the cylindrical shape to a created Path (see Figure 11). When the user asks for help for the execution of the task he/she is requested about whether or not the Path is already created. If the Path is already created, a popup window asks the user to select the object and the remaining steps are automatically executed. Otherwise a popup windows indicates that the Path will be automatically created and ask the user to select the object for the automatic task execution of the rest.

We believe that the advantages of the proposed approach are made clearer with this case study as the system adapts to the user needs, helping them to complete tasks and learn how to perform them. To sustain this claim results of a preliminary evaluation with users are presented next.

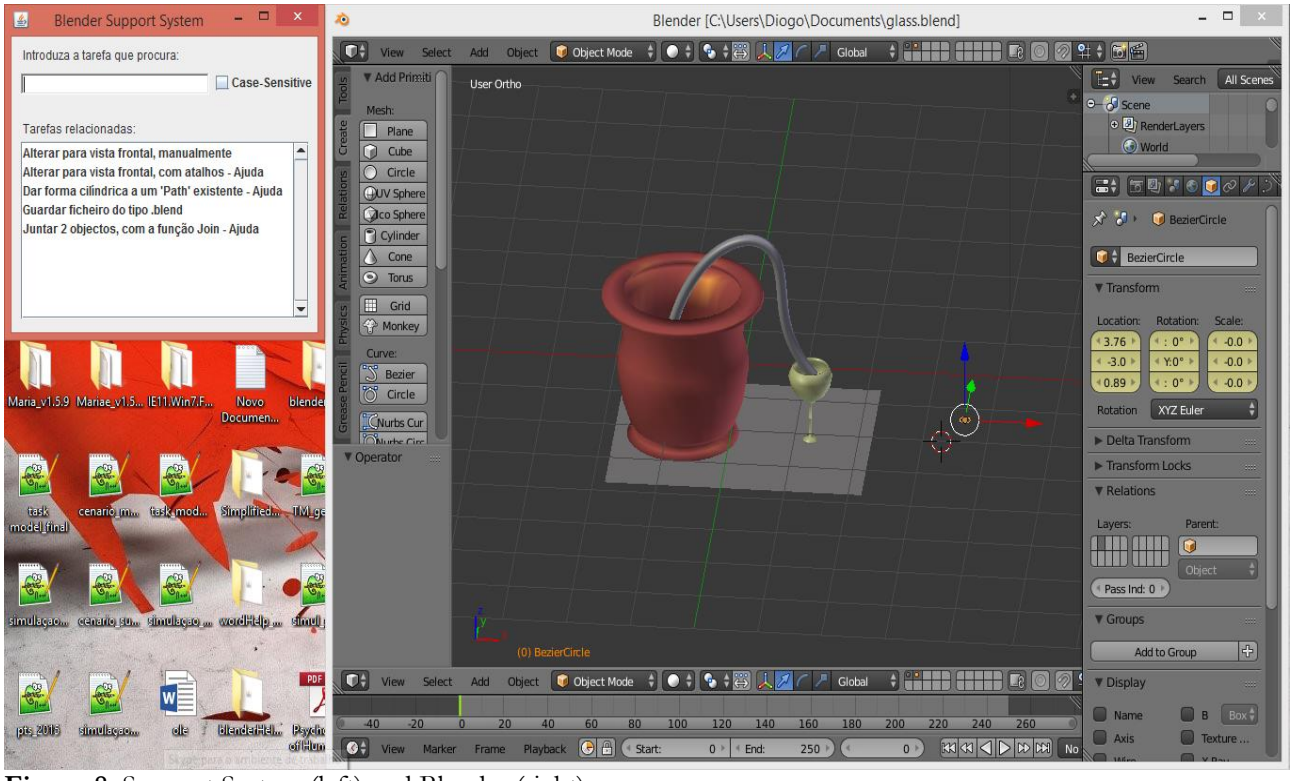

Figure 8. Support System (left) and Blender (right)

Seven persons participated in the preliminary evaluation. Their age varied between 19 and 26 years, four of them were male and two of the participants did not know the Blender software at all. Some instructions were given to the participants (e.g. the task to be performed – as stated above) and after performing the task the participants

were asked to answer a set of questions. The questionnaire (USS Questionnaire, 2016) filled in by participants after the exercise addressed five aspects (as defined in the standard USE questionnaire (Lund, A.M., 2001)): participant characterization; usefulness; ease of use; ease of learning; and user satisfaction. Subjects were asked to answer on a 5 point Likert scale with values from 0 (strong disagree) to 5 (strong agree). The questionnaire included open questions on the tool's strong and weak points, and enabled the participants to make any further comments they wished.

In general, the questionnaires indicated a positive reaction to the approach, with all criteria obtaining a mode of 3 or more. Participants found it useful (mode of 5), making them being more efficient (mode of 4), making them to save time (mode of 3 and 4), easy to use (mode of 5) and are satisfied with it (mode of 4).

Overall the approach was found to provide results that met their goals in the proposed exercise. Beginners seems to learn how to perform tasks more easily and advanced users seems to be more efficient.

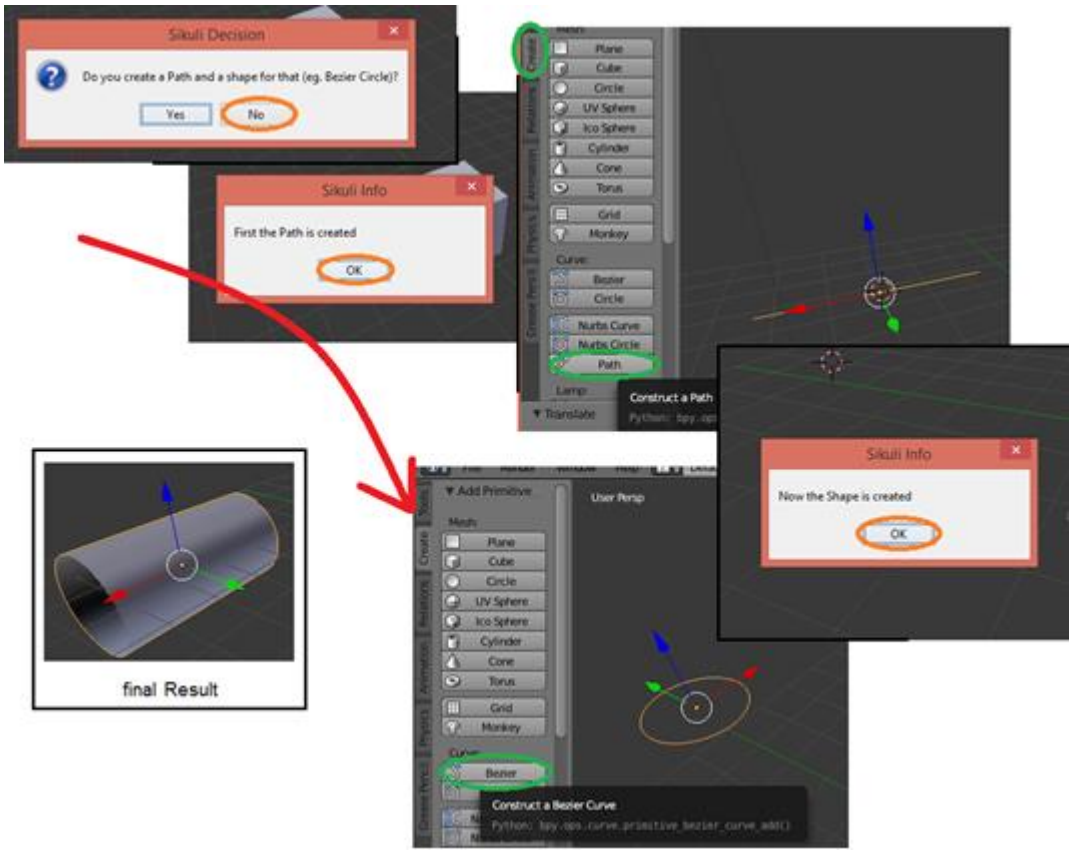

Figure 9 – Partial scenario of the script execution (Blender's task)

DISCUSSION

The decision about which task or which part of a task will be automated is the responsibility/decision of the USS users (i.e. the developers). This decision will obviously influence the end user. It should be noted that the execution of the generated scripts by end users of an application should be triggered by more descriptive widgets (as happen in the Support System used in the case study presented) to avoid repeating part of the problems with the original GUI.

An aspect that influences the use of the tool is the use of task models. When the users of the USS tool are developers of the system to be enhanced, they usually have access to the task models (developed in the early phases of the development). Consequently, they have only to enrich them by following the rules, which are quite straightforward. However, the development of task models that reflect the current behavior of the system might be necessary when access to a developed version is not possible. When required this step might be time-consuming, but the alternative would be to develop concrete Sikuli scripts manually for each task, which in the long term would be even more time-consuming.

This approach serves mainly as a basis for assisting beginners, without (or with little) knowledge about a GUI, to perform a task. They do not need to know which button has to be pressed, textbox to be filled or the valid sequence of steps required to accomplish a task. The case study used supports the use of Blender as the work and previous knowledge required by users to accomplish the task was significantly reduced.

Some insights about the viability of the approach were obtained with the preliminary evaluation however, further evaluations should be made with a larger sample to better understand its applicability and support the stated claims.

Some applications and Operating Systems (OS) offer the possibility to create Macros which also aims to the automation of task execution (generally used for the execution of repetitive tasks). In comparison with our approach, the use of Macros do not take control of the mouse/keyboard enabling users to be (in theory) more efficient. However, when the task does not produce any feedback it might be more difficult for the user to know when it is already done. Due to the fact the execution of the tasks is visible in our approach, an advantage is that it can be used as a learning tool teaching the user how to perform the task. Additionally, all Macros run only on a specific application or OS. The presented approach, besides being illustrated only with one single application, it is (application/platform)-independent. This means that the automatically executed tasks can be inter-applications and inter-OS (without need for any modification in USS tool). Finally, Macros cannot be parameterized (several scripts have to be created) which represent a disadvantage in relation to our approach.

CONCLUSIONS

The current paper extends our previous paper (Ornelas, J. D. et al., 2016b) by describing the approach in more detail and by illustrating it by means of a more complex case study. Additionally, an evaluation and corresponding results are presented and discussed.

The use of interactive systems by beginners is often difficult (O. Al-Shara and A. Dix., 2004). Improvements in usability, design and help systems are trying to address this issue but some problems still remain. This paper presents the USS tool by means of a case study and illustrates its value by presenting the results of a preliminary evaluation with users. The results of the evaluation provide some insights in line with the tool's goal which aims to be the basis for the adaptation of interactive systems to users through automation. The process used was described in detail and results were presented and discussed. Besides some current limitations planned to be solved, this approach seems to be potentially helpful for the use of interactive systems. Further evaluations of the tool with additional end users but also developers are planned as future work to assess those insights.

ACKNOWLEDGMENTS

This work is funded by *Fundacão para a Ciência e a Tecnologia* (FCT) - UID/EEA/50009/2013. Thanks to the participants of the evaluation made.

REFERENCES

Automate testing of web applications. (2016). Available at: http://www.cogitek.com/riatest.html. (Accessed 5 October 2016).

Bergman, L., V. Castelli, T. Lau, and D. Oblinger. (2005). DocWizards: a system for authoring follow-me documentation wizards. Proc. UIST'05, 191-200.

Célia Martinie, Philippe Palanque, David Navarre, Marco Winckler, and Erwann Poupart. (2011a). Model-based training: an approach supporting operability of critical interactive systems. In Proceedings of the 3rd ACM SIGCHI symposium on Engineering interactive computing systems (EICS '11). ACM, New York, NY, USA, 53-62.

Célia Martinie, Philippe Palanque, Eric Barboni, Martina Ragosta. (2011b). Task-Model Based Assessment of Automation Levels: Application to Space Ground Segments. IEEE International Conference on Systems, Man and Cybernetics, Anchorage. IEEE Computer Society - Conference Publishing Services.

Chang, Tsung-Hsiang, Tom Yeh, and Robert C Miller. (2010). GUI Testing Using Computer Vision. Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, 1535-1544.

Drummond, Jon. (2009). Understanding interactive systems. Organised Sound 14.02, 124-133.

eggPlant range, Test automation tools. (2016). Available at: http://www.testplant.com/eggplant/. (Accessed 5 October 2016).

Grudin, Jonathan. (1991). Interactive systems: Bridging the gaps between developers and users. Computer 4. 59-69.

Guy A. Boy. (1998). Cognitive function analysis for human-centered automation of safety-critical systems. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '98), Clare-Marie

Karat, Arnold Lund, Joëlle Coutaz, and John Karat (Eds.). ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 265-272.

Harrison, S. M. (1995). A comparison of still, animated, or nonillustrated on-line help with written or spoken instructions in a graphical user interface. Proc. CHI'95, 82-89.

James R. Eagan, Michel Beaudouin-Lafon, and Wendy E. Mackay. (2011). Cracking the cocoa nut: user interface programming at runtime. In Proceedings of the 24th annual ACM symposium on User interface software and technology (UIST '11). ACM, New York, NY, USA, 225-234.

Jenifer Tidwell. (2005). Designing Interfaces: Patterns for Effective Interaction Design. O'Reilly Media, Inc.

Knabe, K. (1995). Apple guide: a case study in user-aided design of online help. Proc. CHI'95 286-287.

Kourousias, George, and Silvio Bonfiglio. (2010). Picture-Driven Computing In Assistive Technology And Accessibility Design. 1st International ÆGIS Conference, 210-218.

Li, Jiao and Liying, Feng and Qing, Xue and Shi, Zhang and Xu Yiliu, (2010). Interface Generation Technology Based on Concur Task Tree", International Conference on Information, Networking and Automation (ICINA), 350-354.

Lund, A.M. (2001). Measuring Usability with the USE Questionnaire. STC Usability SIG Newslett. 8(2).

Next generation GUI automation. (2016). Available at: http://www.getautoma.com/. (Accessed 5 October 2016).

O. Al-Shara and A. Dix. (2004). Graphical user interface development enhancer (guide).

Ornelas, J.D., Silva, J.C. and Silva, José Luís. (2016a). Demonstration-based Help for Interactive Systems. In the Proceedings of the 2nd International ACM Conference in HCI and UX, 2016, ACM, 125-128.

Ornelas, J. D., Silva, J.C. and Silva, José Luís. (2016b). USS: User support system. In the 11th Iberian Conference on Information Systems and Technologies (CISTI). 1-6, AISTI.

Philippe A. Palanque, Rémi Bastide, Louis Dourte. (1993). Contextual Help for Free with Formal Dialogue Design. HCI (2) 615-620.

Phillips, Betsy, and Hongxin Zhao. (1993). Predictors of assistive technology abandonment. Assistive Technology 5.1, 36-45.

Piyawadee Sukaviriya and James D. Foley. (1990). Coupling a UI framework with automatic generation of context-sensitive animated help. In Proceedings of the 3rd annual ACM SIGGRAPH symposium on User interface software and technology (UIST '90). ACM, New York, NY, USA, 152-166.

R. Parasuraman, T. B. Sheridan, and C. D. Wickens. (2000). A model for types and levels of human interaction with automation. Trans. Sys. Man Cyber. Part A 30, 3, 286-297.

Racim Fahssi, Célia Martinie, Philippe Palanque. (2014). HAMSTERS: un environnement d'édition et de simulation de modèles de tâches (Démo). Interaction Homme-Machine (IHM). ACM DL.

S. Pangoli and F. Paternó. (1995). Automatic generation of task-oriented help. In Proceedings of the 8th annual ACM symposium on User interface and software technology (UIST '95). ACM, New York, NY, USA, 181-187.

Silva, J. C., and J. L. Silva. (2014). A Methodology for GUI Layer Redefinition through Virtualization and Computer Vision, Computational Science and Its Applications (ICCSA), 2014 14th International Conference on, IEEE, 58-63.

Silva, José Luís, Ornelas, Jorge Diogo and Silva, João Carlos. (2016). Make it ISI: interactive systems integration tool. In Proceedings of the 8th ACM SIGCHI Symposium on Engineering Interactive Computing Systems, ACM, 245—250.

Tom Yeh, Tsung-Hsiang Chang, Bo Xie, Greg Walsh, Ivan Watkins, Krist Wongsuphasawat, Man Huang, Larry S. Davis, and Benjamin B. Bederson. (2011). Creating contextual help for GUIs using screenshots. In Proceedings of the 24th annual ACM symposium on User interface software and technology (UIST '11). ACM, New York, NY, USA, 145-154.

Tom Yeh and Tsung-hsiang Chang and Robert C. Miller. (2009). Sikuli: Using GUI Screenshots for Search and Automation. In Proceedings of the 22nd annual ACM symposium on User interface software and technology (UIST '09). ACM, 183-192.

USS Questionnaire. (2016). Available for download here: https://drive.google.com/file/d/0BwUAmY8OB1EUZ1V5MC1Na2xpQjA/view?usp=sharing