Generation of Task Models from Observed Usage Application to Web Browsing Assistance

Karim Sehaba[1][a] and Benoît Encelle[2][b]
[1]*Université de Lyon, CNRS, Université de Lyon 2, LIRIS, UMR5205, France*
[2]*Université de Lyon, CNRS, Université de Lyon 1, LIRIS, UMR5205, France*

Keywords: Knowledge Extraction, Interaction Traces, Task Model, ConcurTaskTrees (CTT), Web Browsing Assistance.

Abstract: This work focuses on the extraction of knowledge from observed usage. More specifically, it aims to design a Web browsing assistance system that helps the user in carrying out his browsing task, or the designer in adapting or redesigning his Web application, according to real usage. The proposed approach consists of generating task models from interaction traces, which are then used to perform assistance. The characteristics to be supported by a task metamodel for assistance purposes are first identified and then confronted with the characteristics of existing task metamodels. This first study led us to choose the ConcurTaskTrees (CTT) metamodel. We then developed processes to generate CTT task models from traces. Finally, to validate our approach, we conducted unit testing and validation based on two real web browsing scenarios.

1 INTRODUCTION

This article focuses on knowledge extraction from data coming from observed usage. More precisely, it is part of Web browsing assistance and redesign of Web applications. The question under concern is how to set up a system able to assist the user in carrying out a web browsing task and the designer in the adaptation and/or the redesigning of his Web application.

In most existing assistance systems, the helps provided, and in general the assistance knowledge, are predefined during the design phase and correspond to the uses envisaged by the designers. Nevertheless, it is often difficult to apprehend for a given system, and for a Web application more particularly, the full spectrum of the real uses: users can have very different profiles, the user needs can be in constant evolution, there may have different conditions of use, etc. If tools and methodologies exist to predict certain uses (including requirements analysis, rapid prototyping and assessments in ecological situation), these will remain intended uses and may in some cases not cover / correspond to all real uses. This may be due to several difficulties related to: the analysis of all the contexts of use, the representativeness of the observed sample, the achievement of truly ecological conditions in assessments, etc. Thus, the design of an assistance system with a complete representation of the needs of future users and their evolution is very difficult, if not impossible.

To remedy these difficulties, we propose to base the assistance on the real uses, observed after the installation of the system. More precisely, the approach developed in this paper aims to generate task models based on these real uses. As an input in the task model generation process, we start from interaction traces (i.e. logs). Here, a trace represents the history of the actions of a given user on a Web application. At the output, a task model represents the different possibilities of performing a given task. Formally, a task model is a graphical or textual representation resulting from an analysis process, making it possible to logically describe the activities to be carried out by one or more users to achieve a given objective, such as booking a flight or hotel room.

The task models obtained by the proposed process will then be used, in an assistance system, to guide the users in the accomplishment of their tasks and the

[a] https://orcid.org/0000-0002-6541-1877
[b] https://orcid.org/ 0000-0002-0734-6480

Sehaba, K. and Encelle, B.
Generation of Task Models from Observed Usage Application to Web Browsing Assistance.
DOI: 10.5220/0008068300740082
In *Proceedings of the 11th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (IC3K 2019)*, pages 74-82
ISBN: 978-989-758-382-7

designers in the analysis of their applications, or even to carry out possible redesigns of them.

In this paper, we begin by identifying the characteristics that should be supported by task models so that they can be used for assistance purposes. We thus identified two main properties, namely:

1. The intelligibility of the task model for the user; and
2. The expressiveness of the task model and its ability to be manipulated by a computer system, more specifically an assistance system.

In the literature, several metamodels have been proposed to represent task models. Therefore, we confront these metamodels with the characteristics previously identified to determine the most suitable for our purpose of assistance. This study leads us to choose the ConcurTaskTrees (CTT) metamodel. Then, we develop a process for generating task models, representing real uses, based on interaction traces.

In summary, the work presented in this paper aims at three contributions:

1. The specification of the characteristics of the task metamodels for assistance purposes;
2. The confrontation of existing metamodels regarding the characteristics identified: the choice of the CTT metamodel;
3. A process for generating CTT task models from interaction traces.

This article is organized as follows. Section 2 presents a state of the art on web browsing assistance based on traces and task models. Section 3 details the target characteristics of a task model for assistance purposes. Section 4 presents a comparison of existing task models against our target characteristics, leading in the choice of the CTT model that we present next. Section 5 describes our approach for generating task model from interaction traces. Section 6 is dedicated to the validation process and results. Finally, we conclude and state some perspectives in the last section.

2 TASK MODELS GENERATION

The design and/or the generation of task models has been the subject of several studies in the literature. In this work, we are interested in generating task models from traces. In this context, the methods proposed in the literature can be classified into two main approaches, namely: the generation of task models from one task instance (1) and from multiple task instances (2).

2.1 From an Instance to a Task Model

The first approach is to start from one instance (i.e. a particular way of performing a task) to generate a model. In this context, let us mention the CoScripter system (Leshed et al., 2008) which makes it possible to automate web tasks via a scripting language, used among other in the Trailblazer assistant (Bigham, Lau and Nichols, 2009).

```
   1. goto
"http://www.mycompany.com/timecard/"
   2. enter "8" into the "Hours worked"
textbox
   3. click the "Submit" button
   4. click the "Verify" button.
```

Figure 1: Example of a CoScript (from (Bigham, Lau and Nichols, 2009)).

Figure 1 shows an example of a CoScript representing the actions for performing a "book purchase" task on Amazon. As this example shows, a CoScript is very close to a trace and possibly generalizes it to a minimum. Indeed, the scripting language integrates only one element of generalization (with the notion of personal database (Leshed et al., 2008)). Therefore, a CoScript is more an instance than a task model itself. In addition, control structures such as the condition (e.g. if A then B else C) or iteration do not exist in CoScripter (Leshed et al., 2008). Therefore, to generate a task model from a CoScript, the challenges of a generalization process from an instance remain open as pointed out in (Allen et al., 2007).

To answer these challenges, namely the elicitation of a task model from an instance, the approach used in PLOW (Procedural Learning on the Web) (Allen et al., 2007) increases the description of a task instance by knowledge provided during its performance. This knowledge makes it possible to represent the conditions, the iterations, etc. This "expert" knowledge is, within the framework of PLOW, provided by a certain kind of user named demonstrator. The latter teaches the system new task models by providing this expert knowledge orally, while he/she is performing the task. This knowledge is then interpreted using natural language processing technologies. This approach requires 1) that the demonstrator keeps in mind to bring a maximum of knowledge and 2) that the tool is able to correctly interpret this knowledge to modify the model accordingly. Thus, the more a demonstrator brings knowledge and more a model can be generic. Overall,

with respect to this knowledge, the more it is expressed in language similar to that used by the system to model the task, the less likely it is that it will be misinterpreted.

2.2 From Instances to a Task Model

The second approach attempts to eliminate the need of expert knowledge by using multiple instances to generate a task model, as in LiveAction (Amershi et al., 2013). The latter focuses on the identifying and modeling of repetitive tasks, tasks represented using CoScripter scripts. In LiveAction, a task model is generated using a set of CoScripts and Machine Learning techniques. With this kind of approach, task models are represented as finite state automata (FSA) (Amershi et al., 2013)(Mahmud et al., 2009). However, to our knowledge, an assistant based on such automata has not been developed yet.

With this kind of approach, the level of genericity of the models obtained depends on the quantity of instances as well as their quality (variability in particular). It should be noted that, as suggested in (Amershi et al., 2013) and with the aim of quickly reaching satisfactory models, users should be able to add knowledge to generated models by manipulating them directly.

2.3 Synthesis

In summary, the first approach, from instance to model, allows the generation of more specific than generic task models and requires expert knowledge. The second, from instances to model, requires a large number of different instances to produce in sufficiently complete / generic models.

We base our work on models that can be generated from multiple instances to limit the amount of expert knowledge to be provided initially and that are necessary to obtain complete models, sufficiently generic. Then, to minimize the constraint due to the sufficient number and the necessary variability of these instances, we will opt for "user-friendly" task models, i.e. allowing users to 1) understand them easily and 2) intuitively add knowledge by modifying the generated models. These additions of knowledge will have to be made directly on the models by handling them, thus evacuating the risks of no or bad interpretation which can lead to erroneous models (risks existing in PLOW for example).

Therefore, reusing an approach such as the one described in LiveAction seems interesting to us, except that the generated models are represented by finite state automata (FSA). These FSAs are indeed difficult to understand and manipulated by end users:

- no decomposition, hierarchical relationships between the tasks and their subtasks. Hierarchy makes it easier to read and understand a model;
- large number of elements: many states and transitions even for a simple browsing task, which generates difficulty of reading and comprehension.

To solve these problems, we propose to use more user-friendly task metamodels than FSAs. To do this, we will first identify a set of characteristics to be supported by a metamodel dedicated to browsing assistance. Then, we will confront the existing metamodels with the characteristics identified in order to choose one. The metamodels studied were designed to be quickly understandable and manipulable by "novice" (i.e. user-friendly).

3 METAMODEL CHARACTERISTICS

As mentioned above, task models generated from traces must be user-friendly, easily understandable, and even directly manipulable by the user. In addition, these models must also be machine-friendly to be automatically used by an assistance system to guide the user in performing his task. This characteristic implies that a machine can a) be able to identify models that do not conform to the metamodels (i.e. models that can lead to misinterpretations) and b) understand and interpret these models at a certain level. This requires a certain level of formality, a very precise semantics of the elements constituting these metamodels.

To these two main characteristics, we add other secondary ones. The first is related to the set of tasks we want to model and their intrinsic properties. We want to model web browsing tasks, essentially sequential tasks (actions to be carried out one after another) and single-user tasks (collective tasks are not considered). This characteristic therefore corresponds to the expressiveness of the metamodels and more specifically to their ability to represent browsing tasks. In this perspective, elements of a metamodel must make it possible to specify the component of the user interface to be manipulated. Another important thing is that the metamodels must also allow the expression of optionality (to model the fact that a sub-task is optional to carry out a given task), for a sufficiently generic modeling of Web browsing tasks. For example, the modeling of a "ticket reservation"

task must be able to represent optional steps, such as the possibility of entering a discount card id.

The second secondary characteristic is related to the adaptability and extension capabilities of metamodels: these are initially designed to be used in a set of software engineering phases (Balbo, Ozkan and Paris, 2004), but they must be adaptable to meet our assistance goal. However, these adaptations may require complements (e.g. concepts), i.e. new elements added to the metamodels. As a result, metamodels have to be extensible if necessary (extensibility characteristic).

The third is related to the plurality of devices that can be used to browse the Web (smartphone, tablet, computer, etc.) and their respective characteristics (screen sizes, proposed interaction modalities, etc.). Indeed, several variants of user interfaces or process of accomplishing tasks may exist for the same Web application ("responsive" aspect). These variations of the context of use must be able to be supported by the chosen metamodel: the same task has to be described in several ways according to the context of use.

In summary, a "candidate" metamodel that could integrate the assistance process we wish to develop, must have key characteristics (user-friendly and machine-friendly) and secondary ones (expressivity, adaptability/ extensibility, support variations in the context of use). The following section presents a study comparing existing metamodels with the above characteristics.

4 TASK MODELS FOR ASSISTANCE

4.1 Comparison of Metamodels with Target Characteristics

Several comparative studies of the most well-known metamodels have been proposed in the literature (Limbourg and Vanderdonckt, 2004) (Balbo, Ozkan and Paris, 2004) (Jourde, Laurillau and Nigay, 2014), including:

- HTA: Hierarchical Task Analysis,
- MAD: "Méthode Analytique de Description",
- GOMS: Goals, Operators, Methods and Selection rules,
- CTT: Concur Task Trees.

These studies propose an analysis framework to compare the metamodels between them and even to guide the choice of one or more specific metamodels for a given objective. These analyzes are based on a set of characteristics, including those we target (see previous section).

Concerning the user-friendly aspect, the authors of (Balbo, Ozkan and Paris, 2004) refer to it through the "usability axis in communication" and specify in particular that, in relation to textual or formal metamodels, the graphic metamodels are more suitable. The authors of (Paternò, 2004) also approve this position. For example, the highly textual GOMS metamodel is moderately user-friendly (Balbo, Ozkan and Paris, 2004). Being able to break down tasks into sub-tasks (decomposition characteristic (Balbo, Ozkan and Paris, 2004)), how to break down tasks and thus describe the relationships between tasks and sub-tasks are also important. For example, a tree-like representation of tasks / sub-tasks appears intuitive (Paternò, 2001), as in MAD or CTT, and offers several levels of detail, including the ability to unfold/fold branches of the tree. Similarly, the ability of the metamodel to allow the reuse of elements helps to minimize the number of elements present and improve readability.

Concerning the machine-friendly characteristic, the degree of formality of a metamodel is related to what must be generated (Balbo, Ozkan and Paris, 2004): a formal metamodel can be used for the automatic generation of code while a semi-formal model can be used to generate user documentation for example. The authors of (Limbourg and Vanderdonckt, 2004) confirm the need for a certain level of formality of the metamodels to be machine-readable: for example, a plan in HTA described informally (textual descriptions) the logic of execution of the sub-tasks that make up a task, which can lead to interpretation ambiguities. On the contrary, CTT has a set of formal operators, based on the LOTOS language (Bolognesi and Brinksma, 1987), to describe this same logic, which guarantees an unambiguous automatic interpretation.

Regarding the secondary characteristics, concerning the expressivity of the metamodels for Web browsing tasks, in addition to sequentiality and single-user criteria, certain models such as MAD or GOMS do not allow the expression of the optionality (Balbo, Ozkan and Paris, 2004), unlike CTT. In addition, some models, such as HTA for example, are not intended to indicate the user interface components that must be manipulated to perform the tasks / subtasks while others, such as CTT, allow it. We also have to mention that the W3C Working Group

"Model-Based User Interfaces" has chosen CTT to model web tasks[3].

Concerning adaptability/extensibility, graphical metamodels are more easily extensible to express relationships or concepts that were not initially planned (Balbo, Ozkan and Paris, 2004). For example, several extensions have been proposed for models of this type, such as MAD or CTT (e.g. MAD*, CCTT).

Regarding the characteristic "context of use variations support", few metamodels integrate this dimension as underlined by (Limbourg and Vanderdonckt, 2004). Nevertheless, CTT integrates this dimension through the platform concept (Paternò, 2004).

Table 1: Comparison of metamodels with target characteristics.

	User-friendly	Machine friendly	Expressivity	Adaptability/ extensibility	Variations to the context of use
CTT	+	+	+	+	+
HTA	+	-	-	-	-
MAD	+	+	-	+	-
GOMS	-	+	-	-	-

Table 1 gives a summary of our confrontation of metamodels with regard to our target characteristics. It thus appears that the CTT metamodel, among the metamodels studied, is the only one that meets all the characteristics previously identified. The following section provides a short introduction to this model.

4.2 CTT metamodel

A Concur Task Tree (CTT) model exposes a hierarchical structure of tasks as a tree. Each tree node corresponds to a task or a subtask. The node icon identifies the category of the task or subtask:

- cloud: abstract task, decomposable;
- user and keyboard: interaction task;
- computer: task performed by the system.

Logical or temporal relationships between tasks are indicated by operators. For example, the operator "[]" represents the choice and the operator "[] >>" represents the enabling with information operator. In Figure 2, the room booking task can only be done if the user has specified the type of the room he wants to reserve (single or double). Thus, the "Make reservation" sub-task is only activated if the "Select room type" task has been performed.

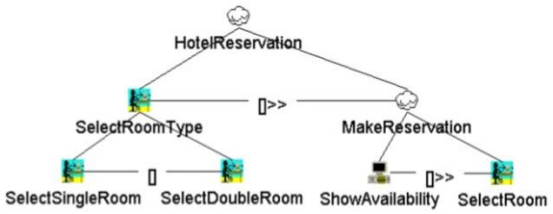

Figure 2: CTT modeling of a room booking task (Paternò, 2004).

These operators are formally defined (mainly from the LOTOS language (Bolognesi and Brinksma, 1987)). CTT allows to add features to tasks as needed: iteration (indefinite or finite) and optionality. Finally, the tasks can be correlated to the application domain objects (the type of room for example) and to their representation(s) on a given user interface (for the selection of a type of room for instance, radio buttons or other).

5 TASK MODEL GENERATION AND ASSISTANCE

Remember that our goal is to propose a task model generation approach based on observed usage. These task models will assist the user in performing their web browsing task and the designer in the adaptation and redesign of their web application. To represent the tasks, we propose to use CTT and, for the observed uses, we rely on the traces / logs resulting from the actions of the user on the Web application.

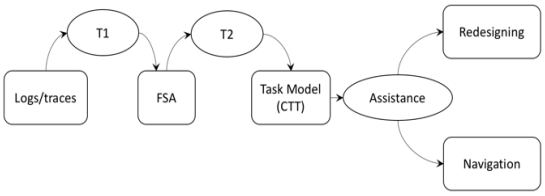

Figure 3: Task model generation process and assistance.

Figure 3 presents the general principle of our approach. At first, the logs / traces are transformed (T1) into finite state automata (FSAs) or, more

[3] https://www.w3.org/TR/task-models/

precisely, deterministic finite state automata (DFSAs). We already mentioned that a bunch of work related to web browsing assistance generate FSAs from the traces, such as LiveAction. If these automata have the advantage of being relatively simple to process automatically (machine-friendly), they remain difficult to understand for the general public given: a) the large number of states and transitions they can contain, even for represent a simple browsing task, and b) their linear reading as there is no hierarchy of states to represent task / subtask relations. Therefore, the second step of our approach is to transform these automata into CTT models (T2).

In this section, we are interested only in 1) the transformation T2 (FSAs-> CTT models), since T1 (Traces-> FSAs) has already been treated by other works, notably (Amershi et al., 2013), as well as to 2) assistance based on task models.

5.1 Task Model Generation from Deterministic Finite State Automata

Formally, a deterministic finite state automaton (DFSA) is a 5-tuple (Q, Σ, R, qi, F) consisting of:

Q: a finite set of states (Web resources/pages)

Σ: an input alphabet, which in our case represents all the events applied to web resources (e.g. button click, typed text, etc.)

R: a part of Q × Σ× Q called the set of transitions

qi: the initial state. qi ∈ Q

F: a part of Q called the set of final states

Since the automaton is deterministic, the relationship R is functional in the following sense:

If (p, a, q) ∈ R and (p, a, a, q') ∈ R then q = q'

In the following subsections we detail the algorithms for converting DFSAs to CTT models, focusing for the example only on CTT enabling and independence operators.

5.1.1 Enabling Operator (“>>”)

The CTT enabling operator indicates that a subtask B cannot start until a subtask A is performed. From the DFSA point of view, if there is an *endState* state for which there is only one transition from a previous *startState* state, we can deduce that there is an enabling operator (see figure 4 for an example of conversion from DFSA to CTT).

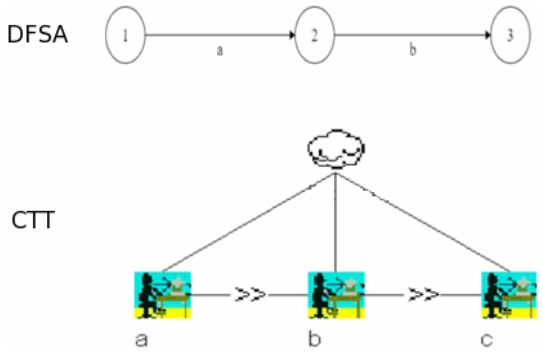

Figure 4: Example of an DFSA to CTT conversion, two enabling operators (between states 1,2 and 2,3).

DFSA to CTT model conversion algorithm (pseudocode) - enabling operator identification:

```
Input: State startState, State endState
Output: Boolean (true if enabling op.
between startState and endState, false
otherwise)

If(!existTransactionBetween(startState,
endState))
    return false
EndIf

For each state s of (AEF.states -
startState) //
    If (existTransactionBetween(s,
endState)
    return false
    EndIf
EndFor
return true
```

5.1.2 Order Independency Operator (“|=|”)

There is an order independency operator between two or more subtasks if they can be performed in any order. From the DFSA point of view, if there are two *startState* and *endState* n paths (n>1) that link them, each path has the same k labels (which will then be in a different order) and k = n!, then we can conclude that these are subtasks that can be performed in any order, expressed in CTT using independence operators (see Figure 5 for an example of conversion).

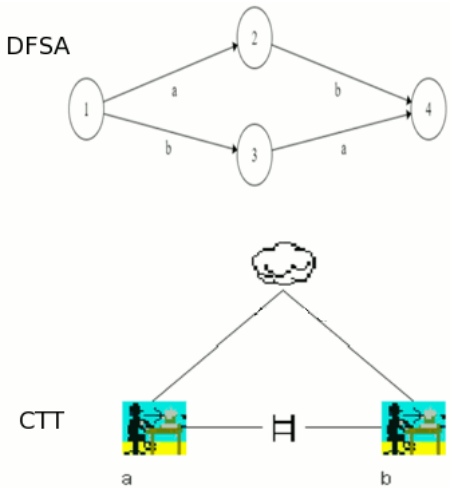

Figure 5: Example of an DFSA to CTT conversion, one order independency operator (between states 1 and 4, two equivalent paths (through state 2, through state 3)).

DFSA to CTT model conversion algorithm (pseudocode) - order independency operator identification:

```
Input: State startState, State endState
Output: Boolean(true if independance
op. exist, false otherwise)
  paths <-
getAllPossiblePaths(startState,
endState)
If (paths.size<=1)
    return false
Endif
  j = paths[0]
if (factorial(j.transitions.size) !=
paths.size)
    return false
Endif
For (i = 1 to paths.size-1)
If(!hasSameTransitionLabels(j,paths[i])
)
       return false
Endif
EndFor
return true
```

5.2 Task Model-based Assistance

Once task models are generated from traces, they will be used to assist:

- the user in his task performance by showing him the actions that must be performed to achieve his objective, or
- the designer in the adaptation of his Web application by highlighting the observed uses of his application.

In both cases, we assume that tasks are organized by categories and that each task model is associated with metadata specifying the name of the task, its purpose and, possibly, the traces that generate it.

User assistance can be provided in three modes: manual search, search through the declaration of the purpose of the task and automatic search. In the first mode, as a classic help, the user is supposed to select the task by navigating in a task tree. The second mode is a keyword search, entered by the user, specifying his objective. It is a question of searching in the task database those whose objective matches the user's keywords. In automatic mode, as the user performs his task, the assistance system displays the tasks that match the actions performed by the user.

Re-design assistance simply consists of displaying tasks that have emerged from usages and that were not initially planned by the designer. This will allow the designer to become aware of these uses and eventually adapt his Web application to these uses through CHI re-design for instance.

6 TESTS AND RESULTS

In order to validate our approach, we implemented our CTT operator identification algorithms from DFSAs, performed unit testing first and then tested our approach on two real scenarios. The Java language was used for coding the algorithms and the DFSAs were stored in XML files.

Unit Testing.
We checked our algorithms for generating CTT operators on 24 XML files, each containing a DFSA that correspond to a particular CTT operator. The generation of CTT operators was successful for 21 out of 24 files, that leads to an average success rate of 87.5% (see Table 2).

Table 2: Results of unit testing. Each row represents a DFSA (an XML file) that corresponds to a CTT operator.

Operator	Nb. of states	Result	Rate
Enabling	5	Success	100%
	7	Success	
	13	Success	
	5	Success	
	3	Success	
Disabling	9	Success	100%
	4	Success	
	3	Success	
Choice	4	Success	100%
	12	Success	
	14	Success	
	48	Success	
Order independency	4	Success	100%
	12	Success	
	14	Success	
	48	Success	
Optionality	6	Failure	50%
	4	Success	
Iteration	5	Failure	33%
	6	Success	
	4	Failure	
Finite iteration	9	Success	100%
	9	Success	
	13	Success	

Real Scenarios Testing.

We have also tested our CTT task model generation algorithms on two real scenarios that represent respectively the booking of a flight on "www.airfrance.fr" and the search for an itinerary on “www.google.fr/maps”. For testing these scenarios and thus evaluating our algorithms, we studied algorithm capacities to correctly identify CTT operators.

The Air France scenario DFSA is composed of 265 states and mainly contains CTT operators that are often encountered in web browsing tasks (choice, order independency, enabling, disabling, iteration). We obtained an average operator identification accuracy rate of 76.58% (see Table 3). Some identification errors are related to each other, especially in the case of the disabling operator which can only take place in an iteration. Therefore, if the identification of an iteration operator that contains a disabling operator fails, then the disabling operator will not be identified either.

For the Google Maps scenario, the DFSA is composed of 64 states and the average operator identification accuracy rate is 71.73% (see Table 4). Most operators are correctly identified except the optional one (i.e. optionality of tasks).

Table 3: Results of the Air France scenario CTT operator identification.

Operator	Instances	Identified instances	Rate
Enabling	104	80	76,92%
Disabling	1	0	0%
Choice	49	38	77,55%
Order independency	3	3	100%
Iteration	1	0	0%
Total	**158**	**121**	**76,58%**

Table 4: Results of the Google maps scenario CTT operator identification.

Operator	Instances	Identified instances	Rate
Enabling	30	21	70%
Choice	13	10	76,92%
Order independency	2	2	100%
Optionality	1	0	0%
Total	**46**	**33**	**71,73%**

7 CONCLUSIONS

This article deals with the extraction of knowledge from data coming from observed usage in the context of Web browsing assistance. Our approach is based on the generation of task models from interaction traces (logs). A task represents all the user's actions performed on a device to achieve a given objective. A trace represents the history of the user's actions on the digital environment. The aim is therefore to consider the traces left by users as sources of knowledge that an assistance system can use to generate user-specific help, and thus overcome the limitations of assistance based on intended uses during the design phase of a system.

From a usage perspective, the task model generated from traces could thus be used automatically or semi-automatically to assist a user in performing his task or a designer in adapting the digital environment to the observed uses.

Our contributions focused on 1) specifying the characteristics of task metamodels for user assistance, 2) comparing existing task metamodels with the identified characteristics, this study allowed us to choose the CTT metamodel, and 3) developing a process for generating CTT task models from traces.

In future work, we plan, as an extension of the work already done, to continue the study of task metamodels - being aware that not all existing task metamodels (and there are many) have been covered. For instance, UML statecharts and the task modeling

part of Web Semantics Design Method (WSDM) (based on CTT) (De Troyer, Casteleyn and Plessers, 2008) deserve to be studied in the light of the identified characteristics. A comparative study of the selected, compliant metamodels, assessing their ability to be quickly understood and manipulated by "novice" - "first time users", could also be carried out to consolidate the choice of a given metamodel.

Concerning Web task model's generation using CTT, an interesting study could be conducted to compare our approach against another one based on the analysis of web sites code (Paganelli and Paterno, 2003), and for investigating to what extend both of them could be used in a complementary manner. To do this, we also have to develop further conversion algorithms to cover all metamodel operators and evaluate these algorithms using Web browsing data. Then, an assistance system using our approach has to be developed and evaluated in an ecological situation.

REFERENCES

Allen, J., Chambers, N., Ferguson, G., Galescu, L., Jung, H., Swift, M., & Taysom, W. (2007, July). Plow: A collaborative task learning agent. In *AAAI* (Vol. 7, pp. 1514-1519).

Amershi, S., Mahmud, J., Nichols, J., Lau, T., & Ruiz, G. A. (2013). LiveAction: Automating web task model generation. *ACM Transactions on Interactive Intelligent Systems (TiiS)*, *3*(3), 14.

Balbo, S., Ozkan, N., & Paris, C. (2004). Choosing the right task-modeling notation: A taxonomy. *The handbook of task analysis for human-computer interaction*, 445-465.

Bigham, J. P., Lau, T., & Nichols, J. (2009, February). Trailblazer: enabling blind users to blaze trails through the web. In *Proceedings of the 14th international conference on Intelligent user interfaces* (pp. 177-186). ACM.

Bolognesi, T., & Brinksma, E. (1987). Introduction to the ISO specification language LOTOS. *Computer Networks and ISDN systems*, *14*(1), 25-59.

De Troyer, O., Casteleyn, S., & Plessers, P. (2008). WSDM: Web semantics design method. In *Web engineering: Modelling and implementing web applications* (pp. 303-351). Springer, London.

Jourde, F., Laurillau, Y., & Nigay, L. (2014). Description of tasks with multi-user multimodal interactive systems: existing notations. *Journal d'Interaction Personne-Système (JIPS)*, *3*(3), 1-33.

Leshed, G., Haber, E. M., Matthews, T., & Lau, T. (2008, April). CoScripter: automating & sharing how-to knowledge in the enterprise. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 1719-1728). ACM.

Limbourg, Q., & Vanderdonckt, J. (2004). Comparing task models for user interface design. *The handbook of task analysis for human-computer interaction*, *6*, 135-154.

Mahmud, J., Borodin, Y., Ramakrishnan, I. V., & Ramakrishnan, C. R. (2009, April). Automated construction of web accessibility models from transaction click-streams. In *Proceedings of the 18th international conference on World wide web* (pp. 871-880). ACM.

Paganelli, L., & Paterno, F. (2003). A tool for creating design models from web site code. *International Journal of Software Engineering and Knowledge Engineering*, *13*(02), 169-189.

Paternò, F. (2001). Task models in interactive software systems. In *Handbook of Software Engineering and Knowledge Engineering: Volume I: Fundamentals* (pp. 817-836)

Paternò, F. (2004). ConcurTaskTrees: an engineered notation for task models. *The handbook of task analysis for human-computer interaction*, 483-503.