Club♣ (Trèfle): a use trace model

Elöd Egyed-Zsigmond, Alain Mille, Yannick Prié

LIRIS – Université Lyon 1
Bâtiment Nautibus,
69622 Villeurbanne CEDEX, France
Firstname.Surname@lisi.univ-lyon1.fr

Abstract. In this paper we present a use trace model which allows the collection and reuse of user experience, based on a homogeneous and interconnected representation of users, procedures and objects. All these notions form a connected labeled directed graph containing highly connected and explained use traces. This model enables assistance in non trivial, creativity requiring situations. Our model uses the Case Based Reasoning (CBR) paradigm in order to reuse experience. After a formal description of the model we discuss how it can serve to capitalize and re-use experience.

1. Introduction

A constantly growing large public is using computers and networked electronic devices for a large variety of tasks. "Generic" computer applications like text processors, development environments, etc. became a support for these tasks by providing the required resources. We mean by resource a service delivering data, knowledge, processing… The large variety of uses of these applications and the diverse backgrounds of their users make illusory the development of software assistants capable to help *a priori* the user in fulfilling his task.

Indeed, generic applications allow carrying out complex tasks, providing a high degree of freedom to users. By definition, therefore, the creators of these applications cannot precisely know which tasks their tool will enable to carry out. Moreover "democratization" of the software tools let occasional users handle concepts and methods developed by and for specialists. Although applications become more and more user friendly, the tasks for which they are used remain complex. These applications are used by users having a heterogeneous degree of expertise. It is important thus that the user interfaces of these applications can adapt, to be able to fit the needs of a particular task for a particular user. As computers "do not forget", it is interesting to keep the suitably modeled traces of use sessions, in order to be able to reuse them to help the realization of new tasks. But, if it should be really useful to provide relevant past use episodes, similar to the current one, it is necessary to take into account the almost infinite possibilities of use of a generic application. We propose a way to manage this complexity.

Let us imagine the case of a word processing application we want to use to write a CV. It is the first time that we have this task and we don't know very well nor what this kind of document should have as content nor how it should look like physically. We start writing a title, specifying our name, age and begin describing our professional experiences and education degrees. Let us imagine now that our text processor is equipped with an experience gathering system and that it has been already used by several other people (it thus collected their experience). In this case, after having observed us typing several lines, the system can realize that we are using the application in a manner similar to that of another episode collected before (probably one concerning the writing of another resume). According to this information, the system will be able to propose this example, to help us succeed with our CV writing. It could also propose us to specify at the beginning of the document our address, telephone number and email, and it could help us building a convenient layout.

Beyond this example, we developed and applied this approach to provide help for audiovisual document annotation and search. Let us consider the annotation of audiovisual documents. In this application user assigns keywords drawn from a vocabulary to document fragments. We developed this application in our research team and presented it in [2,9,10]. At the beginning, this task is *ad hoc* and the user assigns key words to the document fragments in a personal and intuitive way. If the system is able to capitalize and re-use experience, it will be able to guide the annotation so as to lead to more coherent and structured descriptions among different users. From the first dropped key words, the system can find other documents annotated by these same words or similar words and propose to supplement the current annotation by the missing characteristics or to integrate it in one of the existing description schemas.

These two different applications (text processing and document annotation) illustrate two manners of helping users carrying out their tasks. The assistance in these situations is not done on the use of the application's user interface, but on the realization of the tasks needing creativity, like annotation, document creation, etc. The system aims to reduce the complexity of concrete tasks, when using tools with several degrees of freedom. The assistance relates to "What to do?" type problems rather than to "How to do?" type problems. In the case of the text processor, the question is more, what to type, not how to type. In the case of a document annotation system, the major difficulty does not come from the manner of using the tool, but from the variety of document contents, the size and richness of the vocabulary and the complexity of description schemas.

In this paper we present a use trace model which allows the collection and reuse of user experience, based on a homogeneous and interconnected representation of users, procedures and objects. This model enables assistance in non trivial, creativity requiring situations. Our model uses the Case Based Reasoning (CBR) paradigm in order to reuse experience [11]. Other systems [1,7,16] use similar strategies to provide help, but they don't integrate the three components (users, procedures, objects) in one model. After a formal description of the model we discuss how it can serve to capitalize and re-use experience.

2. The basic idea

We consider that in a software application, ***users*** handle ***objects*** using ***procedures***.

According to our model we memorize the object handling traces by representing the use sessions, the procedures and the objects, in a single directed connected graph. Each one of the three notions (object, procedure, user) is presented in this section in general and they are instaciated in the graph as *abstract nodes* (referring to abstractions, object, procedure and user type definitions) and *concrete nodes* (referring to instances of objects, procedures and users).

2.1. The objects

Through the use of a computer program, users can manipulate a set of objects. The *use model* contains the objects and the relations between them we choose to observe. In a text processor, typical objects are the document, a section, a sentence, a word, … The objects can have relations between them, for example a document is composed by sections, but in the general model we will call a relation between two objects in the use model : *explanation relation*. In the actual Club♣ model these *explanation relations* are only *contextualization* relations: A_is_the_context_of_B, B_is_in_the_context_of_A, but we think that other kinds of relation between objects can be included in the use model.

For a text processor, the *use model* can be composed, as illustrated in Fig. 1, of words, phrases, sections, titles, document models and documents.

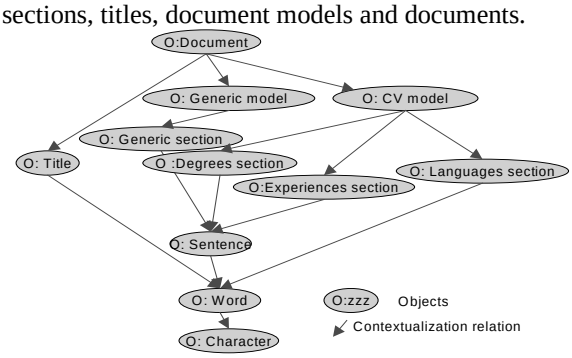

Fig. 1 Example of a *use model* of a text processor application

While observing the use of the application, the objects of the use model are instantiated, instances representing the manipulations carried out on the *concrete objects* through user actions.

2.2. The procedures

If the use model contains everything which can be manipulated in a computer program, the task model would be defined as "everything that make sense" for a given

task. We call *procedure* a manner to use, manipulate objects from the use model. We call them this way in order to emphasize the difference between these procedures, and the task model defined in the knowledge engineering field [3-5,7]. We talk here about a pragmatic point of view on concrete manipulations within a computer program.

Just like it is the case with the relations between objects in the use model, the precise semantic of the relations between procedures is not predefined in the general Club♣ model. We will use the term *contextualization relations* to design these relations, expressing the fact that a procedure is executed in the context of another procedure, or that two procedures represent a decomposition of a complex task in two subtasks.

Procedures are typically actions accessible by the menus, buttons and icons of the graphical user interface or simply typing command line instructions. The procedures available in a same application are seldom entirely independent.

For example in a text processor the creation of a document passes by the choice of a document model ||P: Model choice|| and a suite of section creations ||P: Create section|| according to the model. The Fig. 2 presents two arbitrary decompositions in procedures of this task model.

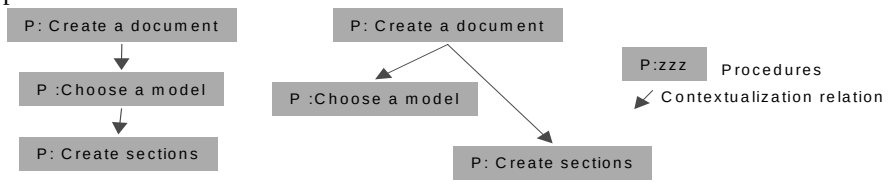

Fig. 2. Two decompositions of a document creation task in a text processor application

We can consider that the section creation procedure is executed in the context of a chosen model, while a model is chosen in the context of a document creation. This way we can link the document creation procedure to the model choice procedure and the model choice procedure to the section creation procedure. We could also consider that the document creation procedure is decomposed in a model choice and section creation procedures. The set of procedures with the relations between them gives us the *procedure model*. A procedure model is a graph the nodes of which represent the procedures and the edges of which represent the *contextualization relations* between them. These procedure models can be expressed using already existing task modeling tools [6,8,14,15].

2.3. Users

A *user* exploits procedures during a use *session*. Our experience reuse and sharing goal gives the *user* a particular importance. It could be a simple use session attribute but we talk here about tracing a session of one specific user. We thus keep the *user* name to represent the notion of use session.

Procedures are launched by *user*s. These users can have different access rights to the procedures. In the Club♣ model we include users we want to follow the actions of,

as nodes of the graph, which are linked to the nodes representing the procedures they have access to. The model enables tracing the actions of the so defined users.

2.4. Application of the Club♣ model to a given computer program

In order to apply the Club♣ model to a given application, we have to define first the set of objects (chosen from the *use model*) we want to trace the manipulation. The set of procedures (from the *procedure model*) permitting the manipulation of the chosen objects has to be created next with the contextualization relations attaching them together. The *users* are defined specifying the constraints on the available procedures.

The procedures chosen to be traced together with the nodes representing the users and the edges of the graph linking these nodes compose the *observation model*. Indeed we define here a sort of *point of view* we whish to follow while following the use of the computer program. This observation model will provide the *explanation* of the traces gathered during the observation of the use of the program. It creates a filter through which the use of the program is observed.

The observation model for a text processor could be composed of the document creation procedure, model choice procedure and section creation procedures. The creation of the document explains the choice of a model and the creation of sections. In fact in order to create a document, one has to choose a model and create sections. In this case the *use model* is composed by object nodes representing the document, the document model and the section. Defined this way, the application of the Club♣ model enables the tracing of the use of a text processor.

3. The Club♣ model formal description

We have decided to include in the Club♣ model all the elements we need to trace the use of a computer program: users, procedures and objects. In our research [9,10,12] we have developed an audiovisual document annotation system (E-AIS : Extended Annotations-Interconnected Strata) based on a graph structured model. We have also designed several tools, like *potential graphs* for querying this structure.

Formally an instance of the Club♣ model M is composed of:

- a global graph G containing the use model subgraph, the observation model subgraph and the trace subgraph;
- a set of *potential graphs* {PG}, comparison functions and adaptation methods enabling the exploitation of the graph G.

The global graph G is defined as $G=<N,R,L,\nu>$, where N is the set of nodes, R is the set of edges. L is the set of edges labels and ν a function which associates to each edge a label. The set N is composed of three types of nodes $N=U\cup P\cup O$. A node can thus be a user $u\in U$, a procedure $p\in P$ or an object $o\in O$. The semantic of the edges is given by the type of the nodes they link.

A node can be abstract or concrete. An *abstract node* is defined when applying the Club♣ model to a specified computer program. The *use* and *observation models* are composed by *abstract nodes*. *Concrete nodes* are created when observing the use of the program, they form the trace sub-graph. A *concrete node* is notated [Type : Name, time_code], where Type is either O(object), P(procedure) or U(user), and the time_code is the moment when the *concrete node* was created (Object created, procedure launched, user session started). The *concrete nodes* can be sorted chronologically using this time_code.

3.1. Object nodes.

Objects are nodes representing all observed entities that the user can manipulate and handles in a conscious way. The objects of the Club♣ model are elements of the *use model*, we decided to trace the handling.

An ***abstract object*** is like the class definition for an object we decided to trace the manipulation of. To each *abstract object* corresponds at least an *abstract procedure* which allows its creation. An *abstract object* can also be in relation with several other *abstract procedures* allowing its handling. Indeed beyond the creation procedures there can be one or more procedures which re-use the object, visualize it or modify it. *Abstract objects* can be linked to other *abstract objects* with *contextualization* relations. An *abstract object* is notated «O:object name». On Fig. 3 we have «O:Document», «O:Generic model », «O:CV model », «O:Section», «O:Degrees section », «O:Experiences section », «O:Languages section», «O:Sentence», «O:Word ».

A ***concrete object*** is an instance of an *abstract object*, it has a type, i.e. it is connected to only one *abstract object* by an *instantiation relation*. Every *concrete object* was created by a *concrete procedure*, on the initiative of a *concrete user*. *Concrete objects* inherit the *contextualization* relations between them from the *abstract objects* they derive from.

3.2. Procedure nodes

We choose to consider that any use of a computer program can be described through ***procedures***. A procedure is included in the graph because we choose to trace its uses. Users have to choose procedures to handle objects. *Procedures* identify specific task signatures, usually connected to the application's graphical interface. A *procedure*, in the Club♣ sense of the word is not necessarily an operation or a task of the application, but a treatment unit chosen to be observed and which enables to handle objects of the use model.

An *abstract procedure* is a *procedure* type. It concerns *abstract objects* (it can be considered as an operator while objects are the operands) and can represent simple or complex user interface functionalities. An *abstract procedure* is notated ||P :procedure name||. The contextualization relations between *abstract procedures* do not represent any constraints. They don't indicate any information neither on the order nor on the

number of instances when using them. They relate only that something has been done *in the context of* one other thing and will be used in similarity assessments.

A ***concrete procedure*** is an instance of an *abstract procedure*, situated in time and linked to a *concrete user*. It is linked as well to a *concrete object*. A *concrete procedure* materializes a highly contextual trace of the launching of an *abstract procedure*. *Concrete procedures* inherit, just like *concrete objects* do, the contextualization relations of their abstract nodes.

3.3. User nodes

A ***user*** is the representation of a user session, that is a role. User nodes are linked to *procedure* nodes by *launched by* relations.

An ***abstract user*** is notated <U : user name> and materializes a registered user. It does not correspond obligatory to a real person. A registered user can log in the program and his actions will be identified thanks to his sessions. These sessions are materialized by *concrete user* nodes. As a concrete node, the ***concrete user*** is tagged with it's time code representing the beginning instance of the session.

In the graph of the Club♣ model, relations are labeled. The label is given by the types of the nodes relations are linking. Table 1 represents these constraints for a relation $r \in R$. Lines of the table give for a given relation: the constraint number of the constraint, the type of the origin node (O-object, P-procedure, U-user), the type of the destination node and the label of the relation ($R_{conetxt}$ - contextualization relation, R_{inst} - instantiation relation, R_{creat} - creation relation, R_{launc} - launch relation).

N°	C1	C2	C3	C4	C5	C6	C7
Start node type	Oa or Oc	Pa or Pc	Oa	Pa	Ua	Pa or Pc	Ua or Uc
End node type	Oa or Oc	Pa or Pc	Oc	Pc	Uc	Oa or Oc	Pa or Pc
$\nu(r)$	$R_{context}$	$R_{context}$	R_{inst}	R_{inst}	R_{inst}	R_{creat}	R_{launch}

Table 1. Constraints on the label of a relation $r \in R$, according to the type of the linked nodes. (Oa, Pa, Ua - abstract nodes, Oc, Pc, Uc - concrete nodes)

4. An example

Fig. 3 represents a concrete example of the global Club♣ graph. For an easier reading we will not represent the direction of the relations, considering that in a figure, an arrow starts always from the upper graph node and goes toward the node below. The observation model allows tracing the document creation according to a particular procedure model. The goal of this example is to illustrate trace construction. The observation model contains only one *user* <U :Jean> and a *procedure* structure which specifies that the document creation procedure (||P:Create a document||) is the context of the document model choice procedure (||P:Choose a model||), in its turn made up of

the section creation procedure (||P:Create a section||), the sentence typing procedure (||P:Type a sentence||), which is finally the context of the word typing procedure (||P:Type a word||). To build the use model we consider the following *objects*: document, generic model, section, sentence and word.

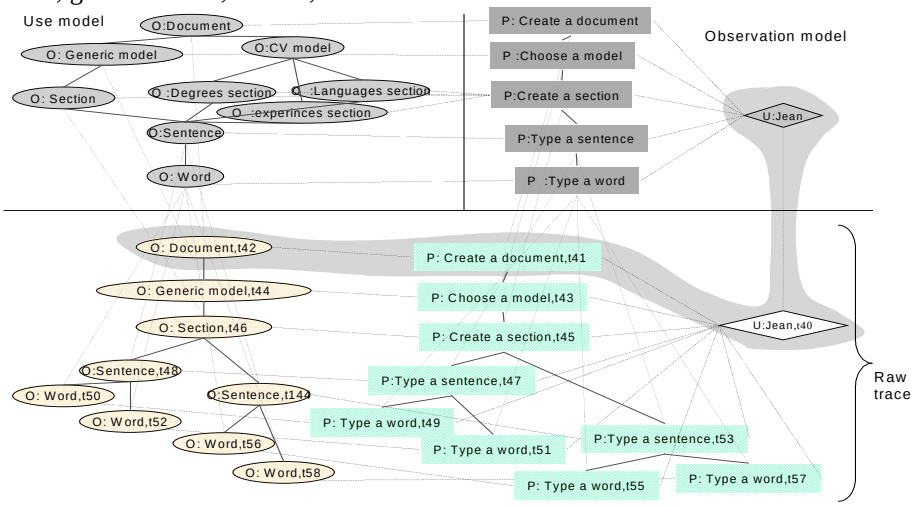

Fig. 3. Global graph representing the Club♣ model applied to a text processor

The example presents only one session along which the user created a document according to the generic model containing a section contextualizing two sentences each of them contextualizing two words. Each operation of the user results in the creation of a concrete node in the graph, tagged with a time code. User uses the procedures of the observation model to handle objects, thus leaving traces made of nodes which are instances of these *abstract procedures* and objects. We can see that the user Jean started a session at the time t40. He choose the document creation procedure, leaving the trace formed by the nodes [P :Create document, t41] and [O :Document, t42]. After this he selected a document model and created a section. Each action gave birth to nodes in the trace. The user used instances of *abstract procedures* to manipulate *concrete objects*. Each node of the trace derives from an abstract node and is in relation with other concrete nodes.

5. Trace construction

After having described formally the graph structure formed by the use and the observation model, we present the construction of traces during the use of a computer program. Let us go on with our text processor example. As presented in Fig. 3 the observation model enables the tracing of document creation. The set of concrete nodes on this figure together with their relations form the ***raw trace***, while the whole global graph can be considered as the ***explained raw trace***. Generally a trace is a connected

sub-graph of the global graph G, containing at least one concrete node. The trace contains the *concrete user* node [U :Jean, t40] representing the fact that a user having the Jean login began a session at the time t40. During this session the user launched first the document creation *procedure*, creating the node [P : create a document, t41] attached to the node representing the session with a *launched relation*, and to the node representing the newly created document [O :Document, t42] with a *creation relation*. All three concrete nodes are linked to abstract nodes they derive from. As a next step the user chooses a document model leaving the trace composed by the nodes [P : Choose a model, t43] and [O :Generic model, t44] and their adjacent edges. The [P : Choose a model, t43] *concrete procedure* is linked to the [P : create a document, t41] by a *contextualization relation*, inherited from their abstract nodes which are connected with a relation having this label. The session ends with the typing of the last word [O :Word, t58]. Every action of the user generates several new nodes and edges in the graph creating this way a highly contextualized trace. These traces are the starting point of our help system. From the trace we can create episodes which will be considered as reusable cases.

6. Potential graphs and episodes

Traces can be processed with the help of *potential graphs* in order to obtain *episodes*. In this section we first introduce the *potential graph* and present after the episode construction methods.

The exploitation of the graph structure is based on a sub-graph-matching algorithm described in [13]. In fact a request is expressed through so-called *potential graph*. These are graphs made up from several nodes partially filled. For example the request: "Find objects manipulated by <U:Jean> and the procedures used to manipulate them" will give the p*otential graph* shown in Fig. 4. A request is solved by searching the matching of this p*otential graph* in the global graph. The potential graph elements assigned with a "*" are unspecified nodes, these are in fact what we try to match. The algorithm starts the sub-isomorphism search with the specified nodes and tries to match the rest of the *potential graph* on the global graph following the node types and edge labels. A *potential graph* has to be connected and has to contain at least one specified node. *Potential graphs* have *named nodes*, nodes having a label, the instances of which can be retrieved. In the Club♣ model the named nodes have labels notated Nt_i (trace node n° i). The isomorphism of this *potential graph* will contain *concrete objects* and *concrete procedures* in place of that specified with "*", and these concrete nodes represent the objects and procedures concerning the user <U:Jean>. We have emphasized one instance of isomorphic sub-graph of the global graph on Fig. 3 by the grayed shape, surrounding the nodes: [Odocument,t42], [P:Create a document,t41], [U:Jean,t40] and <U:Jean>. There are 8 other matching sub-graphs in the global graph, containing each *concrete object* and the related *concrete procedures*.

In the Club♣ model *potential graphs* are used to calculate traces and cut them in episodes. An ***episode*** is an ordered list of nodes in the global graph G, built by the instanciation of a *potential graph*.

The nodes of the raw trace, sorted by their time-code constitute the ***linear raw trace***. This contains every concrete node chronologically ordered. The Fig. 4 presents the *linear raw trace* of the document creation session illustrated on Fig. 3. This trace was obtained by the instantiation of the *potential graph* "linear raw trace" *<U :Jean>* applied to the global graph G. This *potential graph* has two named nodes, indicating that their correspondents in the global graph should be included into the trace. In order to build linear traces, a *potential graph* has to be instanciated, the matching nodes of the global graph to the named nodes of the *potential graph* retrieved and sorted following a total order (in our case the temporal order). In our example the *potential graph* has two named nodes indicating that every *concrete object* and *procedure* corresponding to them has to be included in the trace.

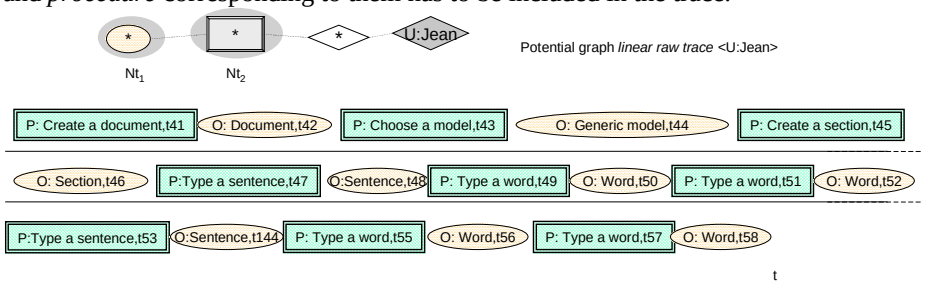

Fig. 4. The *linear raw trace* of the session presented in Fig. 3 and the *potential graph* which served to retrieve it.

The *linear raw trace* can be cut in episodes in different ways. It is possible to retrieve only *procedure*s or only *objects*. Another way is to cut it in episodes representing each a different session. Of course, the global graph has to contain in this case more than one session. We can consider *potential graphs* not containing any contextualization relations. Using these, we can calculate linear traces and episodes on any application of the Club♣ model. Other *potential graphs* can exploit the structure of the use or observation model of a specific program. Indeed the goal of these models is to provide a rich tracing framework. This way we are considering not only the *linear raw trace* but a larger context of each node as well.

Let us consider the use and observation model of a text processor as presented on Fig. 3. It is possible to exploit the contextualization relations between the *abstract procedure*s. Doing this, for this text processor we can create several *potential graphs* in order to retrieve only words, phrases and words, or only phrases, manipulated during one or several sessions. We can consider that a word is a more contextualized object than a sentence which is more contextualized than a section and so on. Fig. 5 represents a set of these *potential graphs*. The first one (PG_words) enables to calculate an episode containing the manipulated words only.

If we instantiate this *potential graph* in the global graph of the Fig. 3 it will retrieve the episode $\text{episode}_{\text{PG\_Words}}$ from the Fig. 6. The second one adds the phrases as well

producing the episode $episode_{PG\_sentences\_and\_words}$ from the Fig. 6. We consider that the first episode is *finer* than the second one because it contains only objects coming from the *bottom* of the use model.

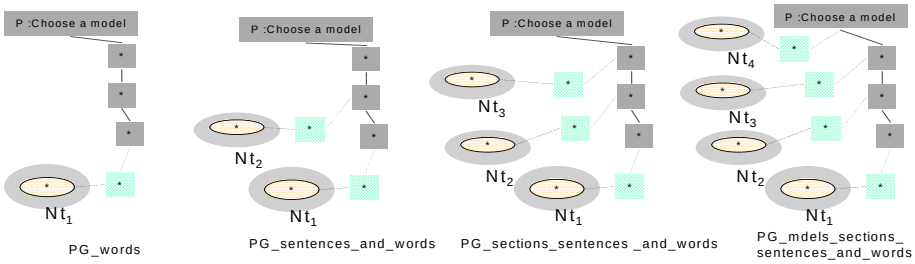

Fig. 5. *Potential graphs* enabling to retrieve traces containing less and less "fine" nodes.

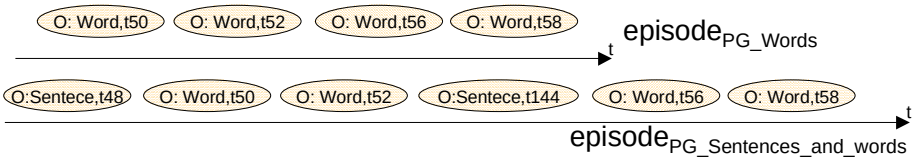

Fig. 6. Episodes calculated with *potential graphs* containing contextualization relations

When applying Club♣ to a concrete computer program it is important to create well structured observation and use models and a convenient set of *potential graphs* in order to get episodes with a high *reusability factor*.

7. Experience reuse

A computer program equipped with a Club♣ based tracing system can help users in two different manners:

- as an assistant : following user actions continuously and proposing help tips after each action;
- as a counselor : tracing the user actions but only to intercede when help is explicitly asked.

In both cases the help system has to find similar situations in the collected traces, in order to find inspiration in the already found solutions. The fulfilling of tasks is supported by the case based reasoning [11]. The implementation of this technique needs several formalizations.

7.1. The cases in the Club♣ model

In the section 6 we have presented several methods to calculate episodes in the use traces gathered following the Club♣ model. We consider every episode found this

way a *case* in the CBR meaning of the word. These episodes contain as well the problem part of the case as the solution part. The first nodes composing the episode can be considered as the problem and the remaining nodes as the solution. The identification of the target problem and solution is done when help is needed. Generally the target problem is given by the user's last actions and the solution will give his next steps.

The observation and use model applied to the computer program have to enable a tracing that facilitates the identification of problem/solution couples. It is as well necessary to be able to compare and to adapt episodes. To do this we create ordered *potential graph* suites enabling the retrieving ever less finer episodes. With these *potential graphs* the system can calculate episodes representing the current working context with increasing depth explanations, covering a growing period of time. For example while the episode got with first *potential graph* of the Fig. 5 begins at the time t50, the episode created with the second one commences at t48 and if we calculate the episode retrieved with the PG_models_sections_sentences_and_words, its starting time would be t43.

In order to illustrate our approach let us consider the trace left by a novice user writing his CV represented on the Fig. 7. This trace corresponds to the observation model of the Fig. 3. The session begins with the node [U :Jean,t40] and continues with the creation of a document following the generic model. The user types two phrases, one concerning his degrees, the other his experiences. Once he types the word Assistant, represented by the node [O :Assistant,t60] he asks himself (and the system) what else can he put on his CV? In this situation the episodes for a case based reasoning cycle are calculated departing from this node. Applying the *potential graphs* of the Fig. 5, the episodes of the Fig. 8 are retrieved. These episodes contain only *concrete objects*.

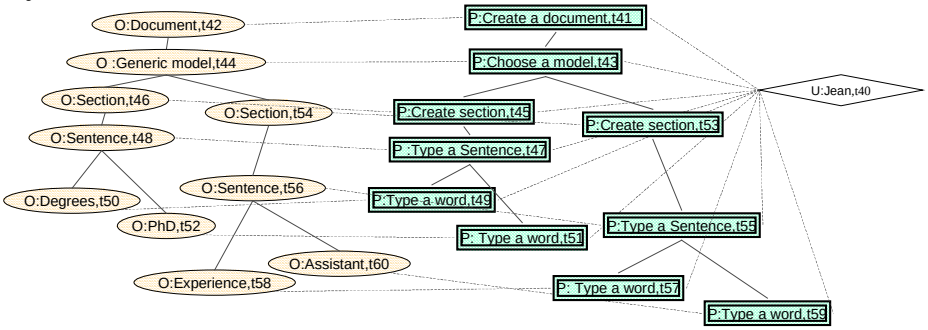

Fig. 7. Traces left by a document creation session

The episode E'1 is retrieved using the *potential graph* : PG_words, the episode E'2 using the PG_sentences_and_words and so on. In this situation the episodes E'1,E'2,E'3,E4 and E5 are potential target cases.

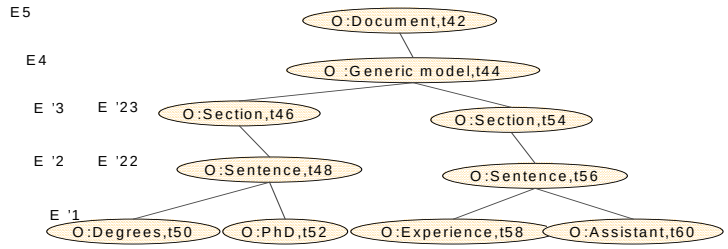

Fig. 8. Overlapping episodes retrieved from a document creation trace

Consider now that another, more expert user had already created a CV. He left the trace represented on Fig. 9. We can notice that this user knew the existence of a CV model and choose it to create the document. He entered his degrees, and the languages he speaks.

When the first user asked for help, both traces were in the global graph G, so the *potential graphs* are instanciated in the trace left during the session identified by the node [U :Pierre, t0] as well. The episodes retrieved from this trace are represented on Fig. 10. We find here the same kind of episodes as on the Fig. 8, this time being potential source cases.

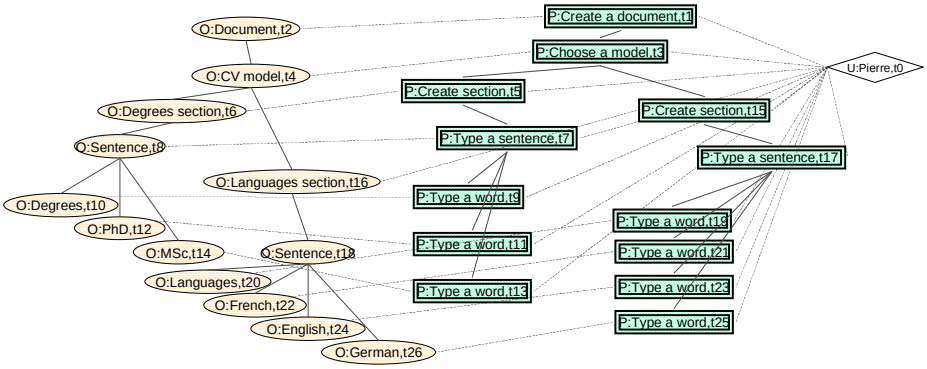

Fig. 9. Traces left by a document creation session

Now that the cases are retrieved, they need to be compared in order to find the most similar ones. When comparing episodes, the number of similar nodes is considered as well as the total number of nodes. We search the longest episode containing the maximum number of similar nodes.

After comparison we find that episodes E'41 and E'1 began with similar nodes, but episodes E'32 and E'2 (Fig. 11) too. As the latter are longer, represent a larger context, we consider the episode E'32 as being a better source case.

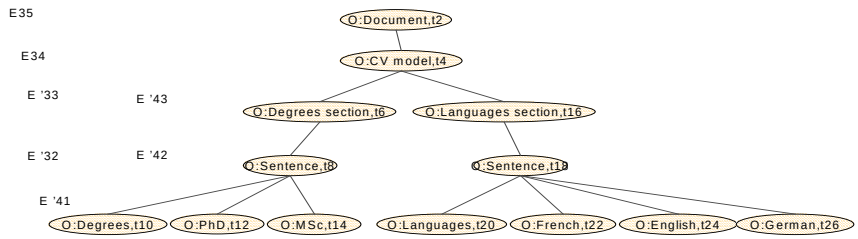

Fig. 10. Overlapping episodes retrieved from a document creation trace

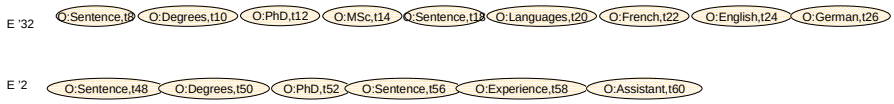

Fig. 11. Two episodes extracted from the traces left by the document creation sessions.

7.2. The actual help, case adaptation

Once the similar episodes found, several things can be done: the system can simply show the found similar episode to the user in order to give him new ideas. If the applied use and observation model enable it, more advanced adaptation can be done. In our example, we can consider that the generic model is less precise than the CV model, so the system could propose the help asking user to change his document model from generic to CV model and to add a languages section.

8. Conclusion

In this paper we presented an original use trace model, enable to gather user actions while using a computer program in a homogeneous, highly connected and well formalized structure. The basic idea was to represent as well users as *procedure*s and objects as elements playing a role in the use of a computer program in a same way. The graph of the Club♣ model is in fact a very rich log file, with multiple use trace retrieving possibilities. We introduced a robust exploitation tool, the *potential graph*, which enables us to filter in a flexible manner the traces left by users and to create specific episodes aligning nodes of the graph.

We are currently working on a better formalization of adaptation and comparison methods. We plan to consider not only ordered nodes but sub-graphs as cases to compare and to adapt.

The text processor application was used only for simplicity reasons. We are actually working on the adaptation of the Club♣ model on our audiovisual document annotation system.

References

1. E. Auriol, R. M. Crowder, R. MacKendrick, R. Row and T. Knudsen: Integrating Case-Based Reasoning and Hypermedia Documentation: An Application for the Diagnosis of a Welding Robot at Odense Steel Shipyard. Lecture Notes in Computer Science, Vol. 1650, (1999), 372-384
2. A. Bénel, E. Egyed-Zs., Y. Prié, S. Calabretto, A. Mille, I. Andréa and P. Jean-Marie: Truth in the Digital Library: From Ontological to Hermeneutical Systems. ECDL 2001 European Conference on Research and Advanced Technology for Digital Libraries, Darmstadt (D), Springer-Verlag, Vol. 2163 (2001), 366-377
3. L. Birnbaum, R. Bareiss, T. Hinrichs and C. Johnson: Interface Design Based on Standardized Task Models. Intelligent User Interfaces, San Francisco CA, USA, ACM, ACM, Vol. (1998), 65-72
4. B. Chandrasekaran, J. R. Josephson and V. R. Benjamins: Ontology of tasks and methods. Eleventh Workshop on Knowledge Acquisition, Modeling and Management (KAW '98), Banff, Canada, Vol. (1998), 25
5. J. Charlet, B. Bachimont, J. Bouaud and P. Zweigenbaum: Ontologie et réutilisabilité: expérience et discussion. (1996), 69-87
6. T. Clarke: Epistemics PC-Pack. http://www.epistemics.co.uk/products/pcpack/tools/ (2001)
7. J. Delgado and N. Ishii: Formal Models for Learning User Preferences, A Preliminary Report. International Joint Conference on Artificial Intelligence (IJCAI-99), Workshop on Learning about Users, Stockholm, Sweden, Vol. (1999), 8p
8. Dukas: Dukas Graphical Task Modeling Tool. http://www.cc.gatech.edu/gvu/user_interfaces/Mastermind/Dukas/ (1998)
9. E. Egyed-Zs., A. Mille, Y. Prié and J.-M. Pinon: Trèfle : un modèle de traces d'utilisation. Ingénierie des Connaissances, Rouen, F, Vol. (2002), 13
10. E. Egyed-Zs., Y. Prié, A. Mille and J.-M. Pinon: A graph based audio-visual document annotation and browsing system. RIAO 2000, Paris, France, Vol. 2 (2000), 1381-1389
11. A. Mille: Associer expertise et expérience pour assister les tâches de l'utilisateur. PhD Thesis, CPE Lyon France, (1998)
12. Y. Prié: *Modélisation de documents audiovisuels en Strates Interconnectées par les annotations pour l'exploitation contextuelle*. INSA-Lyon, (1999) 270
13. Y. Prié, T. Limane and A. Mille: Isomorphisme de sous-graphe pour la recherche d'information audiovisuelle contextuelle. 12ème congrès Reconnaissance de Formes et Intelligence Artificielle, RFIA2000, Paris, FR, Vol. 1 (2000), 277-286
14. Protégé: Protégé 2000. http://protege.stanford.edu/ (2002)
15. G. Schreiber and J. Wielemaker: ModelDraw. http://www.commonkads.uva.nl/INFO/tools/modeldraw.html (2001)
16. B. Trousse: Evaluation of the Prediction Capability of a User behaviour Mining Approach for Adapative Web Sites. In RIAO 2000, 6th Conference on "Content-Based Multimedia Information Access", Paris, C.I.D. C.A.S.I.S., (2000), 1752-1761