World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:1, No:8, 2007

Improved Algorithms for Construction of Interface Agent Interaction Model

Huynh Quyet Thang, Le Hai Quan

***Abstract*—**Interaction Model plays an important role in Model-based Intelligent Interface Agent Architecture for developing Intelligent User Interface. In this paper we are presenting some improvements in the algorithms for development interaction model of interface agent including: the action segmentation algorithm, the action pair selection algorithm, the final action pair selection algorithm, the interaction graph construction algorithm and the probability calculation algorithm. The analysis of the algorithms also presented. At the end of this paper, we introduce an experimental program called "Personal Transfer System".

***Keywords*—**interface agent, interaction model, user model.

I. Introduction

MODEL-BASED Software Agent for developing Intelligent Interface Agent is a good mathematical problem which was studied in the recent years. Neal Lesh, Charles Rich and fellow-workers developed researches and put forward Architecture model MI^{2}A (Model-based Intelligent Interface Agent Architecture) [1,2]. In this one, Interaction model and Task model play a very important role [1]. Predictor agent used Interaction model for predicting the future action as well as the purpose of users [1]; User Model Manager updated Interaction history and Interaction model. Yanguo Jing [3,4,5] studied process proposal and algorithms for developing interaction model of interface agent. In this paper we presented improving the algorithms for development interaction model of interface agent, which helps it to be developed more effectively.

The structure of this paper is developed as follows: In section II we presented Interaction model, method of developing it and algorithms applied in Development interaction model process. In section III we presented in detail algorithms applied in Development interaction model process and proposal improvement applied for algorithms with analysis, evaluation of algorithms complication level. In section IV we presented experiment installation on specific application. In section V is evaluation and conclusion.

Manuscript received January 15, 2007. This work was supported in part by the Ministry of Science and Technology of Vietnam under Grant KHCB2.034.06.

Huynh Quyet Thang is a Head of Software Engineering Department, Hanoi University of Technology, Hanoi, Vietnam (phone: 844-8682595; fax: 844-8692906; e-mail: thanghq@it-hut.edu.vn).

Le Hai Quan is a senior engineer of the FPT corporation, Hanoi, Vietnam. (e-mail: quanlh76@yahoo.com).

II. Interaction Model and Proposed Improvement

Figure 1 shows interface architecture MI^{2}A (Model-based Intelligent Interface Agent Architecture) [1,2,6]. In this model there are interface agents: DM Manager - Domain Model Manager, facilitator agent, reactor, guide, reactor, UM Manager-User Model Manager). The details of each agent role are in [1,2]. Interaction model plays an important role in this architecture model. The observer follows one person's action to supervise their interactions and collect actions. It updates the latest actions of the user in the predictor. Observer agent also sends action chains to the user model manager agent in order to update the interaction history of the user. Using interaction model, predictor can predict future actions of user. The method of developing interaction model was illustrated in figure 2, including 5 steps: action segmentation, action pair selection, final action pair selection, interaction graph

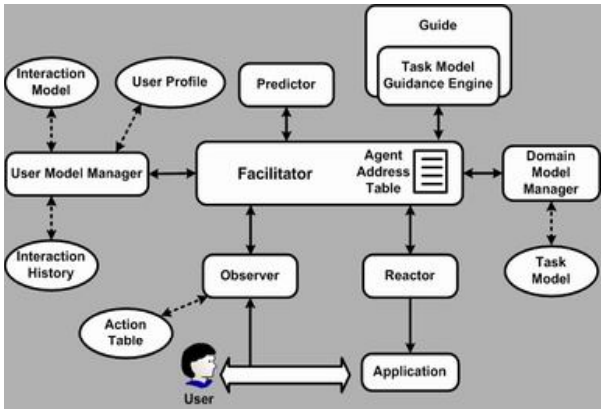

Fig. 1 Model-based Intelligent Interface Agent Architecture

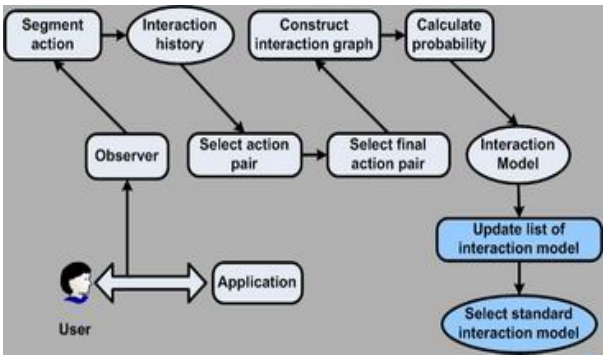

Fig. 2 Method of constructing interaction model

construction and probability calculation [1,5]. When the user interacts with the system, many actions and manipulations are carried out. The Agent will accept these actions. When trying to finish the task, user actions, in general, depend on previous actions and are related to the task. However, actions performing different tasks are relatively independent of each other. In the step of action segmentation, the actions accepted would be segmented into sensible action chains, telling which action the user is trying to finish. Those action chains will be added into user interaction history. In the step of action pair selection, it will calculate how many times action pairs appear, and arrange action pairs in appearance order. In the step of final action pair selection, some action pairs having more times of appearance than or exceed a certain threshold will be taken out from valid action pairs. Interaction graph is constructed to show action order information. In this graph each action can have more than one succeeding action, which can happen and is called post-condition actions. Since then each one will be assigned a probability value showing the possibility of happening; and the post-condition action with the highest probability will have the highest possibility of becoming the user's next action. We proposed improvement of interaction model by adding the updating the list of interaction model and selecting the standard one.

In [3,4] it puts forward 5 applied algorithms constructing the interaction model: the action segmentation algorithm, the action pair selection algorithm, the final action pair selection algorithm, the interaction graph construction algorithm, and finally the probability calculation algorithm. Next we will present in detail algorithms and proposal improvements to each algorithm and the interaction model list updating algorithm.

$S_{temp} \leftarrow \emptyset$
$a_j \leftarrow$ **null**
foreach a_i **in** L **do** {
if Task(a_j) $\subseteq$ Task(a_i)
or Task(a_i) are the post-condition tasks of Task(a_j)
then {
Add a_i to S_{temp} as the last added action in it
$a_j \leftarrow a_i$ (a_j is the last added action in S_{temp})
}
else {
Add S_{temp} to S
$S_{temp} \leftarrow \emptyset$
Add a_i to S_{temp}
$a_j \leftarrow a_i$ (a_j is the last added action in S_{temp})
}
}
S : Interaction history
S_{temp} : Segmented action chain
a_j : Previously attempted action
a_i : Current action
Task(a): Function returns the tasks that associated with action a

Fig. 3 Action segmentation algorithm

III. An Algorithms for Construction of The Interaction Model and Proposed Improvements

A. The action segmentation algorithm

In Model-based Intelligent Interface Agent Architecture (Figure 1), observer agent will take responsibility for collecting all users' actions. Observer agent follows user's actions, collects them (due to events of mouse and keyboard), selects the meaningful actions and sends them to User model manager agent. This one will create and save Interaction model. Collecting method is quite simple, but it's complicating to segment theses actions into meaningful actions. A chain of meaningful actions will include user's actions, which are carried out to get a certain task. We will distinguish the function of each action followed by task model. Each action can associate with one or more tasks in task model. A chain of actions won't be terminated only when it is in one of the following cases [3,4]:

- When the associated tasks of the new-collected action include the previously attempted task.
- When the associated tasks of the new-collected action are post-condition tasks of the previously attempted task.

Description of this algorithm is shown in figure 3 [3,4].

Evaluative analysis: This is the first algorithm among 5 algorithms constructing user's interaction model. The action segmentation algorithm proves quite simple and deploy easily. This algorithm bases oneself on Task model constructed before and mapping board of duties in the task model through actions collected by users (these actions will be also defined before) in order to segment chain of actions collected from users into smaller chains (meaningful chain). The segmentation condition is based oneself on post-condition tasks in Task model. The meaningful action chain needs constructing, at the first moment is empty. Algorithm inspects each action in "coarse" action chain collected by observer agent. Examine its task is whether the post-condition task of task associating action inspected before or not. If this condition is satisfied, which means that this action is still belong to action chain of which purpose is to fulfill the current task, it will be taken in the action chain. In contrast, when this condition is not satisfied, which means that the task associated with this action is not the post-condition task of the task associated with the previous action; it can be considered that this action is the beginning of a new task. In other words, the user moves on to carry out a new task, and the current task finished in the previous action. We will finish this action chain here and save it in the interaction history. Next this action chain will be restarted empty and begin a cycle of creating new action chain, which will finish the next task.

The performance time is assessed as linear O(n); the number of repetition times is equal to the number of action collecting. The result of this algorithm will be an important starting point to the next algorithm.

B. The action pair selection algorithm

Based on the first algorithm result, the action pair selection

algorithm will select every action pair from every chain of actions in the interaction history (from the action segmentation algorithm), and then put them into list of action pair, signed PAIRLIST. The actions from chains of actions will be put in pairs in order. To make it convenient for the next algorithm, this algorithm also count the appearance times of these action pairs each time they repeats in the PAIRLIST, put it into a variable signed COUNT.

```
foreach A_s in S do {
 if (LENGTH(A_s) < 2)
  continue
  foreach a_j in A_s do (j←n downto 1) {
   foreach a_k in A_s do (k←j-1 downto 1) {
    if (a_k, a_j) ⊆ PL then {
     COUNT(a_k, a_j) ← COUNT(a_k, a_j) + 1
        }
    else {
     Add (a_k, a_j) to PL
     COUNT(a_k, a_j) ← 1
        }
     }
   }
}
S  : Interaction history
A_s: Chain of segmented action in S
n  : Number of actions in S
a_k, a_j : Action in A_s
PL   : List of selected action pairs
(a_k, a_j)  : Action pair is selected and put in PL
COUNT(a_k, a_j) : Number of (a_k, a_j) appearance times
LENGTH(A_s) : Function returns number of actions in A_s
```

Fig. 4 Action pair selection algorithm

Before analyzing the action pair selection algorithm, it is necessary to examine following definitions [16]:

Definition1: (action1, action2) is used to represent an ordered action pair which means action1 is performed before action2 is performed. actionPair(J) is used to represent an action pair, which is the Jth one in a PAIRLIST (see Definition 3 below). actionPair(J).firstAction is used to represent the first action in this action pair; actionPair(J).secondAction is used to represent the second action in this action pair.

Definition 2: COUNT(action1, action2) is used to represent the number of occasions that action pair (action1, action2) occurs in the interaction history.

Definition 3: PAIRLIST is used to represent the action pair list that is produced from the interaction history.

This algorithm concern only about the order of two actions; two actions in an action pair will define an order for the action pair. Every certain interaction history includes many chains of actions, the algorithm that select action pair from table 4.2 will use the action pairs collected from the interaction history and count the appearance times of every action pair, using function COUNT above.

Improving the action pair selection algorithm:

The algorithm begins with examining every chain of actions in the interaction history. With every chain of actions, it will examine each action from bottom to top then gather them in pairs. If one gathered pair is not yet in the PAIRLIST, then assign 1 to their COUNT and put them into PAIRLIST. On the contrary, if the action pair is already in the PAIRLIST, then add 1 to their COUNT. However, there are some disadvantages about the action pair selection algorithm (figure 4). In case there are two similar actions in an action pair, but the algorithm does not reject this kind of pair. Moreover, in case there is only one action in an action pair, the algorithm will create only one action pair in which there are two similar actions. Thus, this action pair is not chosen; then in case chain of actions has only one action, we can reject it. To overcome these disadvantages, we can fix the algorithm by adding element-checking part (action) in the chain of actions before gathering in pairs (blue text in the algorithm is the added part).

Fixing process: In the third loop of the algorithm, while chain of actions is being checked, index l begins from (j-l) not from j; then we exclude the chains of actions in which there are two similar actions. To exclude chains of actions that have only one action, we can use function LENGTH that return the number of actions in chain of actions to examine. If the returned value of LENGTH is larger than 1 then we can do the next step of the algorithm. If not, we will ignore this chain of actions and move on to the next chain of actions.

The algorithm is simply carried out with three joined loops. If we see the total actions in chains of actions as n, then the algorithm complex level is $O(n^2)$, time counting is multinomial. After the algorithm finishes, we have the list of chains of actions in order and their total appearance times in the interaction history. It will be the input for the third algorithm.

C. The final action pair selection algorithm

After receiving the list of action pairs from the action pair selection algorithm, the final action pair selection algorithm aims at selecting meaningful action pairs from the list. In this algorithm, we mention of opposite action pair. There are many reasons for the appearance of opposite action pairs such as task model, distributing method, the number of actions attached to task. In case one task need many actions in order to be complete and these actions is not necessarily in order, then there will be opposite action pairs. Opposite action pairs usually appear when users operate in different orders, or when they fix wrong operations. In case each task is associated with only one action, there is no opposite action pair. Because at the first algorithm, while chains of actions are being created, this algorithm itself did not put actions that the task associated with it is in previous conditional task chain of the task associated with previous action into chain of actions. Yet, these actions can create opposite action pairs. In fact, however, one task can be associated with some actions and thus opposite action pairs case can occur. In order to carry out the final action pair selection algorithm, we should check up the following concepts [3,4]:

Definition 4: The Action Pairs (ai, aj) and (aj, ai) are called contradictory action pairs.

Definition 5: Action Pair Confidence (APC):

$APC(a_i,a_j)=COUNT(a_i,a_j)/(COUNT(a_i,a_j)+COUNT(a_j, a_i))$.

APC is used to represent the likelihood that action ai occurs before action aj.

The final action pair selection algorithm will be presented in figure 5. A certain action pair in chain of action pairs (PAIRLIST) is developed from appearance times counting algorithm of action pairs; cardinal number COUNT and one threshold to choose, signed as THRESHOLD. This algorithm will select the final action pairs from PAIRLIST on condition that the action pairs with APC equal or larger than THRESHOLD. The selected action pairs will create the chain of final action pairs, signed as FPAIRLIST.

```
foreach ap_i in PL do {
 Pass ← TRUE
  foreach ap_j in PL do {
   if ISCONTRA(ap_i, ap_j) then {
    if THRESHOLD = 1 then
     Pass ← FALSE
     else if APC(ap_i) < THRESHOLD then
      Pass ← FALSE
      break
                        }
                    }
    if Pass then
   Add ap_i to FPL
                        }
```

PL : List of action pairs
FPL : List of final action pairs
Pass : Check whether APC of action pair is greater than threshold
ap_i, ap_j : Actions in PL
$APC(ap_i)$: Action Pair Confidence
$APC(ap_i)=COUNT(ap_{i1},ap_{i2}) / (COUNT(ap_{i1}, ap_{i2}) + COUNT(ap_{i2}, ap_{i1}))$
$ISCONTRA(ap_i, ap_j)$: Function returns TRUE if ap_i and ap_j are contradiction action pairs, otherwise returns FALSE.

Fig. 5 Final action pair selection algorithm

Improving the final action pair selection algorithm:

In this algorithm, we use variable APC to determine whether one action pair can be put into the chain of final actions or not. APC is the ratio of the appearance times of action pair to their total appearance times plus the appearance times of their opposite action pair. If APC exceed a specified threshold, then this action pair can be put into the list of final action pairs. The final action pair selection algorithm will check up every action pair in the list of final action pairs, signed as FPAIRLIST. On the contrary, if the opposite action pair of the being-checked action pair appears, the algorithm will start calculating APC, and then compare APC with the specified threshold, signed as THRESHOLD. If APC is equal or larger than the threshold then the action pair will be put into FPAIRLIST. If APC is smaller than the threshold then the action pair will be left out. Thus, It depends a lot on selecting a threshold to decide whether one action pair can be put into FPAIRLIST or not. The value of threshold is within (0, 1], if we choose 1 as threshold then only action pairs without opposite action pairs are put into FPAIRLIST. The chosen threshold must be larger than 0. Because if we choose 0 as threshold then it is no need to perform the algorithm but all the action pairs in PAIRLIST can be put into FPAIRLIST. There is still one point need to be fix in the algorithm in order to make it more optimal. It is examining cases each time we choose different thresholds. It is separated into two cases. If we choose 1 as threshold then it is not necessary to calculate APC but we do not put it into FPAIRLIST as long as they are two opposite action pairs. Thus we can escape from loop. If the threshold is smaller than 1, then compare with threshold and escape from loop right away (the blue text in the algorithm is the added fixing part).

This algorithm is easily carried out with two joined loops. The list of final action pairs FPAIRLIST is used to construct interaction graph.

D. The interaction graph construction algorithm

```
foreach ap_i in FPL do {
  if NOT ISPRECOND(ap_i.firstAction,
ap_i.secondAction) then {
  Add ap_i.firstAction to pre-condition of
ap_i.secondAction
  Add ap_i.secondAction to post-condition of
ap_i.firstAction
   }
 Prune_Algorithm_1(ap_i.firstAction, ap_i.secondAction)
 Prune_Algorithm_2(ap_i.firstAction, ap_i.secondAction)
}
```

FPL : List of final action pairs
ap_i : Action pair in FPL
ap_i.firstAction : First action in ap_i
ap_i.secondAction : Second action in ap_i
$ISPRECOND(a_1, a_2)$: Function returns TRUE if a_1 is pre-condition action of a_2, otherwise returns FALSE.
$Prune\_Algorithm\_1(a_1, a_2)$: First prune algorithm
$Prune\_Algorithm\_2(a_1, a_2)$: Second prune algorithm

Fig. 6 Interaction graph construction algorithm

Action pairs in order were constructed from chain of action that stored in the interaction model. Then, final action pairs are chosen from action pairs in order based on threshold ratio. Finally, these final action pairs, one by one, will be put into a graph (we use an arrow to connect 2 actions in a pair, the arrow points at the second action in action pair). Algorithm Prune Successor(s) and Prune Predecessor(s) [3] are used to simplify the graph. Each node in the graph of the interaction model includes: ID, name, pre-condition, post-condition, position and probability distribution. Probability distribution

```
Prune_Algorithm_1(a1, a2) {
 foreach ac1 in POST(a2) do {
   if a1 ⊆ PRE(ac1) then {
    Remove a1 from PRE(ac1)
    Remove ac1 from POST(a1)
     }
     else
    Do Prune_Algorithm_1(a1, ac1)
  }
}
```

a_1, a_2 : Two actions in action pair are parameters
POST(a) : Post-condition actions of a
PRE(a) : Pre-condition actions of a
ac_1 : Action in POST(a_2)

Fig. 7 First Prune Algorithm

```
Prune_Algorithm_2(a1, a2) {
 foreach ac1 in PRE(a1) do {
  if a2 ⊆ POST(ac1) then {
    Remove a2 from POST(ac1)
    Remove ac1 from PRE(a2)
     }
     else
   Prune_Algorithm_2(ac1, a2)
  }
}
```

a_1, a_2 : Two actions in action pair are parameters
POST(a) : Post-condition actions of a
PRE(a) : Pre-condition actions of a
ac_1 : Action in PRE(a_1)

Fig. 8 Second Prune Algorithm

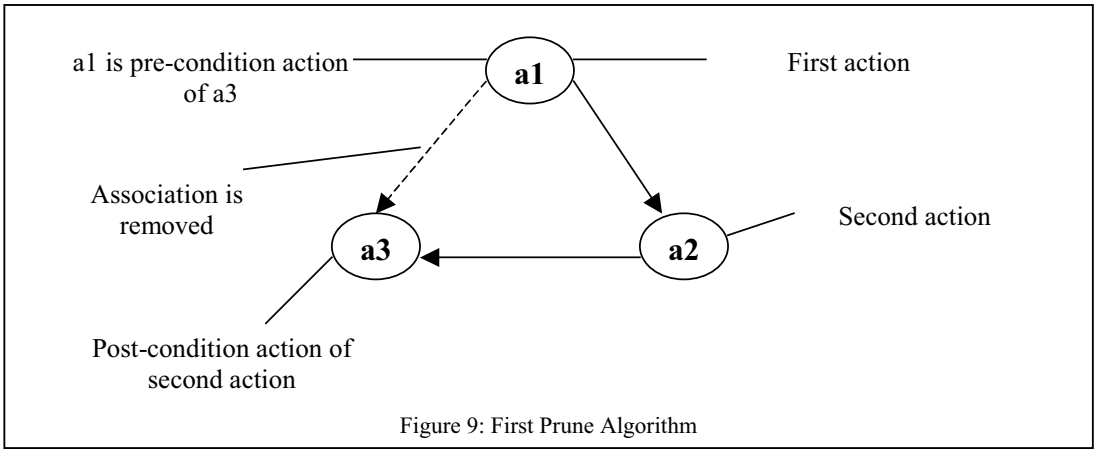

Figure 9: First Prune Algorithm

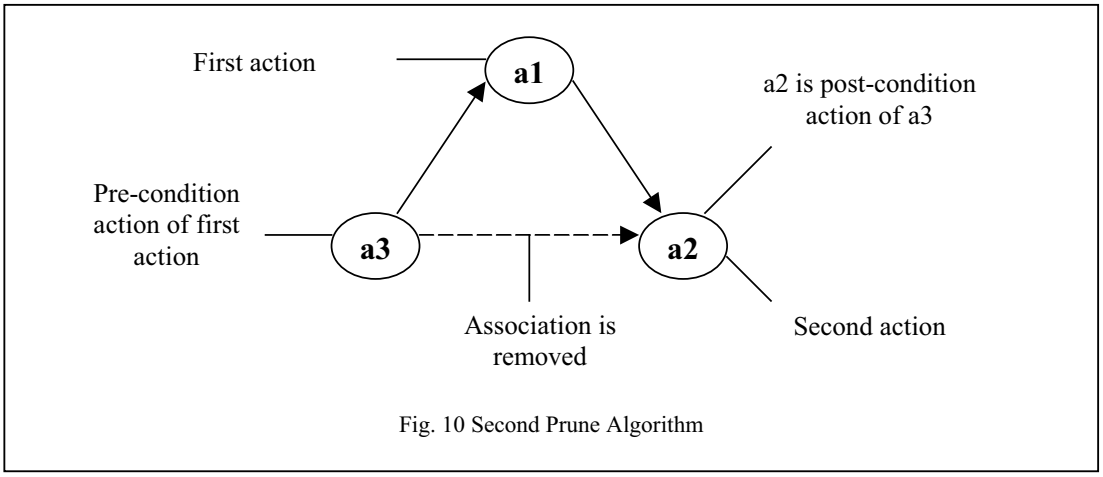

Fig. 10 Second Prune Algorithm

of each node shows us the probability that whether a post-condition action can be the next action of the node or not. The interaction graph construction algorithm is described in figure 6. First Prune Algorithm (Prune_Algorithm_1) as shown in figure 7 [3,4], is used to remove the first action from the pre-conditions of all the second action's Successors.

First Prune Algorithm (Prune_Algorithm_1) as shown in figure 7 [3,4], given an action pair, this algorithm removes the first action from the direct pre-condition of all the second action's post-conditions.

Second Prune Algorithm (Prune_Algorithm_2) is shown in figure 8 [3,4], it is used to remove the second action from the Post-condition of all the first action's Predecessors.

This algorithm removes the second action from the direct post-condition of all the first action's pre-conditions. In this algorithm, there are also two parameters - two actions in action pair (a1, a2).

Improving the interaction graph construction algorithm:

The algorithm will examine every action pair in the FPAIRLIST, decide whether the first action is the pre-

```
Update_IG(IG) {
 ISEXIST ← FALSE
 foreach IG_i in IGLIST do {
  if IG_i = IG then {
   COUNT(IG_i) = COUNT(IG_i) + 1
   ISEXIST ← TRUE
  break
     }
  }
   if not ISEXIST then {
     COUNT(IG) = 1
     Add IG to IGLIST
  }
}
```

IGLIST : List of interaction model
IG_i : Interaction model in IGLIST
ISEXIST : Check the existence of interaction model in IGLIST
COUNT(IG) : Repeated times of interaction model

Fig. 11 Interaction model list update algorithm

condition action of the second one or not. If not, we will add the first action to the list of pre-condition actions of the second one, and add the second action to the list of post-condition actions of the first action. After that, the algorithm will perform two prune algorithms: the first prune algorithm and the second prune algorithm. These prune algorithms aim at removing unnecessary actions in the list of their pre-condition and post-condition actions. In other words, they remove unnecessary connection in the interaction model. The first algorithm will remove the first action of action pair from pre-condition actions of post-condition action of the second action.

On the contrary, the second prune algorithm will remove the second action from the post-condition actions of pre-condition action of the first action.

Through the figure, we can imagine how these two algorithms remove unnecessary connection between action pairs in the model. The algorithm that examines every action pair has linear time, the two prune algorithms have linear time as well, and thus it is easy to perform.

For the improvements, we add up the storage part in order to compare interaction models. We will build a list of interaction models, and a calculating variable named COUNT for each model. Every time after an interaction model is built, we compare this interaction model with other interaction models in the list. If this interaction model is not yet in the list, then we add it in and assign 1 to variable COUNT. Otherwise, if the interaction model already exists, we have to add 1 to variable COUNT. The interaction model list update algorithm improves interaction graph construction algorithm as shown in figure 11.

The important thing in this supplementary algorithm is, while comparing two interaction models whether they are "equal" or not, we can skip optional actions in the interaction model. Because, these actions can be done or not, thus it is not necessary to put them into compare.

After each time revising interaction model, we can check up variable COUNT of every interaction model. If one interaction model has the largest variable COUNT, then it is the standard interaction model for users.

```
ProbabilityCalculation(A_s) {
 foreach (a_i, a_j) contiguous in A_s do {
   if a_i ⊆ IG then
  UpdateProbability(a_i, a_j)
  }
}
UpdateProbability(a_1, a_2) {
 if PRO_POST(a_1) is null then
  PRO_POST(a_1) ← 1/n
  PRO_POST(a_1) ← PRO_POST(a_1) * α
  PRO(a_2) ← PRO(a_2) + (1-α)
                    }
```

IG : Interaction graph
A_s: Chain of actions is parameter
(a_i, a_j) : Contiguous action pair in A_s
(a_1, a_2) : Action pair is parameter
$PRO\_POST(a_1)$: Probability of post-condition actions of a_1
$PRO(a_2)$: Probability of a_2
n : Number of post-condition actions of a_1
α : Specified constant

Fig. 12 Probability calculation algorithm

E. The probability calculation algorithm

The probability calculation algorithm is used to calculate probability of the next action occurrence to each action node in the interaction model. Each action depends only on the previous action. The collected data are used to calculate each action pair continuous occurrence and then calculate probability. This algorithm increases probability of the chosen closest post-condition action. The algorithm uses a constant named α, this constant will decrease probability of the post-condition actions but not the closest ones, easily by multiple probability of these actions with α. Probability of the closest post-condition actions is multiplied with α, too, but plus (1-α). This way, we can make sure that probability of the post-condition actions is equal or smaller than 1, and probability of the closest post-condition actions will continuously increased or stay still in some cases.

Evaluation analyzing: The algorithm begins with examining every contiguous action pair in each action chain, then check up whether the first action is in the interaction model or not. If the action is in the interaction model, then we must revise probability for the action. This algorithm use only one loop, therefore the complication of calculation is linear, easy to perform.

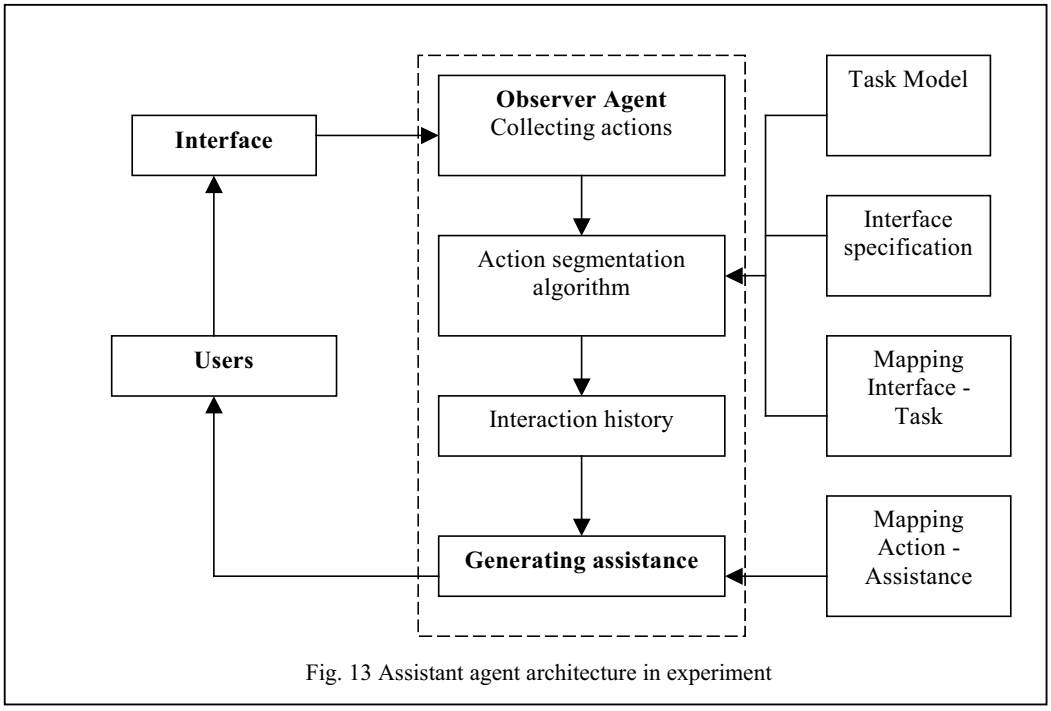

Fig. 13 Assistant agent architecture in experiment

F. Conclusion

The five algorithms above have a close relation with each other; the result of the previous algorithm will be the input for the next one. The action segmentation algorithm is the first to perform; it aims at selecting chains of meaningful actions (action segmentation) among the actions collected from users and then stores it in the interaction history.

The second algorithm to perform is the action pair selection algorithm. It uses the first algorithm's result - segmented chains of actions in the interaction history to select action pairs in order or meaningful action pairs. The target of this algorithm is to select user's actions in time order, then use them as a basis for the interaction model construction in the future. This algorithm uses chains of meaningful actions that were constructed since the previous algorithm. It selects action pairs in order then stores them in the list of action pairs (PAIRLIST). While selecting action pairs, it calculates these action pairs occurrence times in the interaction history and store them in variable COUNT.

The final action pair selection algorithm is the next one to perform. It use the results of the second algorithm - the list of action pairs in order (PAIRLIST) and the variable that does the action pairs occurrence times calculation in the interaction history (COUNT) to select the final action pair (the action pair with probability of being in the chain of actions used to complete the task). The algorithm begins selecting action pairs in PAIRLIST, gather them in pairs, check up whether they are contradictory action pairs or not; calculate APC of the first action pair and compare with threshold (THRESHOLD). If APC is equal or larger than the threshold then this action pair will be added in the list of final action pairs (FPAIRLIST). If APC is smaller than the threshold then this action pair will be left out, not in the FPAIRLIST.

The interaction graph construction algorithm is used to construct the interaction model graph of users to system. This algorithm examines action pairs in FPAIRLIST and then put them into the interaction graph in turn. However, it does not stop yet, after each action pair put into the interaction model, it will use two prune algorithms to remove actions coincided with the actions that was put into the model.

The probability calculation algorithm aims at calculating probability contribution for the actions in chain of contiguous actions collected from users. Based on the constructed interaction model above, this algorithm will use constant α to adjust probability contribution for action pairs with a defined α. We can find out the probability of an action pair with the highest occurrence possibility.

IV. Experiment and Practical Development

We test it with creating an interface for Banking Transfer System (belong to the BIDV - Bank of Investment and Development Branch Hanoi - Vietnam) in order to serve customers, allow them to do any of monetary transaction from an account to another one themselves. The transfer account must be user's account that was opened in BIDV, and the transferred one can be any account opened in any bank. Transfer is originally a profession of banks, teller is so familiar with this job, but it is difference to customers. Customers have never done this before and may get panic for the first time contacting with the program. Assistant agent is built to help customers do the transfers successfully. The agent will observe users, store the interaction history, analyze the interaction history, and then bring out the appropriate assistants for users under the form of instruction messages. Construction in testing problem of this agent is described in figure 13. While users interact with the interface, actions are

made and agent needs to record these actions. They are stored in the interaction history. When analyzing the interaction history, we can realize wrong movements or unnecessary movements from users. Thus, the interaction history will be the input of the assistant unit, help the process of giving appropriate instructions for users. The main target of the agent is to assist users while they are interacting with the interface. Thus, the assistance must be easy to understand, at the right time and right place. The assistant messages presented must be the combination between professor's experience, assessments from customers and potential users. As already stated, there are usually opinion polls and surveys before creating the assistant unit, make it "easy to identify", so it can really assist users when they are incapable of using.

The software is developed based on language C# of Visual Studio .NET, and the specification is on language XML. The program is tested with a set of data including information of users and more than ten accounts. The program is capable of reading documents under XML format, including task model specification, interface specification, mapping between task and elementary actions, mapping between interface and task.

V. Conclusion

In this paper, we presented generally current state about intelligent user interface, the importance of the intelligent user interface in use. The article also presented methods of constructing interface agent based on task model and interaction model, made a research on constructing interaction model based on five algorithms. The article contributes to completing the interaction model algorithms, inside there are corrections at three algorithms: the action pair selection algorithm, the final action pair selection algorithm and the interaction graph construction algorithm for one specified user. The article also proposes some improvements of the algorithms and makes it easier to execute effectively.

Based on presented theories, the article made a test construction of an assistant agent with the entire task model component, included descriptions and actions associated with each task. The agent can read the task model and documents under XML format. The agent can be integrated with the interface and then give instruction to help users in interacting with the interface.

The next direction of this research may focus on making the agent more intelligent, capable of talking to users like "a friend". First step, using some optional questions, the agent gives users some choices so they can define the next action needed to be done themselves. After that, much better, the agent will create for itself a larger knowledge basis that allows users to enter questions or their demands through keyboard. The agent must be capable of answering or carrying out some work for users.

Reference

[1] Andrew Garland Neal Lesh (2004), "Applying Collaborative Discourse Theory to Human-Computer Interaction" www.merl.com/reports/docs/ TR2002-004

[2] Andrew Garland, Neal Lesh (2004), "COLLAGEN: Applying Collaborative Discourse Theory to Human-Computer Interactio,www.merl.com/reports/docs/ TR2002-004

[3] Yanguo Jing (2001), "The Interaction Model Construction Method in Model based Intelligent Interface Agent Architecture ", Heriot-Watt University, Department of Computing & Electrical Engineering, Technical report RM/01/5, 2001.

[4] Yanguo Jing (2001), "The Domain Model Design in Model based Intelligent Interface Agent Architecture ", Heriot-Watt University, Department of Computing & Electrical Engineering, Technical report RM/01/4, 2001

[5] Yanguo Jing (2003). A Model Based Intelligent Interface Agent Architecture - Doctor of Philosophy, Heriot-Watt University, School of Mathematical and Computer Sciences, Computer Science Department Intelligent Systems Laboratory - http://smealsearch2.psu.edu/98584.html

[6] Huynh Quyet Thang, Pham Thanh Trung (2005). Building an Architecture Model and Tool for Interface Agent Generation. Proceedings of FAIR (Fundamental and Applied Information Technology Research) Hochiminh City, 23-24/9/2005, pp. 432-444 *(in Vietnamese)*

[7] Huynh Quyet Thang, Do Thanh Vu (2004). A Model of Development of the Interface Agent in Standalone Application. Posts and Telecommunication Journal, Special issue Research and Development on Telecommunications and Information Technology ISSN 0866-7039, No. 13, 12-2004, p. 73-81 *(in Vietnamese)*