Mise en place d'un système d'assistance personnalisée dans une application existante

Blandine Ginon, Stéphanie Jean-Daubias, Pierre-Antoine Champin

Mise en place d'un système d'assistance personnalisée dans une application existante

Blandine Ginon[1,2], Stéphanie Jean-Daubias[1,3] et Pierre-Antoine Champin[1,3]

[1] Université de Lyon, CNRS,
[2] INSA-Lyon, LIRIS, UMR5205, F-69621, France
[3] Université Lyon 1, LIRIS, UMR5205, F-69622, France
{prenom.nom}@liris.cnrs.fr

Résumé : Nous proposons un modèle générique d'assistance aux utilisateurs d'applications informatiques permettant la spécification et l'exécution de systèmes d'assistance personnalisée dans une application-cible existante, sans avoir à la modifier. Nous avons implémenté ce modèle dans un prototype opérationnel comportant un éditeur d'assistance destiné aux concepteurs de systèmes d'assistance et un moteur générique d'assistance. Ce moteur exploite des collecteurs d'informations capables de surveiller une application-cible et de l'informer des événements détectés. Des assistants épiphytes réalisent dans l'application-cible sans perturber son fonctionnement, les actions d'assistance élaborées en réponse aux besoins des utilisateurs.

Mots-clés : assistance à l'utilisateur, systèmes épiphytes, personnalisation, modèle générique.

1. Introduction

Le développement d'un système d'assistance pour une application est une tâche complexe et coûteuse, mais importante pour pallier les difficultés de prise en main et d'utilisation que peuvent ressentir les utilisateurs. Cependant, les développeurs négligent souvent ce travail pour se concentrer sur le développement des fonctionnalités de l'application. Une alternative consiste à adopter une démarche épiphyte pour permettre a posteriori la spécification et l'exécution d'un système d'assistance dans une application existante sans avoir à la modifier. Un assistant épiphyte est une application externe capable de réaliser des actions dans l'application-cible sans perturber son fonctionnement (Paquette, et al., 1996). La spécification de l'assistance pourrait ainsi être faite par une personne autre que le créateur de l'application.

Par ailleurs, des études ont montré qu'un système d'assistance peut être rejeté par les utilisateurs s'il les interrompt dans leur tâche pour leur proposer une assistance non pertinente (Galluccio, 2006) (Randall, et al., 1998). Pour cette raison, la personnalisation de l'assistance est une problématique pertinente : elle doit permettre la prise en compte des besoins spécifiques de chaque utilisateur, comme ses expériences, ses objectifs, ses connaissances, ses capacités et éventuels handicaps, ainsi que ses préférences.

Dans cet article, nous présentons notre modèle générique d'assistance aux utilisateurs qui permet la spécification et l'exécution de systèmes d'assistance personnalisée dans une application existante, appelée application-cible, sans avoir à modifier cette dernière. Nous avons partiellement mis en œuvre ce modèle dans un premier prototype opérationnel que nous présentons également.

2. État de l'art

Depuis les années 90, les systèmes épiphytes ont fait l'objet de plusieurs travaux. En particulier, Epitalk (Paquette, et al., 1996) permet de concevoir des systèmes conseillers épiphytes pour une application-cible développées en Smalltalk. Cependant, des connaissances de programmation en Smalltalk et en systèmes multi-agents sont requises.

Plus récemment, (Richard, et al., 2004) proposent un système générique permettant de greffer un système conseiller à un site web existant. Toutefois, l'assistance n'est pas personnalisable et est spécifique aux sites web. Le modèle CAMELEON (Carlier, et al., 2010) permet quant à lui d'ajouter un agent animé de manière épiphyte afin de guider les utilisateurs de plateforme de e-learning. Néanmoins, cette approche est basée sur l'ajout de balises html, et ne concerne donc que les applications web.

Dans le domaine des hypermédias, ExploraGraph (Dufresne, 2001) permet la spécification par l'enseignant, puis l'exécution de systèmes conseillers pour faciliter la navigation des apprenants au sein d'un scénario pédagogique d'ExploraGraph. L'assistance est proposée sous la forme d'un agent animé et peut être personnalisée en fonction du profil des apprenants et de l'historique de l'assistance. Enfin, Telos (Paquette, 2012) permet la spécification, puis l'exécution de systèmes conseillers épiphytes dans des scénarios pédagogiques de Telos.

L'assistance peut y être personnalisée en fonction du profil de l'apprenant et de l'historique de l'assistance. Les acteurs des scénarios pédagogiques peuvent être aidés par des messages textuels.

À notre connaissance, il n'est actuellement pas possible d'ajouter un système d'assistance épiphyte personnalisable dans une application existante non spécifique au web ou à un environnement donné, sans avoir à la modifier et sans connaissances en programmation.

3. Modèle générique d'assistance aux utilisateurs

Nous proposons un modèle générique d'assistance aux utilisateurs d'applications informatiques (cf. Fig. 2). Dans ce modèle, le processus d'adjonction d'un système d'assistance à une application existante est organisé en deux phases (cf. Fig. 1). La première phase concerne la spécification d'un système d'assistance par un concepteur d'assistance, elle est réalisée une seule fois pour chaque application-cible. La seconde phase concerne l'exécution de ce système d'assistance dans l'application cible, elle a lieu à chaque utilisation de l'application-cible par un utilisateur qui souhaite bénéficier d'assistance.

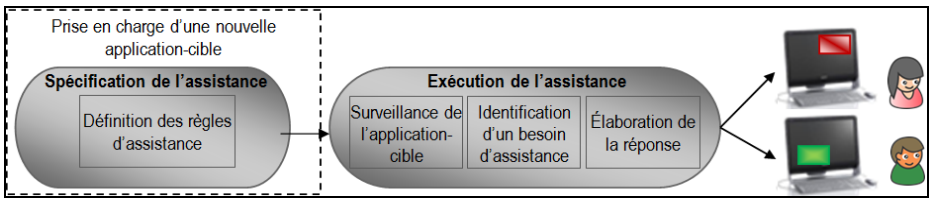

Fig. 1 – Processus d'adjonction d'un système d'assistance à une application existante

3.1 Spécification d'un système d'assistance

La spécification d'un système d'assistance pour une application-cible s'appuie sur un formalisme (cf. Ⓐ Fig. 2) qui permet la description de l'assistance pour cette application. Avec ce formalisme, l'assistance est spécifiée par un ensemble de règles d'assistance de la forme <événement déclencheur, conditions de déclenchement, actions d'assistance>. Ce formalisme est mis en œuvre dans un éditeur d'assistance à destination des concepteurs d'assistance. Ces derniers peuvent être le créateur de l'application-cible ou un utilisateur expert : par exemple, un enseignant qui utilise un logiciel éducatif peut

concevoir un système d'assistance destiné à ses étudiants afin de les aider à manipuler ce logiciel.

La description de l'assistance par un concepteur se fait en deux étapes. La première étape est automatisée et consiste à décrire l'interface de l'application-cible afin de pouvoir désigner ses composants par la suite. La seconde étape est l'étape principale, celle de définition des règles d'assistance. Dans cette étape, le concepteur de l'assistance doit définir les conditions de déclenchement de l'assistance. Nous avons défini quatre types de conditions de déclenchement que nous décrivons dans la section suivante. Ces conditions de déclenchement peuvent être combinées dans une formule logique associée à une règle d'assistance. L'événement déclencheur associé à une règle d'assistance est de même type qu'une condition sur les actions de l'utilisateur. Lorsqu'un système d'assistance est mis en œuvre dans une application-cible, si l'événement déclencheur se produit et si la formule logique composée de conditions de déclenchement est vérifiée, alors les actions d'assistance associées à la règle sont réalisées afin de répondre aux besoins de l'utilisateur.

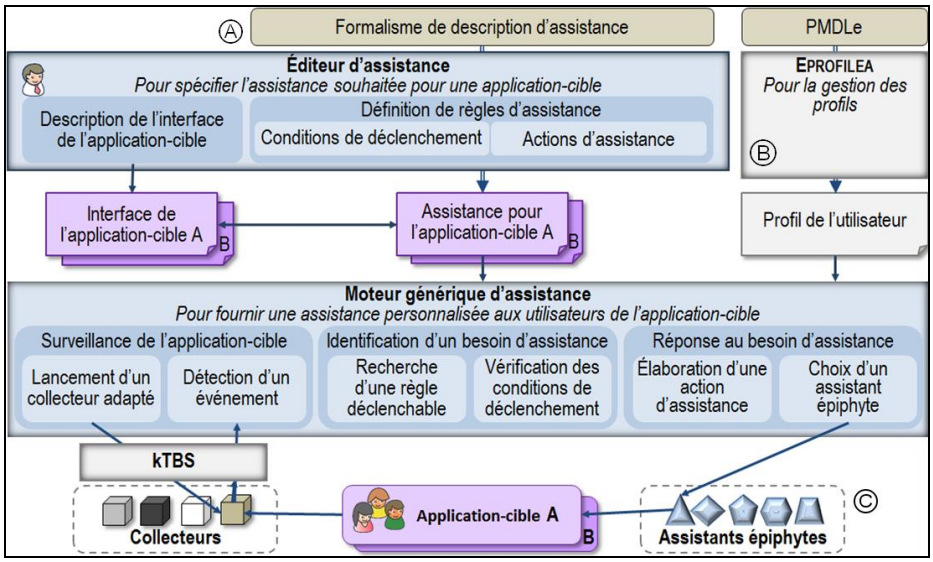

Fig. 2 – Modèle générique d'assistance aux utilisateurs

Les conditions de déclenchement de l'assistance

Les conditions sur les actions de l'utilisateur rendent possible la personnalisation de l'assistance en fonction des interactions entre

l'utilisateur et l'application-cible. Ce type de conditions de déclenchement concerne aussi bien les événements bas niveau, comme les actions de la souris et du clavier, que les événements plus haut niveau, comme un copier-coller ou l'envoi d'un mail. Par exemple, il est possible de déclencher une action d'assistance quand l'utilisateur clique sur le bouton X, lorsqu'il passe la souris sur l'image Y, ou lorsqu'il sélectionne un item dans la liste déroulante Z.

Les conditions sur le profil de l'utilisateur rendent possible la personnalisation de l'assistance en fonction des spécificités de l'utilisateur exprimées dans son profil. Nous avons choisi d'utiliser des profils d'utilisateurs respectant le formalisme PMDLe (Jean-Daubias, et al., 2011), implémenté dans l'environnement EPROFILEA (cf. ® Fig. 2) de définition et d'exploitation des profils respectant PMDLe. Il est possible de définir des profils contenant les informations jugées pertinentes par le créateur de profils en fonction des exploitations qu'il désire faire. PMDLe est associé à cPMDLe (Lefevre, et al., 2012), un modèle permettant de définir des contraintes sur les profils PMDLe, afin de sélectionner les utilisateurs en fonction des informations contenues dans leur profil (qui peut concerner les connaissances et compétences, les capacités et éventuels handicaps, les préférences…). Par exemple, si le profil de l'utilisateur contient des informations relatives à la maîtrise de l'application-cible, il est possible de déclencher une action d'assistance uniquement pour les utilisateurs qui ne maîtrisent pas une tâche donnée.

Les conditions sur l'historique de l'assistance permettent de faire évoluer l'aide proposée à l'utilisateur au cours de la ou des séances d'utilisation de l'application-cible. En effet, quand une action d'assistance a déjà été déclenchée, et si l'utilisateur se retrouve dans la même situation problématique quelques minutes après, il peut être pertinent de déclencher une autre action d'assistance puisque la première action d'assistance ne semble pas avoir répondu efficacement au besoin de l'utilisateur. Il est également possible de spécifier le nombre de fois qu'une action ou une règle d'assistance peut être déclenchée, ainsi que la durée entre deux déclenchements de l'action ou de la règle.

Les conditions sur l'état de l'application-cible permettent la personnalisation de l'assistance en fonction des informations relatives à l'application-cible : est-ce que le bouton X est actif ? Quel item est

sélectionné dans la liste déroulante Y ? Est-ce que la case à cocher Z est sélectionnée ?

Les actions d'assistance

Nous avons proposé une typologie de l'assistance aux utilisateurs d'applications informatiques (Ginon, et al., 2013), qui s'appuie sur un état de l'art du domaine et sur une étude des systèmes existants. Cette typologie confronte les besoins d'assistance des utilisateurs aux différents moyens et approches d'assistance qui permettent de répondre à ces besoins. Nous avons identifié quatre moyens d'assistance : les messages (comme l'affichage de conseils ou de recommandations, ou la lecture d'instructions), les exemples (comme les vidéos de démonstrations), la modification de l'interface de l'application-cible (comme la mise en valeur de composants), et la création automatisée qui permet de réaliser tout ou partie de la tâche pour l'utilisateur. Pour ces quatre moyens d'assistance, notre typologie montre qu'il existe plusieurs façons de les mettre en œuvre. Nous faisons l'hypothèse qu'un système d'assistance sera plus efficace et mieux accepté par les utilisateurs s'il dispose de plusieurs approches d'assistance. En effet, cela rend possible une plus grande adaptation de l'assistance aux spécificités de l'utilisateur, notamment de ses préférences.

Dans notre modèle générique d'assistance aux utilisateurs, les actions d'assistance sont réalisées par des assistants épiphytes (cf. © Fig. 2), à la demande du moteur d'assistance. Ce choix d'une assistance épiphyte découle de notre volonté de rendre possible l'exécution d'un système d'assistance dans une application-cible existante sans nécessiter aucune modification dans cette application. Nous considérons en priorité les assistants épiphytes génériques, c'est-à-dire pouvant être greffés sur n'importe quelle application.

Quand le concepteur de l'assistance définit une action, il doit spécifier le type de l'action. S'il le souhaite, le concepteur peut également spécifier quel assistant épiphyte réalisera l'action. Dans tous les cas, le concepteur doit préciser certaines informations relatives à l'action. Ces informations dépendent du type de l'action. Le concepteur peut également préciser certaines informations complémentaires pour une action d'assistance. Par exemple, pour un message d'assistance, le concepteur doit spécifier au minimum le contenu de ce message, mais il peut également donner des précisions de mise en forme du message. Si, pour une action d'assistance, le concepteur ne précise pas toutes les

informations possibles (comme la police dans le cas d'un message d'assistance textuel), alors les informations manquantes seront déterminées par le moteur d'assistance au moment de l'exécution.

3.2 Exécution d'un système d'assistance

L'exécution d'un système d'assistance dans une application-cible existante sans modification de celle-ci peut être réalisée en trois principales étapes (cf. Fig. 2) : la surveillance de l'application-cible, l'identification d'un besoin d'assistance et l'élaboration d'une réponse adaptée à ce besoin. Ces trois étapes sont gérées par le moteur d'assistance qui utilise pour cela la description de l'interface de l'application-cible, la description de l'assistance souhaitée par le concepteur et le profil de l'utilisateur. Le moteur d'assistance exploite également un ensemble de collecteurs d'informations qui surveillent l'application-cible, ainsi qu'un ensemble d'assistants épiphytes capables de réaliser des actions d'assistance dans l'application-cible à la demande du moteur d'assistance.

Détection d'événements dans l'application-cible

Afin de fournir une assistance personnalisée à l'utilisateur de l'application-cible, il est nécessaire de posséder certaines informations sur les événements ayant lieu dans l'application-cible, tels que les actions de l'utilisateur et l'état de l'application-cible. Pour cela, des collecteurs d'informations surveillent l'application-cible.

Nos collecteurs s'appuient sur des bibliothèques d'accessibilité (prévues pour permettre qui permettent par exemple l'utilisation d'un lecteur d'écran) de surveiller les interactions entre l'utilisateur et l'interface d'une application sans avoir à la modifier. Il n'est pas nécessaire que l'application-cible soir conçue pour la collecte de traces, mais seulement que son interface soit visible par une bibliothèque d'accessibilité. Après avoir détecté un événement dans l'application-cible, le collecteur d'informations identifie le composant concerné grâce à la description de l'interface de l'application-cible. Ainsi, un événement détecté est un triplet <type d'événement, identifiant du composant concerné, date et heure>. Les collecteurs sont également utilisés pour connaitre l'état de l'application. Ils permettent par exemple de savoir quel item est sélectionné dans une liste déroulante ou quel texte est contenu dans une zone de saisie. Chaque événement

détecté par les collecteurs sera transmis au kTBS (Zarka, et al., 2013), un système capable de stocker des traces faites à partir d'événements bas niveau, mais aussi de les transformer en événements de plus haut niveau (« répondre à un mail », « écrire une lettre ») si on lui fournit les connaissances appropriées.

Identification d'un besoin d'assistance.

Quand un événement est détecté dans l'application-cible, le moteur d'assistance recherche dans la description de l'assistance souhaitée quelles sont les règles d'assistance associées à cet événement déclencheur (cf. Ⓑ Fig. 2). Si une règle d'assistance est trouvée, le moteur d'assistance doit vérifier si ses conditions de déclenchement sont vérifiées. Cette étape correspond à l'identification d'un besoin d'assistance chez l'utilisateur de l'application-cible, d'après la description de l'assistance spécifiée par le concepteur de l'assistance qui exploite les quatre types de conditions présentés en section 3.1.

Afin d'établir si une condition sur les actions de l'utilisateur est vérifiée, le moteur d'assistance consulte le kTBS. Par exemple, pour la condition « l'utilisateur n'a réalisé aucune action depuis 5 minutes », le kTBS vérifie que les collecteurs ne lui ont pas transmis d'événements liés à l'action de l'utilisateur depuis 5 minutes.

Afin d'établir si une condition sur le profil de l'utilisateur est vérifiée, le moteur d'assistance utilise le modèle cPMDLe. Par exemple, si une condition concerne une valeur numérique du profil de l'utilisateur, comme un pourcentage, le moteur d'assistance vérifie que la valeur correspondante dans le profil de l'utilisateur appartient à l'intervalle spécifié.

Afin d'établir si une condition sur l'historique de l'assistance est vérifiée, le moteur d'assistance consulte le kTBS. Par exemple, pour la condition « la règle d'assistance R ne doit pas avoir été déclenchée depuis 15 minutes », le moteur d'assistance demande au kTBS la date et l'heure du dernier déclenchement de R.

Afin d'établir si une condition sur l'état de l'application-cible est vérifiée, le moteur d'assistance fait appel aux collecteurs d'informations. Par exemple, si une condition est « l'utilisateur a sélectionné le mode d'affichage graphique », et que ce mode peut être sélectionné dans la liste déroulante 48, alors le moteur d'assistance demande aux collecteurs quel item est sélectionné dans cette liste.

Élaboration d'une réponse adaptée au besoin

Quand l'événement déclencheur associé à une règle d'assistance est détecté et si les conditions de déclenchement de la règle sont vérifiées, alors le moteur d'assistance fait appel à un assistant épiphyte pour réaliser la ou les actions d'assistance associées à cette règle afin de répondre au besoin de l'utilisateur. Nous avons vu que lorsque le concepteur de l'assistance crée une action d'assistance, il doit spécifier le type de l'action, ainsi que certaines informations complémentaires. Si le concepteur a précisé quel assistant épiphyte doit réaliser l'action, le moteur d'assistance fait spécifiquement appel à cet assistant épiphyte pour réaliser l'action. Cependant, si le concepteur n'a pas imposé d'assistant épiphyte, ou si cet assistant n'est pas disponible (par exemple s'il n'est pas installé sur l'ordinateur de l'utilisateur), le moteur d'assistance doit sélectionner parmi les assistants épiphytes disponibles celui qui est le plus adapté pour réaliser l'action d'assistance. Pour faire ce choix, le moteur prend en compte les informations spécifiées par le concepteur de l'assistance, et les spécificités de l'utilisateur. Ainsi, le choix de l'assistant épiphyte permet également de personnaliser l'assistance.

De plus, l'action d'assistance elle-même peut être personnalisée pour l'utilisateur. Par exemple, dans le cas d'un message d'assistance, le concepteur de l'assistance peut spécifier que le contenu du message contiendra des informations issues du profil de l'utilisateur, comme son nom ou son niveau de maîtrise d'une tâche. Dans ce cas, le moteur d'assistance doit trouver ces informations dans le profil de l'utilisateur avant de demander à un assistant épiphyte de l'afficher. Un message d'assistance pourra ainsi être « Salut Lucas, souviens-toi que la dernière fois tu as fait 3 fautes dans cet exercice. Est-ce que tu veux de l'aide pour la suite ? ».

4. Première implémentation de notre modèle

Nous avons partiellement implémenté notre modèle générique d'assistance aux utilisateurs d'applications informatiques dans un prototype opérationnel comportant un éditeur d'assistance à destination des concepteurs d'assistance et un moteur générique d'assistance qui exécute les systèmes d'assistance définis dans leur application-cible.

4.1 Éditeur d'assistance

Nous avons développé un éditeur d'assistance qui met en œuvre la phase de spécification de systèmes d'assistance présentée en section 3.1. Cet éditeur permet aux concepteurs de définir un système d'assistance personnalisée pour une application-cible. Le concepteur de l'assistance peut être le créateur de l'application-cible, ou un utilisateur expert : aucune compétence en programmation n'est requise et le code source de l'application-cible n'est pas utilisé.

```
<Interface>
▼<window text="Bienvenue dans Regards" type="JFrame" id="1">
 ▼<component text="null" type="JRootPane" id="2">
    <component text="null" type="JPanel" id="3"/>
   ▼<component text="null" type="JLayeredPane" id="4">
    ▼<component text="null" type="JPanel" id="5">
       <component text="Regards" type="JLabel" id="6"/>
       <component text="Definir une représentation" type="JButton" id="7"/>
       <component text="Modifier une représentation" type="JButton" id="8"/>
       <component text="Definir une vue" type="JButton" id="9"/>
       <component text="Modifier une vue" type="JButton" id="10"/>
       <component text="Modifier une séance d'activités" type="JButton" id="11"/>
    Ⓐ <component text="Definir une séance d'activité" type="JButton" id="12"/>
       <component text="Regards vous permet de préparer et personnaliser les visualisations
         des profils par les différents types d'utilisateurs." type="JLabel" id="13"/>
       <component text="Quitter (Retour à Eprofiléa)" type="JButton" id="14"/>
     </component>
    </component>
   </component>
 </window>
 ...
</Interface>
```

Fig. 3 – Description de l'interface de l'application Regards

Pour définir un système d'assistance, l'étape de description de l'interface est automatisée : le concepteur doit seulement lancer l'application-cible et ouvrir toutes ses fenêtres. L'arbre des composants de l'application-cible est alors créé automatiquement et un identifiant est associé à chaque composant. La Fig. 3 montre l'exemple de cette description pour l'application Regards (Ginon, et al., 2011).

Une fois cette description effectuée, le concepteur passe à l'étape principale, celle de définition des règles d'assistance. Le concepteur y crée les conditions de déclenchement qu'il souhaite utiliser pour personnaliser l'assistance, ainsi que les actions d'assistance qui répondront aux besoins des utilisateurs de l'application-cible. Une copie d'écran de notre éditeur d'assistance présentant cette étape est donnée en Fig. 4. L'écran est divisé en quatre parties. La partie supérieure-gauche (cf. Ⓐ Fig. 4) permet la création d'une nouvelle règle d'assistance. La partie supérieure-droite (cf. Ⓑ Fig. 4) présente les règles d'assistance déjà créées. Finalement, les parties inférieure-

gauche et inférieure-droite présentent respectivement les conditions de déclenchement créées (cf. © Fig. 4) et les actions d'assistance créées (cf. ® Fig. 4). Pour l'instant, notre éditeur permet la définition de conditions de déclenchement sur le profil de l'utilisateur, et la définition d'action d'assistance de type message, exemple et mise en valeur.

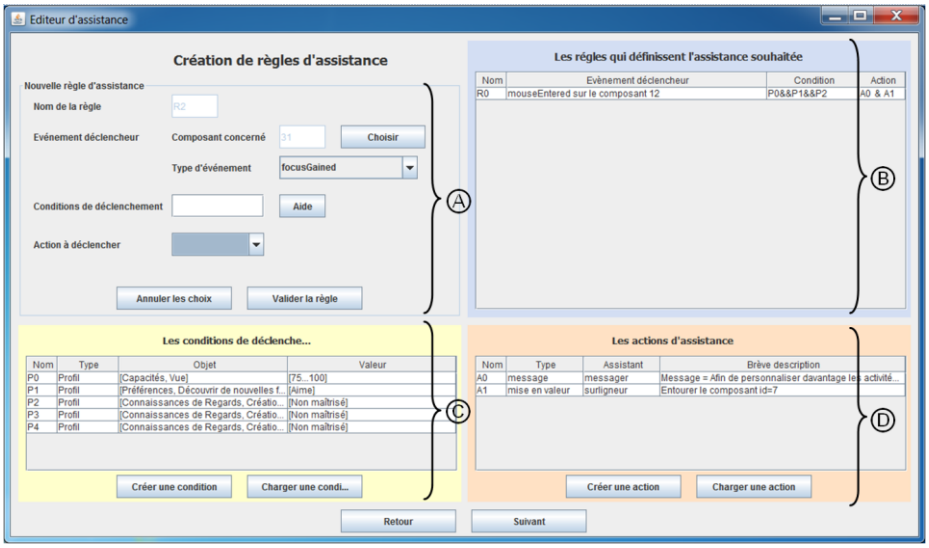

Fig. 4 – Copie d'écran de notre éditeur d'assistance

La Fig. 5 présente une description de l'assistance pour l'application Regards définie à l'aide de notre éditeur d'assistance (cf. Fig. 4). On peut y voir que la règle d'assistance R0 (cf. ① Fig. 5) est déclenchée par le passage de la souris sur le composant 12, qui correspond au bouton « définir une séance d'activités » (cf. Ⓐ Fig. 3). Cette règle est exécutée uniquement si les conditions P0, P1 et P2 sur le profil de l'utilisateur sont vérifiées. Ces conditions signifient respectivement que l'utilisateur a une vue comprise entre 75 et 100% (P0 : cf. ② Fig. 5), qu'il aime découvrir de nouvelles fonctionnalités (P1 : cf. ③ Fig. 5) et qu'il ne maîtrise pas la fonctionnalité de Regards « Création de représentations » (P2 : cf. ④ Fig. 5). Si ces conditions sont vérifiées, les actions d'assistance A0 et A1 sont déclenchées. A0 affiche dans une fenêtre un message qui conseille à l'utilisateur de découvrir la fonctionnalité « Création de représentations », et A1 encercle en vert le bouton 7 « définir une représentation ».

```
<required_assistance> <metadata>...</metadata>
 <rules>
  <rule id="R0"><metadata>...</metadata>
   <trigger_event type="mouseEntered" idcomp="12"/>
   <condition formula="P0andP1andP2"/>
   <action id_1="A0" id_2="A1"/>
  </rule>···
 </rules>
 <actions>
  <action id="A0" type="message">
   <text> Afin de personnaliser d'avantage les activités sur les profils,
          vous pouvez créer vos propres représentations d'éléments. </text>
  </action>
  <action id="A1" type="enhancement">
   <component id="7" type="surround" color="[r=0,g=0,b=255]" curved="10" distance="5"/>
  </action> ···
 </actions>
 <conditions>
  <P id="P0"> ② <numeric element="[Capacités, Vue]" inf="75" sup="100"/> </P>
  <P id="P1"> ③
   <textual element="[Préférences, Découvrir de nouvelles fonctionnalités]">
    <value>Aime</value> </textual> </P>
  <P id="P2"> ④
   <textual element="[Connaissances de Regards, Création de représentations]">
    <value>Non maîtrisé</value> </textual> </P>···
 </conditions> </required_assistance>
```

① (braces the rule block)

Fig. 5 – Description de l'assistance souhaitée pour l'application Regards

4.2 Moteur générique d'assistance

Nous avons développé un premier prototype opérationnel de moteur d'assistance qui met en œuvre la phase d'exécution d'un système d'assistance présentée en section 3.2. Ce moteur d'assistance exploite le fichier de description de l'assistance préalablement définie pour l'application-cible et fournit une assistance personnalisée pour les utilisateurs de cette application.

Pour compléter notre moteur d'assistance, nous avons développé deux collecteurs d'informations afin de permettre la détection d'événements dans une application-cible, sans avoir à la modifier. Le premier collecteur s'appuie sur UIAutomation (Haverty, 2005), il est dédié aux exécutables Windows. Le second collecteur s'appuie sur JavaAccessibility (Harper, et al., 2005), il est dédié aux applications développées en langage Java.

Nous avons également développé plusieurs assistants épiphytes capables d'exécuter des actions à la demande du moteur d'assistance. Ces assistants épiphytes peuvent réaliser trois types d'actions : messages, exemples et modifications de l'interface de l'application-cible. Les messages peuvent être affichés dans une fenêtre pop-up, lus par une synthèse vocale ou présentés par un agent animé (l'agent peut afficher le message dans un bulle et le lire, le message peut être

accompagné de gestes et d'animations). Les exemples peuvent être présentés sous forme textuelle, graphique ou vidéo. Enfin, la modification de l'interface de l'application-cible concerne pour l'instant la mise en valeur de composants : un composant peut être entouré, coloré ou désigné par un agent animé.

5. Premières évaluations de notre approche

Nous avons testé les outils présentés dans cet article en spécifiant puis exécutant des systèmes d'assistance dans une dizaine d'applications-cibles variées. À titre d'exemple simple, la Fig. 6 montre l'exécution de la règle d'assistance R0 (cf. Ⓐ Fig. 5) qui suggère à l'utilisateur de découvrir la fonctionnalité de Regards permettant de créer de nouvelles représentations.

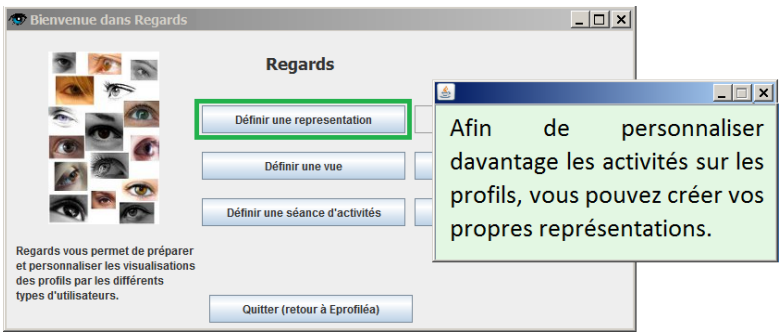

Fig. 6 – Exécution d'une règle d'assistance dans l'application Regards

Dans le but d'évaluer les parties description de l'interface d'une application-cible et surveillance d'une application-cible, nous avons testé avec succès nos collecteurs basés sur JavaAccessibility et UIAutomation sur plus de 50 applications-cibles[1], des plus simples, comme la calculatrice Windows, aux plus complexes, comme l'IDE NetBeans. Nous sommes actuellement en train de terminer le développement d'un collecteur dédié aux applications web, afin de permettre l'adjonction de système d'assistance dans le plus grand nombre d'applications-cibles.

Nous avons ensuite réalisé une première mise à l'essai de notre approche dédiée à la spécification d'un système d'assistance, présentée

[1] La liste de ces applications est disponible en http://liris.cnrs.fr/blandine.ginon/detection.html

en section 3.1. Nous avons demandé à des étudiants de Master informatique d'endosser le rôle de concepteur d'assistance en étudiant deux logiciels pédagogiques simples, créés par des étudiants de la promotion précédente. Les étudiants devaient concevoir sur papier un système d'assistance à destination des utilisateurs de ces logiciels (des élèves de primaire), puis le spécifier avec notre éditeur d'assistance : ils disposaient pour cela de deux séances pour un total de 2 h 30 de travail.

Les étudiants, répartis en 20 binômes, ont réussi à concevoir 20 systèmes d'assistance : pour cela, ils ont créé 134 règles d'assistance, 81 conditions de déclenchement et 158 actions d'assistance. La Fig. 7 présente la répartition par type des conditions de déclenchement (à gauche) et des actions d'assistance (à droite). Le nombre de conditions de déclenchement (81 conditions pour 134 règles d'assistance) montre la volonté des étudiants de personnaliser l'assistance. Parmi les actions d'assistance, les messages dominent largement, probablement en raison de leur simplicité et parce que les étudiants les ont jugés adaptés pour répondre aux besoins des utilisateurs.

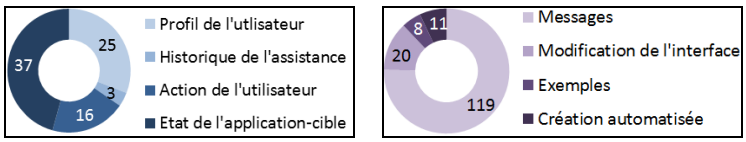

Fig. 7 – Utilisation des conditions de déclenchement et des actions d'assistance

6. Conclusion et perspectives

Dans cet article, nous avons présenté notre modèle générique d'assistance aux utilisateurs d'applications informatiques. Il permet la spécification de systèmes d'assistance personnalisée pour n'importe quelle application-cible dont on ne possède pas nécessairement le code source. Ces systèmes d'assistance sont définis par un concepteur d'assistance qui n'est pas forcément le créateur de l'application-cible, aucune compétence en programmation n'étant requise. L'assistance est exécutée par un moteur d'assistance utilisant des collecteurs d'informations qui surveillent l'application-cible, ainsi que des assistants épiphytes qui réalisent des actions d'assistance dans l'application-cible sans perturber son fonctionnement. Nos assistants épiphytes sont génériques : ils peuvent être utilisés dans n'importe

quelle application-cible. Les deux caractéristiques principales de notre approche sont donc sa généricité et sa capacité à personnaliser l'assistance.

Nous avons implémenté notre modèle dans un premier prototype opérationnel comportant un éditeur d'assistance à destination des concepteurs et un moteur d'assistance, ce qui montre la faisabilité de notre approche. Pour le moment, notre moteur d'assistance dispose d'un ensemble d'assistants épiphytes capables de réaliser des actions de type message, exemple ou modification de l'interface. Nous travaillons actuellement au développement d'autres assistants épiphytes afin d'enrichir les possibilités de notre moteur d'assistance. Nous avons réalisé de premiers tests afin d'étudier la couverture de notre approche et une première mise à l'essai testant l'utilisabilité de notre éditeur d'assistance.

Nous souhaitons maintenant améliorer les capacités de notre moteur d'assistance à élaborer une réponse adaptée aux besoins de l'utilisateur, notamment pour gérer des règles contradictoires ou redondantes. Plus le moteur d'assistance sera capable d'exploiter automatiquement les informations contextuelles (profil de l'utilisateur, historique de l'assistance…) plus la tâche de spécification de l'assistance sera facilitée. Plus généralement, nous souhaitons mieux définir la tâche du concepteur d'assistance, qui est une tâche complexe relavant également de l'ingénierie des connaissances. Nous pourrons ensuite expérimenter notre approche et nos outils en situation réelle.

Références

CARLIER, F. & RENAULT, V. (2010). Educational webportals augmented by mobile devices with iFrimousse architecture. In *International Conference on Advanced Learning Technologies*, Sousse, Tunisia.

DUFRESNE, A. (2001). Conception d'une interface adaptée aux activités de l'éducation à distance - ExploraGraph. In *Sciences et technologies éducatives*, p 301-319.

GALLUCCIO, R. G. P. (2006). Humanizing CALL: The use of pedagogical agents as language tutors. In *New England Regional Association for Language Learning Technology*.

GINON, B. & JEAN-DAUBIAS, S. (2011). Models and tools to personalize activities on learners profiles. In *Ed-Media, Portugal*.

GINON, B., JEAN-DAUBIAS, S. & CHAMPIN, P.-A. (2013). Une typologie de l'assistance : exemple d'application aux EIAH. Rapport de recherche LIRIS.

HARPER, S., KHAN, G. & STEVENS, R. (2005). Design Checks for Java Accessibility. In *Accessible Design in the Digital World, Scotland*.

HAVERTY, R. (2005). New accessibility model for Microsoft Windows and cross platform development. In *ACM SIGACCESS Accessibility and Computing*, p 11-17.

JEAN-DAUBIAS, S., GINON, B. & LEFEVRE, M. (2011). Modèles et outils pour prendre en compte l'évolutivité dans les profils d'apprenants. In *Sticef* p 99-134.

LEFEVRE, M., JEAN-DAUBIAS, S. & GUIN, N. (2012). An approach for unified personalization of learning. In *Workshop PALE of User Modeling Adaptation and Personalization, Montreal, Canada*.

PAQUETTE, G. (2012). Référencement par compétence, recherche et assistance dans les environnements d'apprentissage et de travail. In *TICE*, p 190-199, Lyon, France.

PAQUETTE, G., PACHET, F., GIROUX, S. & GIRARD, J. (1996). EpiTalk, a generic tool for the development of advisor systems. In *IJAIED*, p 349-370.

RANDALL, N. & TCHOUNIKINE, I. (1998). Enhancing the Adaptivity of an Existing Website with an Epiphyte Recommender System. In *New Review of Hypermedia and Multimedia, Vol. 10(1)*, p 31-52.

RICHARD, B. & PEDERSEN, P. (2004). Who exactly is trying to help us? the ethos of help systems in popular computer applications. In *international conference on Computer documentation, Canada*, p 63-69.

ZARKA, R., CHAMPIN, P.-A., CORDIER, A., EGYED-ZSIGMOND, E., LAMONTAGNE, L. & MILLE, A. (2013). TStore: A Trace-Base Management System using Finite-State Transducer Approach for Trace Transformation. In *MODELSWARD, Barcelona, Spain*.