Projet LDTP (Linux Desktop Testing Project, ou projet de test du bureau Linux)

J. Premkumar

Programmer des interfaces utilisateur est un travail long et périlleux : ce n'est qu'au cas par cas que vous réalisez qu'un clic sur le bouton X ne produit pas l'action Y escomptée, et ainsi de suite. D'où la nécessité de disposer d'un cadre de tests comme le projet LDTP, capable de contrôler l'exactitude de votre application, afin de vous laisser plus de temps pour le développement...

Le projet LDTP (Linux Desktop Testing Project) vise à produire un cadre de tests automatisés d'excellente qualité ainsi que des outils à la pointe de la technologie permettant de tester le bureau Linux pour pouvoir éventuellement l'améliorer. Il s'agit ici d'une initiative libre, dont le code source est distribué sous la licence GPL/LGPL.

Test : état des lieux

Dans un marché de plus en plus compétitif, chaque société informatique tente de privilégier la gestion qualité de leur logiciel, en utilisant, notamment, diverses méthodologies de tests comprenant généralement de nombreuses tâches répétitives. Le cas du test de santé est l'exemple le plus connu et le plus évident. Une fois les jeux de tests de santé développés, ils sont alors exécutés plusieurs fois (disons après chaque intégration d'une étape de développement), afin de s'assurer que les fonctionnalités élémentaires du logiciel ne soient pas affectées après chaque étape du développement.

Le test est une tâche réellement stimulante et intéressante, plus particulièrement pendant la phase de développement d'un jeu d'essais. En revanche, ce n'est plus du tout le cas lorsqu'il s'agit de tester de manière répétitive les logiciels. C'est ici qu'entre en jeu l'automatisation.

En quoi le projet LDTP peut vous aider ?

Le projet LDTP est un cadre de tests automatisés d'excellente qualité, permettant d'exécuter des jeux d'essais de manière automatique afin de contrôler la fonctionnalité du logiciel testé. Qu'il s'agisse de tests fonctionnels ou structurels, une fois les jeux d'essais développés, ils doivent être rédigés sous forme de scripts de test. Est désignée par script de test, une séquence d'évènements générés par l'utilisateur comme un clic sur un bouton, entrer du texte dans une boîte de texte, etc.

Sur l'auteur :

Ingénieur en solutions logicielles, J. Premkumar est diplômé en Informatique et Ingénierie. Il participe activement au projet LDTP, et fait partie de l'équipe centrale du projet. Passionné de technologies réseau, il consacre son temps libre à des jeux d'extérieur comme le criquet, ou le ping-pong, etc...

Selon le script de test, ces évènements sont automatiquement générés lors de l'exécution au moyen d'interfaces de programmation fournies par le serveur LDTP.

Technologie sous-jacente

Le projet LDTP fait appel au démon AT-SPI pour la génération automatique des évènements utilisateur pendant l'exécution. Acronyme de Assistive Technology – Service Provider Interface (Technologie d'Aide – Interface Fournisseur de Services), le démon AT-SPI fait parti du projet Accessibilité GNOME. Il se charge d'exposer les interfaces de programmation afin de générer des évènements utilisateur comme les évènements souris ou clavier, par exemple. Les liens pour le langage C de ces interfaces de programmation sont proposés par la bibliothèque AT–SPI, utilisée à son tour par le projet LDTP.

Certains pourraient se demander pourquoi ne pas utiliser directement les interfaces de programmation du projet Accessibilité ? Ce procédé rendrait en réalité la programmation des jeux d'essais bien plus difficile : il faudrait bien maîtriser le langage C, la boîte à outils d'accessibilité ainsi que les bibliothèques AT-SPI. A l'inverse, grâce à LDTP, il est possible de rédiger des jeux d'essais très facilement en se référant simplement à l'excellente documentation de l'interface de programmation. La plupart des interfaces de programmation sont plutôt directes et leur utilisation est facilement mémorisable. Ainsi, il est possible de développer des jeux d'essais puissants en un laps de temps très court. Par dessus tout, LDTP tient compte de la complexité inhérente au traitement des objets d'accessibilité pendant l'exécution.

Architecture LDTP

Le cadre LDTP observe une approche client/serveur. Le client et le serveur communiquent entre eux au moyen du protocole CTP (Command Transfer Protocol, ou protocole de transfert de commandes) de LDTP. Le protocole CTP permet de définir les requêtes client et les réponses du serveur au format XML. Une requête client correspond à un évènement utilisateur comme un clic sur un bouton, par exemple. Les réponses du serveur désignent les divers résultats de l'exécution de l'opération requise par le client.

Les fonctionnalités des divers composants exposés dans la Figure 1 sont les suivantes :

1. Le client LDTP prend en entrée le script de test et contacte le serveur LDTP pour l'exécution d'opérations utilisateur individuelles, comme un clic de souris, etc.
2. Le serveur LDTP répond aux requêtes du client LDTP en invoquant le gestionnaire de composants de l'interface graphique correspondante sur l'élément visé par l'action.
3. Le gestionnaire client s'efforce de gérer plusieurs clients de manière simultanée.
4. Le gestionnaire des composants communique avec la couche AT-SPI afin d'exécuter l'opération requise via l'interface d'accessibilité.
5. Le gestionnaire des évènements se charge de gérer les fenêtres / boîtes de dialogue non désirées pendant le test d'une application. Il est possible de s'inscrire afin de recevoir les notifications de tels évènements via le mécanisme de rappel.
6. L'enregistreur chronologique sauvegarde les résultats de l'opération exécutée.

Rédiger des applications avec LDTP

LDTP repose sur des scripts de tests, également désignés sous le terme de scripts d'automatisation. Ces scripts contiennent une séquence d'appels vers les interfaces de programmation de LDTP correspondant aux évènements utilisateur censés exécuter la fonctionnalité pour laquelle ils sont testés. Nous avons exposé dans le Listing 1 un extrait de script d'automatisation pour Python, visant à

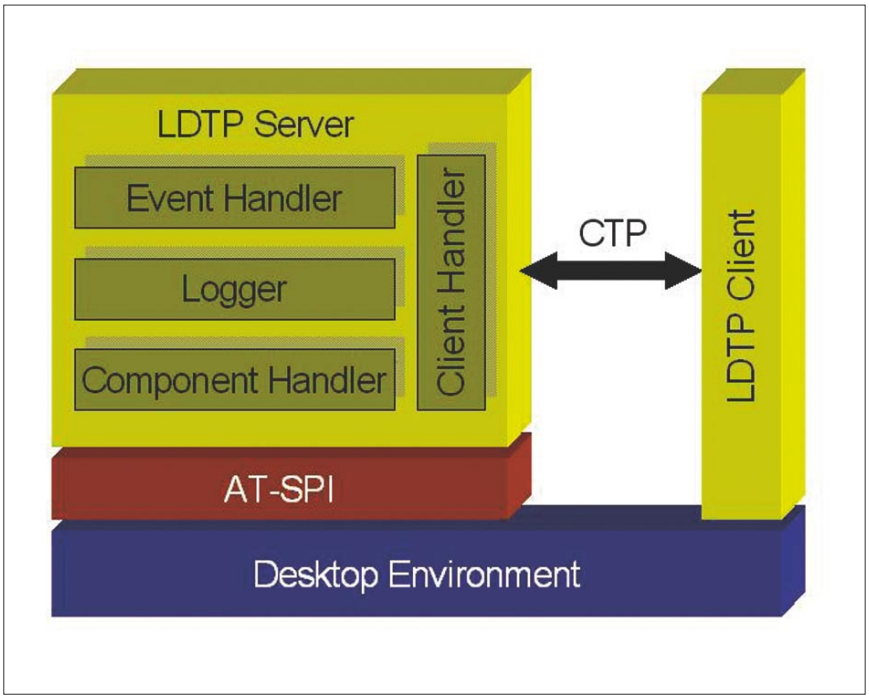

Figure 1. *Diagramme représentant l'architecture de LDTP*

créer un nouveau fichier sous Gedit au moyen de LDTP.

En observant ce script, vous comprendrez mieux l'utilisation de LDTP. Les mots clés `log`, `selectmenuitem`, `settextvalue`, `waittillguiexist`, etc..., représentent les interfaces de programmation exposées par LDTP. Les noms des interfaces de programmation sont explicites et correspondent à leur fonction ; par exemple, l'interface de programmation intitulée `selectmenuitem` est chargée de sélectionner un élément du menu dans la fenêtre ouverte selon le nom de fenêtre donné. La documentation du projet contient une liste exhaustive de toutes les interfaces de programmation disponibles, ainsi que des exemples de leur utilisation. Ainsi, l'outil LDTP comporte des fonctions pour les menus, des boîtes de contrôle, des options, des boutons ainsi qu'un grand nombre d'autres éléments spécifiques d'une interface utilisateur.

Toujours d'après le Listing 1, vous constaterez la présence intéressante d'arguments passés dans les interfaces de programmation de LDTP, plus particulièrement le premier et le second. Ces deux arguments permettent d'identifier l'objet de l'interface graphique sur lequel l'action doit être pratiquée. Par exemple, dans la déclaration ci-dessous, le premier argument représente le nom de la fenêtre dans laquelle se trouve l'objet de l'interface graphique, alors que le second argument l'élément du menu sur lequel il faut cliquer :

```
selectmenuitem ('frmgedit',
   'mnuFile;mnuNew')
```

Carte d'application, ou comment faciliter l'accessibilité aux objets

LDTP cherche le nom des objets, comme par exemple `frmgedit` ou `mnuFile` dans la carte de l'application. Est désignée par carte d'application une description détaillée de l'application en question et de tous les éléments de son interface graphique, permettant au script de test de les gérer pendant la durée de l'exécution. Par exemple (extrait tiré du site Web du projet) :

```
[gedit]
gedit={class=frame, app_name=gedit}
mnuFile={class=menu, label=File}
mnuNew={class=menu_item, label=New}
mnuOpen={class=menu_item, label=Open...}
```

Listing 1. *Script d'automatisation pour la création d'un nouveau fichier sous Gedit au moyen de LDTP*

```
# Création et sauvegarde d'un nouveau document
from ldtp import *
from ldtputils import *
import string, sys, commands, time, filecmp
import re, os

default_dir = os.getcwd ()
default_image_dir = default_dir + '/images'
default_doc_dir = default_dir + '/doc'
default_tmp_dir = default_dir + '/tmp'
gedit_exe_path = 'gedit'

if os.access (default_tmp_dir, os.F_OK | os.R_OK | os.W_OK | os.X_OK) == 0:
  os.mkdir (default_tmp_dir)

log ('Create New Document', 'teststart')

try:
    try:
        # Fermeture de tous les onglets ouverts
        selectmenuitem ('*-gedit', 'mnuDocuments;mnuCloseAll')
    except error:
        log ('There maybe no documents opened', 'info')

    selectmenuitem ('*gedit', 'mnuFile;mnuNew')
    settextvalue ('*gedit', 'txt0', 'Testing gedit using Linux Desktop Testing
                  Project')

    selectmenuitem ('*gedit', 'mnuFile;mnuSaveAs')

    # Attendre pendant 3 secondes jusqu'à l'apparition de la fenêtre de dialogue
                  de sauvegarde
    time.sleep (3)

    # Contrôler la fenêtre de dialogue
    if waittillguiexist ('dlgSaveas...') == 1:
        settextvalue ('dlgSaveas...', 'txtName', default_tmp_dir + '/
                      sample.txt')
        click ('dlgSaveas...', 'tbtnBrowseforotherfolders')
        selectrowindex ('dlgSaveas...', 'tblShortcuts', 0)
        click ('dlgSaveas...', 'btnSave')
        time.sleep (3)
        if guiexist ('dlgQuestion') == 1:
            click ('dlgQuestion', 'btnReplace')
        mo = re.match (os.path.expandvars ('$HOME'), default_tmp_dir)
        if str(mo) != "<type 'NoneType'>":
            newcontext = '~' + default_tmp_dir [mo.end():]
        else:
            newcontext = default_tmp_dir
        waittillguinotexist ('dlgQuestion')
        waittillguinotexist ('dlgSaveas...')
        time.sleep (5)
        selectmenuitem ('*gedit', 'mnuDocuments;mnuCloseAll')
        log ('Create New Document', 'pass')
    else:
        log ('Save dialog does not appear', 'error')
        log ('Create New Document', 'fail')
except error, msg:
    log (str (msg), 'error')
    log ('Create New Document', 'fail')

log ('Create New Document', 'testend')
```

```
mnuOpenLocation={class=menu_item, label=
  Open Location...}
```

Dans cet extrait de carte, nous commençons par définir l'application (`gedit`), son menu *File* (`mnuFile`), et ses éléments *New* (`mnuNew`) et *Open* (`mnuOpen`).

Cette carte est générée par un outil appelé APPMAP, compris dans le package LDTP. Vous pouvez toutefois le créer de manière dynamique, lors de l'exécution de l'application, au moyen de la fonction `remap()`, qui prend les deux arguments suivants : le nom de l'application et le nom de l'objet :

```
remap ('<application-name>',
   '<dialog name>')
```

Un objet cartographié dynamiquement peut alors être ôté de la carte au moyen de la fonction `undoremap()` :

```
undoremap ('<application-name>',
   '<dialog name>')
```

La recherche d'objets dans la carte demeure très flexible : vous pouvez utiliser des expressions régulières. Afin d'identifier le nom d'une fenêtre dans l'extrait de notre application au moyen d'une expression régulière, il est possible d'utiliser la déclaration suivante, par exemple :

```
selectmenuitem ('*gedit',
   'mnuFile;mnuNew')
```

Dans le lien vers Python actuellement disponible pour les interfaces de programmation de LDTP, un échec d'exécution d'une interface de programmation LDTP dans le script d'automatisation sera remarqué par des exceptions émises par le client LDTP, selon les réponses XML reçues du serveur LDTP.

Principaux avantages de l'outil LDTP

1. Une excellente documentation de référence est disponible. Cette documentation couvre l'ensemble des aspects du projet LDTP, qu'il s'agisse de l'explication de son architecture ou des références aux interfaces de programmation.
2. LDTP propose des interfaces de programmation chargée de contrôler chaque évènement utilisateur (atomique) au cas par cas, comme les clics de bouton.
3. Les éléments de l'interface graphique créés lors de l'exécution sont manipulables grâce à la carte de l'application générée de manière dynamique, et chargée de situer l'objet souhaité lors de l'exécution de l'application.
4. Il est possible de rechercher des éléments de l'interface graphique au moyen d'expressions régulières.
5. LDTP propose un support destiné à la gestion des boîtes de dialogues et des fenêtres inattendues au moyen de mécanismes de rappel.
6. Extrêmement puissants, les scénarios de tests permettent de réutiliser des scripts de tests atomiques. Cette exécution de scripts de test regroupés vous permet de contrôler des séquences de jeux d'essais d'après la réussite ou l'échec des scripts ou des scénarios de tests précédents.
7. Les résultats des scénarios de test exécutés sont enregistrés au format XML, permettant de générer ultérieurement des rapports de tests selon vos besoins. Veuillez consulter l'extrait de code XSLT dans le dossier Exemples disponible avec les fichiers source. Ce code XSLT transforme les journaux générés au format XML en page HTML chargée d'afficher le rapport de test sous forme de tableau.
1. LDTP permet également de tester des applications localisées dans d'autres langues sans avoir à modifier les scripts de tests. Veuillez consulter la documentation pour plus d'informations sur l'activation de la localisation sous LDTP.
2. GNU/LDTP peut tester n'importe quelle application GNOME dont l'accessibilité est activée, toutes les applications Mozilla, Openoffice.org, et Java (à condition de posséder une interface utilisateur à bascule), ainsi que les applications KDE 4.0 basées sur QT 4.0 (reposant sur les sorties presse de KDE).

Comment automatiser les tests grâce au cadre d'application LDTP ?

Nous avons résumé dans cette section les cinq étapes fondamentales, nécessaires au test de votre application avec LDTP :

1. Concevoir les jeux d'essais destinés à tester votre application,
2. Vous pourrez également choisir prochainement le langage de programmation pour rédiger vos scripts de tests. A l'heure actuelle, LDTP ne supporte que Python. Le langage Mono devrait venir s'ajouter à la liste très prochainement. La communauté LDTP s'engage à fournir rapidement un support pour les autres langages de programmation comme Java, Perl, etc.,
3. Convertir les jeux d'essais en scripts de tests au moyen des interfaces de programmation proposées par LDTP. Il est possible de contrôler l'exactitude de chaque opération, ce qui fait de LDTP un outil de choix pour l'automatisation des jeux d'essais, dans la mesure où le résultat d'un jeu d'essais est facilement consultable grâce à la liste exhaustive des interfaces de programmation fournie par LDTP.
4. Créer des scénarios de tests en regroupant des scripts de test individuels. Ce regroupement peut vous aider à tester une séquence d'évènements plus facilement. Cette technique permet également de réutiliser les scripts de test afin de générer différents scénarios de tests en rédigeant tout simplement un fichier XML. Nous avons exposé un extrait de fichier XML chargé de créer un scénario de test dans le Listing 2.
5. Lancer le script / scénario de test au moyen du cadre LDTP.

Personnaliser LDTP

Si vous n'êtes pas satisfait des capacités de LDTP, ou si vous souhaitez y implémenter une de vos bonnes idées, surtout n'hésitez pas : grâce à l'excellente documentation très précise du projet, il est assez facile de personnaliser l'outil. Par ailleurs, la communauté LDTP recherche des volontaires actifs susceptibles d'aider au développement du cadre de la nouvelle génération LDTP. Nous apprécions, par exemple, les mordus d'informatique et leur donnons une chance de proposer des supports pour différents langages de programmation dans lesquels seront rédigés les scripts de tests, en développant une interface utilisateur graphique attractive destinée au contrôle de l'exécution des jeux d'essais, et bien plus encore. Toute amélioration des fonctionnalités existantes ainsi que le développement de nouvelles capacités sont les bienvenues. Vous pouvez toujours commencer par rejoindre la liste de diffusion du projet LDTP, à l'adresse suivante : *http://ldtp.freedesktop.org/wiki/Mailing_20list*. Vous pouvez également rejoindre l'équipe sur le canal IRC *#ldtp* sur *irc.freenode.net*. Pour de plus amples informations sur le projet, veuillez consulter le site *http://ldtp.freedesktop.org*.

Résumé

LDTP est un outil puissant capable d'accélérer le développement de vos

Listing 2. *Fichier XML pour la création d'un jeu d'essais*

```
<?xml version="1.0"?>
<ldtp>
  <logfileoverwrite>1</logfileoverwrite>
  <logfile>/EXAMPLES/gedit/gedit-log.xml</logfile>
  <!-appmapfile>/EXAMPLES/gedit/gedit.map</appmapfile-->
  <group>
    <script>
      <name>create-new-file.py</name>
    </script>
  </group>
</ldtp>
```

applications. Grâce aux tests rapides et simples de l'interface utilisateur, vous serez exempté d'une grande partie de tâches répétitives qui vous étaient souvent dévolues, comme le contrôle des liens inexacts d'événements et d'objets. Nous avons présenté, dans le présent article, ses principales fonctionnalités : c'est donc à votre tour d'explorer ce projet plus en détail, et, si vous le souhaitez, d'améliorer également ses fonctionnalités existantes ou même d'y introduire de nouveaux concepts. ■

Sur Internet :

1. *http://ldtp.freedesktop.org/*

   – page d'accueil
2. *http://gnomebangalore.org/ldtp/index.php/Appmap_generation* – comment générer des cartes d'application
3. *http://safsdev.sourceforge.net/* – SAFS – Software Automation Framework Support (Support de Cadre Automatisé pour Logiciel)