AVERTISSEMENT

* Si ce document est à un indice supérieur à ceux précédemment diffusés, il les annule et les remplace.
* En conséquence, son destinataire doit, DES RECEPTION :

1 - DETRUIRE les versions précédentes en sa possession, à des indices inférieurs.
2 - REMPLACER les documents détruits par le présent document.
3 - APPLIQUER cette règle (destruction/remplacement) à l'ensemble des documents copiés sous sa responsabilité.
4 - S'ASSURER, en cas d'obligation de conservation, que les versions précédentes ne peuvent plus être utilisées.

ETAT DES VERSIONS SUCCESSIVES

INDICE	DATE	OBSERVATIONS	REDAC.	VERIF.	APPR
1.0	16/01/2008	Création du document	RGN / JJN		
1.1	14/02/2008	Mise à jour Chapitre 4	R G N		
1.2	02/06/2008	Corrections pour la re-création de binaires	LV		
1.3	25/08/2008	Mise à jour environnements de construction	R G N		

DOCUMENT ÉTABLI SOUS LA RESPONSABILITE DES SIGNATAIRES

REDACTION	VERIFICATION	APPROBATION
Rafik Goulamhoussen Jérome Jeannin **Date :** 14/02/2008	**Date :**	**Date :**

Sommaire

Sommaire.....1
1. Généralités.....4

2. Guide du DÉVELOPPEUR....5
2.1 Environnement de développement....5
2.2 Organisation de l'arbre des sources....5
2.2.1 Répertoire de données (/data)....5
2.2.2 Sources du programme (/ansmo)....7
2.3 Fonctionnement global de l'interface utilisateur....8
2.3.1 Organisation des écrans....8
2.3.2 Propagation des évènements....10
2.3.3 Lecture/Affichage des données....10
2.3.4 Paramètres....11
2.4 La partie pédagogique....11
2.5 Détection du système....12
2.6 La partie diagnostic....13
2.6.1 Philosophie des tests....14
2.6.2 Tests Linux....14
2.6.3 Tests Windows....14
2.6.4 Tests MacOS....15
2.6.5 Ajout de nouveaux tests....16
3. Guide du rédacteur....17
3.1 Organisation générale des modules....17
3.2 Création/modification d'un module pédagogique....17
3.2.1 Structure du plan....18
3.2.2 Syntaxe pour l'écriture des pages....20
3.3 Ajout/modification de questions pour les tests....21
3.4 Enrichissement du glossaire....24
4. Guide du mainteneur....26
4.1 Installation des environnements de construction....26
4.1.1 Ubuntu 7.10 Installation pas à pas....26
4.1.1.1 Pré requis....26
4.1.1.2 Installation et configuration du chroot....26
4.1.1.3 Compilation de QT4....27
4.1.1.4 Compilation de Python 2.5....28
4.1.1.5 Compilation de SIP 4....28
4.1.1.6 Compilation de PyQT 4....28
4.1.1.7 Installation de PyInstaller....28
4.1.2 Ubuntu 7.10 Installation scriptée....28

4.1.2.1 Lancement du script 1-create-chroot.sh....29
4.1.2.2 Edition du fichier /etc/schroot/schroot.conf....29
4.1.2.3 Lancement du script 2-configure-chroot.sh....29
4.1.3 Windows XP SP2....29
4.1.3.1 Pré requis....29
4.1.3.2 Installation de Python 2.5....30
4.1.3.3 Installation des extensions win32 pour Python 2.5....30
4.1.3.4 Installation de QT 4....30
4.1.3.5 Compilation de SIP 4 et de PyQT 4....30
4.1.3.6 Installation de Py2exe....31
4.1.4 Mac OS X 10.4.10....31
4.1.4.1 Pré requis....31
4.1.4.2 Installation de Python 2.5....32
4.1.4.3 Installation de QT 4.3.5....32
4.1.4.4 Installation de PyQT 4.3 et Sip 4.7....32
4.1.4.5 Installation de Py2app....32
4.2 Création de la distribution binaire à partir des sources....32
4.2.1 Ubuntu 7.10....33
4.2.2 Windows XP SP2....33
4.2.3 Mac OS X 10.4.10....33

1.GÉNÉRALITÉS

ANSMO (acronyme d'Analyser le Niveau de Sécurité de mon Micro-Ordinateur) est une application à vocation pédagogique, destinée à sensibiliser les utilisateurs aux problématiques liés à la sécurité informatique. Deux parties distinctes caractérisent l'application :

- l a **partie diagnostic**, qui permet de réaliser, en fonction du système de l'utilisateur, un diagnostic de sécurité ;

- la **partie pédagogique**, qui présente, sous formes de modules, les différentes thématiques de sécurité. Chaque thématique se décline sous différentes version afin de s'adapter aux différents lecteurs (selon le niveau d'expertise et le système d'exploitation de l'utilisateur). Chaque module se termine par une évaluation, sous forme de questionnaire à choix multiples, qui permet de valider l'acquisition des connaissances.

Cette documentation est architecturée en 3 parties :

- **le guide du développeur** : détaille l'architecture du programme (sans aller dans le détail, le code est fortement commenté) ;
- **le guide du rédacteur** : détaille les principes pour intégrer ou modifier les modules pédagogiques. ;
- **le guide du mainteneur** : détaille les procédures pour installer les environnements de compilation par plate-forme et pour générer les binaires.

L'application est **multiplateforme** (Linux, Windows et Mac), écrite en **Python** et tire partie de la librairie **Qt4**. Selon les systèmes, l'organisation sur le disque peut-être différente, mais l'on retrouvera toujours d'un coté le binaire ou les scripts ansmo (`ansmo.exe`, `ansmo.py`, ...) et de l'autre le répertoire contenant les données de l'application (généralement `/data`).

2.GUIDE DU DÉVELOPPEUR

Ce chapitre détaille quelques points importants afin de faciliter la compréhension du fonctionnement du logiciel pour un nouvel arrivant.

En complément, le répertoire doc de l'archive source contient un descriptif général des sources au format HTML, pouvant être utilisé comme référence lors du développement.

2.1 Environnement de développement

L'application ANSMO est entièrement développée avec le langage Python, et repose sur l'utilisation de la librairie Qt pour toute la partie interface utilisateur. Qt est une librairie écrite en C++. PyQt permet d'utiliser cette librairie depuis Python.

L'environnement nécessaire au développement est :

- Python, au minimum dans sa version 2.5
- QT 4.3
- PyQT 4.3

Et un éditeur de texte afin de modifier les différents fichiers.

Le développement peut aussi bien être réalisé sous les systèmes Windows, Mac que Linux.

Pour des exemples d'installation d'environnement, il est possible de se référer au chapitre 4.1 Installation des environnements de construction.

2.2 Organisation de l'arbre des sources

Les sources sont divisées en deux parties :

- le répertoire `data` : contient toutes les données modifiables sans reconstruction des binaires ;
- le répertoire `ansmo` et le fichier `ansmo.py` : sources du programme.

2.2.1 Répertoire de données (/data)

Ce dossier est structuré ainsi :

```
\data
|   glossaire.html                          généré automatiquement
|   glossaire.xml                           contient les définitions
|   modules.xml                             contient la liste des plans utilisés
|   parameters.xml                          contient divers paramètres du logiciel
|   supported.xml                           contient la liste des systèmes supportés
|   welcome.html                            message affiché dans l'écran d'accueil
|
+---about                                   contenu de la rubrique A propos
|       about.html
```

```
|       contrat.html
|
+---eval                                          messages affichés après les évaluations
|       000.html
|       025.html
|       050.html
|       075.html
|       100.html
|
+---help                                          contenu de l'aide
|       diag.html
|       glossaire.html
|       menuprincipal.html
|       navigation.html
|       onoffline.html
|       pedago.html
|       utilisation.html
|
+---imgs                                          contient les images utilisées hors
module
|       agir.png
|       ansmo.png
|       word.png
|
+---licence                                       contient le contrat utilisateur
|       contrat_utilisateur.html
|       message.html
|
+---modules                                       contient les modules pédagogiques
|   |   modules.html
|   |
|   \---gestiondescomptes
|           1_0Xi.png
|           2_23a.png
|           2_23b.png
|           2_24.png
|           2_24_modification_de_compteWIN.html
|           q1_1.xml
|           q2_5V.xml
|
+---plan                                          contient les plans des modules
|       gestion_MDP.xml
|       _ansmo-about.xml
|       _ansmo-help.xml
|       _ansmo-licence.xml
|
\---syscheck                                      contient les messages des diagnostics
        antivirus_enabled.html
        antivirus_no.html
        antivirus_unabled.html
        antivirus_unknow.html
        diagnostic.xml
        _message_accept.html
        _message_resutats.html
```

Cependant, quelques fichiers jouent un rôle primordial pour le bon fonctionnement de l'application et nécessitent d'être présents. Voici ces fichiers :

- `modules.xml` : définit la liste des modules à présenter dans l'interface
- `supported.xml` : précise les systèmes supportés pour le diagnostique
- `parameters.xml` : contient un large ensemble de paramètres pour l'application (couleurs, polices de caractères, ...)
- `glossaire.xml` et glossaire.html : les données relatives au glossaire et le glossaire complet sous forme HTML.
- `plan/licence`, `plan/about`, `plan/help` : les plans relatifs au contenu affiché respectivement lorsque l'utilisateur lance l'application, dans l'écran « A propos » et dans l'écran « Aide »
- le répertoire `imgs` et son contenu : contient les ressources graphiques qui seront utilisé pour composer l'interface.

2.2.2 Sources du programme (/ansmo)

Les sources sont architecturées ainsi :

```
ansmo.py
\ansmo
|   __init__.py
|
+---misc
|       debug.py
|       orderedDict.py
|       __init__.py
|
+---read
|       Glossaire.py
|       Module.py
|       Os.py
|       Parameters.py
|       Plan.py
|       Question.py
|       ReadHtml.py
|       __init__.py
|
+---screen
|       AnsmoScreen.py
|       BasicView.py
|       Exit.py
|       Glossary.py
|       GlossaryLaunch.py
|       License.py
|       MainMenu.py
|       NavMenu.py
|       Pedago.py
|       Questionnary.py
|       Syscheck.py
|       Welcome.py
|       Zoom.py
|       __init__.py
|
+---system
|       LinuxTest.py
|       MacTest.py
|       PasswordCheck.py
|       SystemDetection.py
|       TestCollection.py
|       WindowsTest.py
|       __init__.py
|
\---view
        MenuLeft.py
        ViewAnimation.py
        ViewItem.py
        Window.py
        __init__.py
```

Le script `ansmo.py` est le point d'entrée pour l'exécution de l'application. Ce script crée une instance de la classe ANSMO, qui contient tout ce qui sera nécessaire à l'exécution et l'utilisation du logiciel.

Le sous-répertoire ansmo contient quand à lui les différents modules suivants :

- `screen`

Pour chaque écran d'ansmo, une classe définit l'organisation de son contenu et le comportement des différents éléments qui la composent. Ce module contient toutes ces classes, comme par exemple `Licence.py` pour l'écran qui invite l'utilisateur à lire et accepter la licence utilisateur, ou bien encore `Questionnary.py` pour le module servant aux évaluations.

- `system`

C'est ici que sont placés les modules servant au diagnostic du système. Étant donné les différences d'architecture assez importantes entre les différentes plateformes sur lesquelles s'exécute l'application, on trouvera ici des versions adaptés pour chaque OS des tests à réaliser.

- `read`

Tous les modules utilisés pour la lecture des fichiers de données (HTML, XML, ...) nécessaire au fonctionnement de l'application se trouvent placé dans ce module.

- `view`

Un certain nombre de classes de Qt4 a été étendu par héritage afin d'adapter leur fonctionnement aux spécificités de l'application. Ce module contient les nouvelles classes relatives aux éléments d'interface utilisées par les modules contenus dans `screen`. On trouvera aussi la définition de la fenêtre affiché à l'utilisateur.

- `misc`

Les derniers modules un peu en marge du reste sont placés ici. Pour l'instant, on trouve dans ce module un dictionnaire ordonné (utilisé par un autre module d'ansmo) et un module utilisé lors du débogage qui permet de tracer l'exécution de l'application via des écritures dans un fichier de log.

2.3 Fonctionnement global de l'interface utilisateur

Pour l'interface utilisateur, nous tirons partie des fonctionnalités offertes par la librairie Qt4, et tout particulièrement du framework « Graphics View ». Celui-ci met à notre disposition un ensemble complet d'outils pour gérer des composants 2D personnalisés, et nous pouvons alors librement implémenter nos propres composants ayant leur propre design.

Grâce à ce framework, nous avons eu l'occasion de concevoir une interface utilisateur au look et à l'ergonomie plus chaleureuse, et moins proche des interfaces utilisant des widgets standards.

J'invite chaleureusement les développeurs qui souhaitent modifier l'interface à se rendre sur le site de Trolltech (le développeur de Qt) afin de consulter la documentation de cette librairie.

2.3.1 Organisation des écrans

Qt définit différentes classes nécessaires au fonctionnement du framework « Graphics view ». On retrouve notamment la notion de scene et de vue (view). La scène va accueillir tous les composants graphiques que l'on souhaite voir apparaître à l'écran, et la vue se chargera de faire le rendu des éléments contenus dans la scène.

C'est la fenêtre de notre application qui fera office de vue, et la scène sera instanciée par la classe qui nous sert de point d'entrée : ansmo.

Lorsque l'application démarre, la classe ansmo crée les instances des différents écrans qui seront susceptible d'être affichés, et les rattaches à la scène. Il nous est maintenant possible d'afficher ou de masquer ces écrans en appelant les méthodes show() et hide(). La classe ansmo se chargera d'afficher et de masquer les écrans existant en fonction des événements déclenchés par les actions de l'utilisateur.

Notons que le même mécanisme est utilisé au sein même de nos différents écrans, qui affichent ou masquent les éléments qui les composent lorsque l'utilisateur les utilise.

On retrouve les écrans suivant :

- « License »

Affiche le contrat utilisateur, et dispose des boutons pour continuer ou quitter l'application

- « Welcome »

L'écran affiché par défaut une fois le contrat de licence accepté. Affiche le contenu d'un fichier de ressources présent dans `data/`.

- « Pedago »

Module qui permet d'afficher les ressources pédagogiques à l'utilisateur. Il se compose d'un menu de navigation, d'un élément de rendu html et du module d'évaluation.

- « Syscheck »

Module qui affiche les résultats de l'analyse système. Ce module est lui-même composé de différents sous-écrans pour les différentes phases du test.

- « Help »

Instance de BasicView (menu + contenu standard), affiche l'aide

- « About »

Instance de BasicView (menu + contenu standard), affiche diverses informations légales

- « GlossaryLaunch »

Offre à l'utilisateur le choix de lancer ou non le glossaire externe.

- « Glossary »

Affiche à l'utilisateur la définition d'un mot sur lequel il vient de cliquer

- « Zoom »

Permet d'afficher à l'utilisateur une version plus grande d'une image sur laquelle il vient de cliquer.

- « Exit »

Affiche un menu invitant l'utilisateur à confirmer la sortie du logiciel.

2.3.2 Propagation des évènements

Afin de répondre aux actions de l'utilisateur sur l'interface, un système d'évènements est utilisé afin de propager l'action de l'utilisateur aux éléments susceptible de réagir à celui-ci.

Dans la librairie Qt, ce concept est représenté par les signaux (Signal). Il est possible pour tous les objets de Qt d'écouter un objet donné et de réagir lorsque ce dernier émet un signal en particulier (voir documentation Qt pour plus de détails.)

Ce mécanisme est utilisé de façon explicite dans les différents écrans, afin de réagir par exemple au clic de l'utilisateur sur un élément du menu et d'afficher le texte correspondant. Il est aussi utilisé d'une façon un peu plus subtile afin de propager les évènements entre les différents écrans. Puisque tous nos éléments graphiques sont rattachés à la scène, il est possible d'utiliser cet objet comme vecteur pour propager l'événement entre des éléments de l'interface si ces derniers n'ont pas de référence directe les uns avec les autres.
Par exemple, lorsque l'utilisateur clique sur le bouton « Quitter » présent dans l'interface, le signal exit est émit à travers la scène. L'instance d'ansmo, qui écoute la scène en attente de ce signal, affiche alors l'écran de confirmation une fois la réception du message.

2.3.3 Lecture/Affichage des données

Pour afficher le contenu des fichiers html, nous utilisons la librairie Qt et particulièrement l'objet `QgraphicsTextItem`. Celui-ci nous permet d'afficher du texte enrichi, et tout particulièrement de l'html.

L'ensemble des éléments standards html supportés pour le formatage des documents dépend donc des éléments supportés par la librairie Qt. A cela, se rajoute un certain nombre de comportements particuliers, spécifiés par des balises ou des attributs particulier à ansmo (voir la partie « Syntaxe pour l'écriture des pages » pour plus de détails).

La modification la plus utile pour le développement de l'application est la personnalisation des liens hypertexte qui permettent d'interagir avec le reste de l'application. L'attribut `href` n'est plus utilisé pour renseigner le chemin vers une ressource html, mais associé à un événement qui sera déclenché suite au clic de l'utilisateur (ce qui nous offre la possibilité de créer des boutons simplement via la création d'un objet html.)

D'autres modifications peuvent plutôt être considérées comme des raccourcis facilitant l'écriture des pages. Les blocs d'emphases par exemple, sont définis par une simple balise dans le fichier html, et transformé en tableau avec des attributs prédéfinis (couleurs, images, ...) pour l'affichage.

Toutes ces modifications particulières sont effectuées lors de la lecture du fichier html. L'arbre html est parcouru, et tous les éléments qui doivent l'être sont transformés les uns après les autres.

2.3.4 Paramètres

Dans le répertoire des données se trouve le fichier `parameters.xml`. Ce fichier contient un grand nombre de paramètres qui permettent d'influer sur divers aspects graphiques de l'application (couleurs, police, ...) et d'activer ou non certaines fonctions (comme les traces utilisées à des fins de débogage).

Ce fichier est lu au lancement de l'application, et toutes les classes ont accès à la même instance de l'objet `Parameters`, qui peut aussi être utilisé pour stocker des valeurs et échanger des données de façon indirecte entre plusieurs classes de l'application.

Pour simplifier, on peut considérer que cette classe fait office de conteneur pour des variables globales accessibles depuis toute notre application.

2.4 La partie pédagogique

Globalement, la partie pédagogique est composée de quatre éléments :

- une instance de MenuLeft, permettant d'afficher le menu qui liste les modules pédagogiques et leurs plans,
- une instance de ViewHtml qui permet de faire le rendu des fichiers html qui contiennent les fiches pédagogiques,
- une instance de Questionnary qui permet de gérer les évaluations,
- une instance du menu de navigation qui permet de passer de pages en pages.

Le fonctionnement de cette partie est simple.

A l'initialisation, le fichier contenant la liste des modules pédagogiques (modules.xml) est lu, et est utilisé pour remplir le menu avec la liste des modules.

Un fichier html contenant un texte explicatif est lu à son tour et affiché dans le *ViewHtml*. Ensuite, ce sont les évènements provoqués par l'utilisateur qui rafraichissent le contenu de ces éléments pour le substituer avec celui des ressources correspondantes à ce qu'il souhaite consulter.

Lorsque l'on souhaite procéder à l'évaluation, le procédé est le même. Mis à part que l'on lit alors les fichiers contenants les questions et que l'on masque l'objet *ViewHtml* pour afficher à sa place l'objet *Questionnary*.

Voir chapitre 3. Guide du rédacteur pour plus d'informations sur la syntaxe des différents fichiers utilisés.

2.5 Détection du système

La détection du système est très importante pour l'application, et tout particulièrement pour la partie diagnostique.

Cette détection repose sur celle de Python, et permet d'obtenir trois valeurs qui caractérisent le système : la plateforme (Windows, Linux ou Mac), l'os/la distribution (Windows Xp, Windows Vista, Ubuntu, Fedora, ...) et enfin la version correspondante.

Ces paramètres sont ensuite conservés, et utilisés à plusieurs fins.

Tout d'abord, ils servent à présélectionner au mieux le système de l'utilisateur pour la partie pédagogique. Ainsi, lorsque l'utilisateur consultera les différents modules, le système le plus adapté sera automatiquement présélectionné à son attention.

Cette détection va également déterminer s'il est possible ou non d'exécuter les tests de la partie diagnostique. Une liste des systèmes sur lesquels les tests de diagnostique ont été validés est maintenue et comparée au système détecté sur la machine de l'utilisateur (fichier supported.xml).

S'il y a correspondance avec une des entrées de notre fichier, alors les tests pourront être exécutés sur la machine. Dans le cas contraire, et afin de ne pas risquer de voir les tests échouer techniquement, un message précisant que la version de l'os actuellement utilisée n'est pas supportée par ansmo est affiché.

Voici un aperçu du fichier supported.xml:

```
<os_list>

<system type="Windows">
   <os name="Windows XP" />
   <os name="Windows 2000"/>
   <os name="Windows Vista"/>
</system>

<system type="MacOS">
   <os name="Mac OS X" />
</system>

<system type="Linux">
   <os name="Ubuntu" version="7.10" />
   <os name="MandrivaLinux"/>
```

```
    <os name="OpenSuse" version="10.3"/>
    <os name="Debian" version="4.0r2"/>
    <os name="Fedora" version="8"/>

</system>

</os_list>
```

Comme pour une grande partie des fichiers de l'application, le format xml est mis à contribution. Sous la racine *os_list*, sont définis les différents systèmes d'exploitation supportés.

Chaque système est représenté par un élément *system*, avec un attribut *type* à préciser obligatoirement et qui fait référence au nom du système (actuellement, les valeurs acceptés pour les systèmes supportés sont *Windows*, *MacOS* et *Linux*).

Ensuite, il est précisé chaque version du système par une entrée de type *os*. Deux attributs peuvent-être précisés : *name* et *version*. '*name*' est comparé au nom du système détecté chez l'utilisateur, et doit nécessairement correspondre pour considérer le système comme supporté. '*version*' permet de vérifier plus précisément l'os : s'il est précisé, cet attribut est comparé au préfixe du numéro de version de l'utilisateur (ainsi, si l'utilisateur dispose d'une version '2.1' d'un os et que version a pour valeur '2', le système sera considéré comme valide). S'il y a correspondance sur au moins l'une des entrées dans cette liste de systèmes, alors le système est considéré comme supporté et les tests peuvent être lancés.

2.6 La partie diagnostic

Etant donnée les différences qui existent entre toutes les systèmes supportés par le logiciel ANSMO, il est nécessaire de définir des tests particuliers pour chacune de ces plateformes.

Une fois la détection du système effectué, les modules contenant les tests supportés par le système sont chargés,

Chaque test, une fois chargé, s'enregistre auprès d'un objet *TestCollection*, qui a vocation de regrouper tous les tests disponibles, en précisant le nom associé au test,

Une liste des tests à effectuer pour chaque plateforme est maintenue dans la classe utilisée pour déclencher le lancement des tests.

Lorsque l'on exécute la batterie de tests, cette liste est confronté aux objets de notre *TestCollection* et les tests correspondants sont exécutés les uns après les autres
Chaque test contenu dans la collection contient à présent le résultat suite à son exécution, et la collection peut-être utilisée pour procéder à l'affichage du bilan à l'utilisateur.

Tous les tests doivent hériter de la classe de base *SystemTest*, afin de posséder obligatoirement de la méthode *run()* utilisée pour procéder à l'exécution du test et de la variable resultText qui est utilisée pour contenir le résultat. Pour le reste, l'implémentation du test est à l'appréciation du développeur.

2.6.1 Philosophie des tests

Les tests implémentés dans l'application ne sont pas destinés à remplacer les applications de diagnostic disponibles sur les différentes plateformes, et respectent les principes suivant afin de correspondre à la philosophie du SGDN :

- Les tests ne doivent en aucun cas être intrusifs sur le système : ils ne doivent accéder qu'en lecture aux différents fichiers présents, et en aucune manière écrire sur le système de l'utilisateur ou effectuer des manipulations pouvant compromettre la sécurité du système même à des fins de diagnostic.

- les tests ne peuvent accéder au système uniquement avec le statut d'un utilisateur standard, et en aucun cas demander des privilèges particuliers (administrateur, root, ...).

2.6.2 Tests Linux

Pour chacun des tests suivants, nous allons brièvement en indiquer le principe afin d'éclaircir leur fonctionnement et en montrer les limites.

Pare-feu : Le but est de déterminer si un pare-feu est installé et si celui-ci est activé. Sur les systèmes Linux, iptables est l'applicatif qui fait office de pare-feu sur la totalité des systèmes. Son fonctionnement est très lié au système, et il aisé de vérifier sa présence. Ainsi, si le binaire *iptables* est présent sur le système, nous pouvons considérer que ce dernier dispose d'un pare-feu.

Il n'est cependant pas possible de vérifier si ce dernier est actif ou non. *iptables* peut-être présent sans qu'aucune règle ne soit définie. Les commandes permettant de déterminer l'activation du pare-feu nécessitant des privilèges administrateur, il n'a pas été possible d'implémenter cette vérification.

Mise-à-jour : Le but est de déterminer la date de dernière mise à jour du système. Cette information est obtenue:

- en vérifiant la date de dernière modification des fichiers de dépôts (pour les systèmes *Ubuntu* et *Debian*),
- ou bien encore via l'utilisation de la commande rpm (systèmes *Mandriva*, *Suse* et *Fedora*).

Droits utilisateur : Le but est de déterminer si l'utilisateur utilise ou non le compte root. Via la commande *groups* (en lui passant l'utilisateur courant en paramètre), cette information peut être déterminée, et il est possible d'en déduire le résultat du test.

2.6.3 Tests Windows

Sous Windows, les tests se révèlent être un peu plus complexe que sous Linux. En effet, l'architecture du système est différente entre les versions de l'os, et les outils ou commandes permettant d'obtenir les informations sur une version donnée ne fonctionneront pas (ou ne sont pas disponibles) sur une autre. Il est donc nécessaire de mettre en place plusieurs stratégies pour tester chaque aspect, afin de s'assurer que l'une d'elle au moins nous permettra d'obtenir un résultat fiable.

Pare-feu : Pour vérifier la présence du pare-feu, la première vérification que l'on effectue fait appel à WMI. Si le système a le centre de sécuité (disponible à partir de windows xp sp2 et +), alors ce dernier va retourner toutes les informations sur un éventuel pare-feu enregistré dans le système (nom, version, état d'activité, ...).

Si la vérification par WMI n'est pas effective, alors la seconde stratégie consiste à effectuer une requête via netsh (disponible à partir de windows xp), et de demander l'état du pare-feu intégré au système (*netsh firewall show state*). En fonction du retour de cette commande, il nous est possible de déduire le résultat du test.

Finalement, si netsh ne donne pas plus de résultat (le firewall n'a toujours pas pu être déterminé), alors c'est via la base des registres qu'est effectuée la dernière tentative. Il est possible de connaître via cette base l'ensemble des programmes installés sur un système. En comparant les entrées contenues dans cette liste de programmes avec un ensemble de chaînes identifiants les principaux pare-feu du marché, nous pouvons estimer si oui ou non un de ces produit est présent. Cette dernière méthode ne permet pas de connaître l'état du firewall (est-il activé ?) mais permet de donner des conseils à l'utilisateur et de le guider dans son usage.

Antivirus : Tout comme pour le pare-feu, deux stratégies sont mises en place : l'une par WMI (Centre de sécuirité), l'autre par la base des registres en comparant aves une liste d'anti-virus connus.

Mise-à-jour : Le but est d'obtenir la date de dernière mise à jour du système et de savoir si les mises à jour automatiques sont activées.

Pour obtenir la date de mise à jour du système, deux stratégies :

- via la base de registre en récupérant la plus récente date d'installation d'un hotfix présent dans les clés Uninstall.
- Via la base de registre en récupérant la date de dernière installation réussie des mises à jour automatiques.

L'état d'activation des mises à jour automatiques est obtenu via une clé de la base de registre.

Droits utilisateur : Pour récupérer le type de compte l'utilisateur, la commande *net localgroup* nous permet de savoir si l'utilisateur appartient au groupe administrateur (ou administrator) ou non. Sous Windows, ce sont les membres de ce groupe qui bénéficient des droits les plus avancés.

2.6.4 Tests MacOS

Pare-feu : Par défaut, le système MacOS embarque un pare-feu. Il est possible d'indiquer à l'utilisateur l'état du pare-feu via la commande *system_profiler*. Cette dernière retourne le statut du pare-feu.

Mise à jour : Le fichier *Software Update.log* contenu sur le système contient l'historique de toutes les mises à jour effectuées. Pour connaître la date de dernière

mise à jour du système, il suffit de consulter la dernière entrée de ce fichier et de consulter la date indiquée.

Droits utilisateur : Comme pour l'équivalent Linux de ce test, c'est via la commande *groups* que l'on obtient les informations sur les privilèges de l'utilisateur. Si celui-ci appartient au groupe *admin*, alors l'utilisateur utilise un compte de super-utilisateur.

2.6.5 Ajout de nouveaux tests

Pour ajouter de nouveaux tests, il faut procéder en deux étapes.

Tout d'abord, créer une nouvelle classe de test héritée de *SystemTest* et la placer dans le fichier correspondant au système concerné (éventuellement, il est possible de placer les nouveaux tests dans d'autres modules, et les charger lors de la détection du système.) Enregistrer le nouveau test dans la collection (*TestCollection*) afin de le rendre disponible.

Ensuite, pour rendre le test effectif, rajouter l'identifiant du test dans la liste des tests du système concerné (WindowTest, LinuxTest, ou MacTest) dans le code source exécutant les tests.

Le test est désormais disponible dans l'application et peut-être lancé par l'utilisateur : il apparaît dans le menu sous l'appellation spécifiée dans le test, et sera incluse dans la batterie de test qui sera exécutée.

3.GUIDE DU RÉDACTEUR

Lorsque l'on souhaite créer du contenu pour le logiciel ANSMO, il est possible d'agir à différents niveaux : ajouter un nouveau module, modifier un des modules existants, éditer les questions des tests d'évaluation ou enfin éditer le glossaire.

3.1Organisation générale des modules

Afin de pouvoir afficher les différents modules dans l'application, nous disposons d'un fichier au format XML qui liste ces modules. Ce fichier, intitulé modules.xml, est présent dans le répertoire data.

Puisqu'un exemple vaut mieux qu'un long discours, prenons comme exemple un extrait de ce fichier contenant trois modules :

A la racine de notre arbre XML, on trouve un élément modules, qui contient les éléments module que nous allons détailler dès à présent.

Chaque élément module, définit, comme vous vous en doutez, un module présenté dans l'application. Trois attributs doivent être spécifiés pour chaque entrée :

- id : une chaine de caractères unique, qui permet de faire référence au module depuis l'application (par exemple, liens vers un module depuis la partie diagnostique)

- title : le titre du module, qui sera affiché dans la liste des modules et dans le bandeau supérieur de l'interface lors de la consultation

- src : chemin vers le plan du module (nous détaillerons plus loin le contenu du plan)

/!\ Afin d'être accessible au plus grand nombre, il n'existe pas de systèmes « d'ascenseur » permettant de faire défiler la fenêtre dans ANSMO. En conséquence, si la liste des modules est trop imposante, celle-ci sortira hors de l'écran et les derniers modules ne pourront être utilisés. Garder ceci à l'esprit !

3.2Création/modification d'un module pédagogique

Pour chaque module, un ensemble de fichier est nécessaire afin de pouvoir l'afficher dans l'application. Il nous faut :

- le plan, référencé dans le fichier modules.xml
- les fiches html et les images à afficher à l'utilisateur

Par convention (et afin d'éviter de se retrouver avec quelque chose d'ingérable par la suite), un dossier est crée pour chaque module afin de contenir l'intégralité des fichiers relatif a ce module (dans le dossier data/modules).

3.2.1 Structure du plan

Le plan d'un module est un fichier XML qui liste l'ensemble des fichiers html relatifs au module, et précise comment sont organisées ces fiches. En fonction de son niveau ou de son système d'exploitation, des parties du plan seront actives et d'autres non.

Un plan se découpe en chapitres (chapter), qui peut éventuellement se découper en multiples sous-parties (part). Ces éléments peuvent contenir des pages (page) et des exercices d'évaluation (test).

Prenons un exemple afin d'illustrer la structure d'un plan :

A la racine de notre arbre XML, on trouve un élément chapters. En dessous, on trouve les éléments chapter qui définissent chacun l'un de nos chapitre. Ici, notre document contient quatre chapitres.

Cet élément nécessite obligatoirement que l'on précise son attribut title, qui permet de remplir le menu présentant les chapitres. De façon optionnelle, les attributs os et lvl peuvent être précisé. S'ils sont définis (l'un, l'autre ou les deux), le chapitre n'apparaitra à l'utilisateur que si l'utilisateur se trouve dans le cas de figure correspondant.

Actuellement, voici les valeurs acceptés pour ces attributs :

os = 'linux' (Linux), 'osx' (MacOS X), 'winXP' (Windows XP), 'win2k' (Windows 2000) ou 'vista' (Windows Vista)

os :

- 'linux' (Système Gnu/Linux, quelque soit la version)
- 'osx' (Système Mac OS X)
- 'winXP' (Système Windows XP)
- 'win2k' (Système Windows 2000)
- 'vista' (Système Vista)

lvl :

- '1' (niveau débutant)
- '2' (niveau avancé)

L'élément part définit les sous-parties, et dispose aussi des attributs facultatifs os et lvl sur le modèle de l'élément chapter. Dans notre exemple, seul le premier chapitre dispose d'une sous-partie, mais bien évidement il est très conseillé de structurer le module pédagogique à l'aide de nombreux chapitres et sous-parties.

Page définit une page de notre module. Le chemin vers le fichier à afficher est précisé via l'attribut obligatoire src (notons que le chemin est relatif par rapport à la racine du répertoire data).Il est aussi possible de renseigner le champ id avec une chaîne de caractère unique de son choix, qui permettra de faire référence à cette page depuis un autre endroit du plan. Comme on peut le voir dans notre exemple, il est possible de placer les pages sous les chapitres et les sous-parties.

Il est possible d'effectuer des sauts dans le plan via un élément de type jump. Cet élément dispose d'un attribut obligatoire goto, qui fait référence à l'id d'une page dans le plan. Cet élément permet de répondre à des problématiques bien particulières au niveau du plan, mais nous déconseillons son usage intensif car il peut très facilement amener des effets de bord.

Maintenant, examinons la syntaxe à utiliser pour ajouter un test d'évaluation. Prenons en exemple le fichier suivant :

On peut voir que le second chapitre contient un élément que nous n'avons pas détaillé jusque là : l'élément test.

C'est cet élément qui nous sert à définir une évaluation. Les questions qui composent notre test sont contenues à l'intérieur via des éléments de type exo.

L'attribut src est obligatoire, et précise le chemin vers le fichier contenant la question (la syntaxe des fichiers de question sera abordée plus loin dans ce document).

3.2.2 Syntaxe pour l'écriture des pages

Les pages présentées dans la partie pédagogique sont de simples fichiers HTML, et font usage des balises les plus courantes. Le langage HTML est utilisé ici à des fin de mise en forme, et il n'est pas possible de faire usage de JavaScript ou de style CSS avancés (l'interpréteur html utilisé n'exposant pas ces possibilités).

L'ensemble des balises supportées peut être consulté à cette adresse :
http://doc.trolltech.com/4.3/richtext-html-subset.html

Il est à noter que dans l'implémentation d'ANSMO, un certain nombre de balises voient leur comportement par défaut modifié afin de faciliter le travail de mise en page des rédacteurs. Par exemple, le style des paragraphes est aligné par défaut, ou bien certaines classes de style sont disponibles pour mettre des éléments en évidence.

Pour palier au manque de feuille de style, il est néanmoins possible de définir l'apparence des composants les plus utilisés via le fichier parameters.xml.

Dans ce fichier, on trouve un certain nombre d'entrées qui permettent de modifier les polices utilisées dans les différents composants, ainsi que leur couleur. Il est possible de définir le styles des blocs titres (H1, H2, ...) ou bien encore les images associés automatiquement aux liens.

Par commodité, des blocs d'emphases ont été prédéfinis. Pour les utiliser, il suffit de préciser la classe d'un bloc de type div de la même façon que si nous disposions d'une feuille de style CSS.

Ainsi :

```
<div class="bloc_emphase_1">
  <p>Le texte à mettre en évident est placé dans le bloc div de type bloc_empahse_</p>
</div>
```

Le style du bloc (couleurs, polices, ...) et l'image qui lui est associé sont stockés dans les fichiers parameters.xml. Pour modifier les styles existants ou en ajouter de nouveaux, il suffit de définir emphase_n_style et emphase_n_img, où n est un entier qui nous permet d'identifier le bloc.

Par exemple :

```
<emphase_1_style>background-color : #8FB7C4;</emphase_1_style>
<emphase_1_img>agir.png</emphase_1_img>

<emphase_2_style>background-color : #8FB7C4;</emphase_2_style>
<emphase_2_img>attention.png</emphase_2_img>
```

Autre modification, le comportement des liens (balise <a>) à été étendue afin de permettre une interaction entre l'application et le contenu des fichiers html. Selon le préfixe de l'attribut href, le lien aura une signification et un comportement différent. Ainsi:

- un lien vers un site externe est défini classiquement par Lien externe

- un lien vers un fichier interne (dans le répertoire data) est défini par Lien interne

- un lien vers une définition du glossaire par mot, où 'id_definition' fait référence à l'id d'un mot existant dans le glossaire de l'application.

- une abréviation ou un acronyme par Abr., où 'Définition' est à substituer par la définition complète de l'abréviation 'Abr.'

- un lien vers un autre module se définit ainsi : lien vers module, où 'ID_MODULE' correspond à l'id d'un module défini dans la liste des modules.

- un lien vers une page du module courant se définit ainsi : lien vers page, où 'ID_PAGE' correspond à l'id d'une page définie dans le plan.

Parmi les autres propriétés qui ont été ajoutées, un nouvel attribut à fait son apparition pour les images (balise img). Il est désormais possible de préciser, via l'attribut zoom, si une image peut-être agrandie lorsqu'un utilisateur clique dessus. Pour permettre cette possibilité, il suffit de placer la valeur 1 dans cet attribut.

Exemple :

```
<img src=”imgs/capture.png” zoom=”1”/>
```

Pour finir, nous attirons l'attention des rédacteurs sur la limitation de la place disponible à l'écran. La vue, à l'intérieur de la fenêtre, à été volontairement conçue sans possibilité de défilement afin d'être accessible au plus grand nombre. De fait, la place disponible pour afficher du contenu à l'écran se trouve être limité, et il est primordial de bien organiser ses documents afin de présenter aux utilisateurs un contenu agréable à lire.

3.3Ajout/modification de questions pour les tests

Tout comme les fiches pédagogiques, des fichiers séparés sont utilisés pour définir les questions qui composent les exercices d'évaluation.

Chaque fichier définit une question.

Les questions se présentent sous la forme d'un questionnaire à choix multiples, et l'utilisateur dispose de deux essais pour tenter de trouver les bonnes réponses (exception faite pour les questions ne disposant que de deux propositions).

Le contenu d'un fichier contenant une question contient les éléments suivants :

- un libellé, qui représente le texte de la question;
- une liste de propositions, parmi laquelle se trouve la ou les réponses exactes;
- des phrases de commentaires éventuelles, qui seront affichées à l'utilisateur suite a ses propositions.

Voici l'exemple d'un QCM très simple :

```
<?xml version="1.0" encoding="UTF-8"?>
<question>
	<label>Quel est la couleur du cheval blanc d'Henry IV</label>

	<answers>
		<answer value="0">Noir</answer>
		<answer value="0">Gris</answer>
		<answer value="0">Blanc</answer>
		<answer value="1">Marron</answer>
	</answers>

	<feedbacks>
		<feedback>Oui, la réponse est bien blanc !</label>
	</feedbacks>

</question>
```

Cet exercice affiche à l'utilisateur le texte de la question (« Quel est la couleur du cheval blanc d'Henry IV ? »), et lui propose de choisir sa réponse parmi la liste des choix offerts (propositions contenues dans <*answsers* />.

Si l'utilisateur donne une réponse correcte, c'est le texte contenu dans *feedback* qui sera affiché. Dans le cas contraire, un commentaire prédéfini sera affiché à l'utilisateur (voir plus loin).

Pour chaque question, il est possible de redéfinir ces textes par défaut par d'autres en les ajoutant au fichier. Il est aussi possible de définir des commentaires particuliers en fonction de la réponse donnée par l'utilisateur.

Par exemple :

```
<?xml version="1.0" encoding="UTF-8"?>
<question>
	<label>Combien d'habitant comporte la ville de Montpellier ?</label>

	<answers>
		<answer value="0">4 900</answer>
		<answer value="0"> 182 500</answer>
		<answer value="1">212 600</answer>
		<answer value="0">318 700</answer>
		<answer value="0">427 200</answer>
	</answers>

	<feedbacks>
```

```
            <feedback id="R">C'est une bonne réponse !</label>
            <feedback id="W">Réponse incorrecte, vous devez réviser </label>
      </feedbacks>
</question>
```

Les éléments answer se sont vu rajouter un attribut feedback, dont la valeur fait référence à l'id des éléments feedback : ce sont ces phrases qui seront affichées lorsque l'utilisateur répondra par ces réponses.

- R (right), affiché lorsque la réponse est correcte : « Réponse correcte ! »
- W (wrong), affiché lorsque la réponse est fausse et que tous les essais ont été utilisés : « Réponse fausse ! »
- I (incomplet), affiché lors d'un choix multiple, quand au moins une des réponses attendues a été sélectionné, mais que d'autres réponses sont manquantes ou bien qu'une mauvaise réponse ait été elle aussi choisie : « La réponse est fausse ou incomplète, essayez à nouveau ! »
- T (try again), affiché lorsque la réponse est fausse, et qu'il reste une autre tentative à l'utilisateur : « La réponse est fausse, essayez à nouveau ! »
- X, affiché lorsque l'utilisateur n'a sélectionné aucune réponse : « Veuillez choisir au moins une réponse parmi celles proposées »

La structure du fichier question est la suivante :

Un élément **question** comme élément racine. Cet élément peut avoir comme fils des éléments de type **label**, **answers** et **feedbacks**. L'attribut **type** est obligatoire et peut prendre les valeurs QCM ou TT (choix multiples ou texte à trous.)

L'élément **label** définit le texte de la question.

L'élément **answers** est utilisé comme conteneur, il contient des éléments **answer**.

L'élément **answer** contient une réponse. Son attribut **value** détermine s'il s'agit d'une réponse vraie (valeur 1) ou fausse (valeur 0).

La présence de plusieurs valeurs à « 1 » entraîne que le questionnaire est à réponse multiple, c'est-à-dire des cases à cocher. A défaut, il est à réponse unique : des boutons radio.

L'élément **feedbacks** est utilisé comme conteneur, il contient des éléments de type **feedback**. Il est facultatif, et dans ce cas ce sont des réponses prédéfinies qui seront affiché à l'utilisateur.

L'élément **feedback** détermine les textes à afficher lors des réponses. L'attribut **id** est facultatif, mais dans ce cas l'élément désigne la réponse affichée lors d'une bonne réponse (cas W). L'id peut prendre une des valeurs parmi R, W ,I, T et X pour redéfinir les messages par défaut.

Si l'on souhaite faire contenir plusieurs questions dans notre fichier, il est aussi possible d'imbriquer plusieurs éléments **question** dans un élément de type **questions**.

L'élément **questions** devient alors le nouvel élément racine de notre document.

3.4Enrichissement du glossaire

Le glossaire est disponible sous deux formes :

- à l'intérieur de l'application, via les liens placés sur les mots à définir, qui permettent de visualiser la définition
- via le glossaire externe, ouvert dans le navigateur disponible sur le système sous la forme d'un fichier html.

Ces deux fichiers sont exploités différemment par l'application, et afin d'éviter la maintenance de ces deux sources, le fichier html est généré à partir du fichier XML.

La génération est effectuée via un fichier XSLT, qui permet de formater autrement le contenu d'un fichier XML.

Examinons la structure du fichier glossaire.xml :

```
<glossary>

  <word id="cnx.in"><![CDATA[<strong><a name="Connexion entrante"/>Connexion
entrante</strong> :
Réception d'informations provenant d'un autre ordinateur.]]></word>

  <word id="cnx.out"><![CDATA[<strong>Connexion sortante</strong> :
Envoi d'informations de son ordinateur &agrave; un autre.]]></word>

</glossary>
```

L'élément glossary occupe la position racine du document. Il contient l'ensemble des mots à définir sous la forme d'éléments word.

Un élément word spécifie un mot du glossaire. Il contient un attribut obligatoire id, à renseigner par une chaîne de caractères unique qui permet de faire référence à ce mot depuis l'application. Le contenu de l'élément est un bloc CDATA avec la définition formatée en HTML, qui sera affiché à la fois dans la fenêtre Ansmo et dans le navigateur.

Il est donc très simple d'ajouter de nouvelles entrées dans ce fichier. Veuillez toutefois noter qu'il est vivement conseillé de placer les nouveaux mots dans le

fichier en tenant compte de l'ordre alphabétique, afin que le glossaire html ne soit pas déroutant pour l'utilisateur.

La génération du glossaire html se fait via un outil externe lors de l'empaquetage de l'application (xsltproc). Si l'on souhaite appliquer les modifications directement suite à une modification du fichier xml, il est nécessaire de relancer la génération du document manuellement.

Cela se fait tout simplement en exécutant le script présent dans le répertoire de l'application, ou bien par la commande suivante :

```
xsltproc -o data/glossaire.html glossaire.xsl glossaire.xml
```

4.GUIDE DU MAINTENEUR

4.1Installation des environnements de construction

4.1.1Ubuntu 7.10 Installation pas à pas

4.1.1.1Pré requis

Afin de pouvoir installer l'environnement de construction, il est nécessaire au préalable d'avoir une connexion internet et les fichiers suivants : (livrés dans le cd ANSMO\Dev\Linux ou téléchargeables sur Internet) :

10 Go d'espace disque sont requis pour la compilation de tous les composants.

- Les sources x11 de Qt 4.3.5 http://trolltech.com/developer/downloads/qt/x11
  qt-x11-opensource-src-4.3.5.tar.gz (40.3 mo)

- Les sources de python 2.5.2
  http://www.python.org/download/
  Python-2.5.2.tgz (10.5 mo)

- Les sources du snapshot PyQT 4.3 (au 12/02/2008)
  PyQt-x11-gpl-4.3.4-snapshot-20080207.tar.gz (5.9 mo)

- Les sources du snapshot de SIP 4.7 (au 12/02/2008)
  sip-4.7.4-snapshot-20080209.tar.gz (437 ko)

- Les sources de PyInstaller (la version stable à ce jour ne permet pas de construire le binaire)

  Voici la commande à exécuter pour télécharger la version svn de PyInstaller :

```
svn co http://svn.pyinstaller.python-hosting.com/trunk pyinstaller
```

Nous recommendons d'utiliser le fichier pyinstaller.svn.080212.tar est présent dans le cd ANSMO.

4.1.1.2Installation et configuration du chroot

Pour pouvoir construire le binaire avec le moins de dépendance possibles

Puisque notre binaire est amené à être exécuté sur un large panel de distribution linux (Debian, Ubuntu, Mandriva, Fedora et OpenSuse), il est recommandé d'utiliser une distribution ayant une libc non récente.

L'environnement ubuntu/dapper qui dispose d'une version relativement ancienne de la libc (2.3) sans pour autant être obsolète convient parfaitement.

Pour l'installation du chroot, il est nécessaire au préalable d'installer les paquets debootstrap et schroot sur le système hôte.

```
sudo apt-get install debootstrap schroot
```

Ensuite pour installer le chroot, taper :

```
cd \
sudo debootstrap --arch i386 ansmo dapper http://archive.ubuntu.com/ubuntu
```

L'installation du chroot peut prendre du temps, environ 1h avec une machine récente et une connexion ADSL 5mbits (en fonction de la configuration de la machine et du débit de la connexion internet).

Une fois l'installation terminée, il faut configurer schroot.
shroot permet de lancer des commandes dans le chroot depuis l'hôte afin de faciliter la construction du binaire.

Pour cela, il faut éditer le fichier /etc/schroot/schroot.conf en ajoutant les lignes suivantes à la fin du fichier :

```
[ansmo]
description=ansmo
priority=2
location=/ansmo
aliases=ansmo,default
```

Il faut à présent copier les différents fichiers des pré requis dans le chroot.
Créer un dossier « packages » et y placer les fichiers :

- qt-x11-opensource-src-4.3.5.tar.gz
- Python-2.5.2.tgz

PyQt-x11-gpl-4.3.4-snapshot-20080207.tar.gzsip-4.7.4-snapshot-20080209.tar.gz
copier le dossier packages à la racine du chroot :

```
cp -r <path>/packages /ansmo
```

Entrer dans le chroot

```
sudo chroot /ansmo
```

Voici les différents paquets à installer dans le chroot :

```
apt-get update --yes --force-yes
apt-get install --yes --force-yes build-essential libxrender-dev xsltproc libxext-dev libz-
dev recode libfontconfig1-dev imagemagick
```

4.1.1.3Compilation de QT4

Voici les commandes à entrer dans le chroot :

```
cd /packages
tar zxf qt-x11-opensource-src-4.3.5.tar.gz
cd qt-x11-opensource-src-4.3.5
./configure -confirm-license -no-largefile -no-exceptions -no-sql-ibase -no-sql-mysqk -no-
sql-odbc -no-sql-psql -no-sql-sqlite -no-sql-sqlite2 -no-qt3support -qt-zlib -no-gif -no-
libtiff -qt-libpng -no-libmng -qt-libjpeg -no-openssl -no-nas-sound -no-sm -no-xinerama
-xrender -fontconfig -no-tablet -no-xkb -no-glib -no-separate-debug-info -static && make &&
make install
```

4.1.1.4Compilation de Python 2.5

Voici les commandes à entrer dans le chroot :

```
cd /packages && tar zxf Python-2.5.2.tgz && cd Python-2.5.2 && ./configure && make && make
install
```

4.1.1.5Compilation de SIP 4

Voici les commandes à entrer dans le chroot :

```
cd /packages
tar zxf sip-4.7.4-snapshot-20080209.tar.gz
cd sip-4.7.4-snapshot-20080209
python configure.py && make && make install
```

4.1.1.6Compilation de PyQT 4

Voici les commandes à entrer dans le chroot :

```
cd /packages
tar zxf PyQt-x11-gpl-4.3.4-snapshot-20080207.tar.gz
cd PyQt-x11-gpl-4.3.4-snapshot-20080207
python configure.py --confirm-license -q /usr/local/Trolltech/Qt-4.3.5/bin/qmake -g -e QtGui
-e QtXml && make && make install
```

4.1.1.7Installation de PyInstaller

Voici les commandes à entrer dans le chroot :

```
cd /packages
tar xf pyinstaller.svn.080212.tar
cd pyinstaller/source/linux
python Make.py && make
cd ../..
python Configure.py
```

4.1.2Ubuntu 7.10 Installation scriptée

Des scripts ont été réalisés pour faciliter l'installation et la configuration du chroot.

Copier le contenu du dossier linux-chroot sur la machine hôte

Ce dossier contient :

```
\linux-chroot
|   1-create-chroot.sh
|   2-configure-chroot.sh
|
\---packages
        install.sh
        pyinstaller.svn.080212.tar
        PyQt-x11-gpl-4.3.4-snapshot-20080207.tar.gz
        Python-2.5.1.tgz
        qt-x11-opensource-src-4.3.5.tar.gz
        sip-4.7.4-snapshot-20080209.tar.gz
```

L'installation est presque complètement automatisée. Toutes les étapes sont réalisées sur le système hôte. L'installation se réalise en 3 étapes :

4.1.2.1Lancement du script 1-create-chroot.sh

Entrer dans le dossier linux-chroot et taper :

```
sh ./1-create-chroot.sh
```

4.1.2.2Edition du fichier /etc/schroot/schroot.conf

Ajouter les lignes suivantes à la fin du fichier :

```
[ansmo]
description=ansmo
priority=2
location=/ansmo
aliases=ansmo,default
```

4.1.2.3Lancement du script 2-configure-chroot.sh

Entrer dans le dossier linux-chroot et taper :

```
sh ./2-configure-chroot.sh
```

4.1.3Windows XP SP2

4.1.3.1Pré requis

L'installation nécessite une connexion internet pour télécharger les différents composants ou en utilisant ceux présents dans le cd ANSMO\Dev\Windows.

Afin de pouvoir installer l'environnement de construction, il est nécessaire au préalable d'avoir une connexion internet et les fichiers suivants : (livrés dans le cd ANSMO ou téléchargeables sur Internet) :

1 Go d'espace disque sont requis pour l'installation de tous les composants.

- Les binaires win32 de Qt 4.3.5
  http://trolltech.com/developer/downloads/qt/windows
  qt-win-opensource-4.3.5-mingw.exe (69.6 Mo)

- Les binaires win32 de python 2.5.2
  http://www.python.org/download/
  python-2.5.2.msi (10.4 Mo)

- Les sources de PyQT 4.3 et de SIP 4.7
  PyQt-win-gpl-4.3.4-snapshot-20080207.zip (7.1 Mo)
  sip-4.7.4-snapshot-20080209.zip (503 ko)

- Les binaires de py2exe 0.6.6 pour python 2.5
  http://sourceforge.net/project/showfiles.php?group_id=15583
  py2exe-0.6.6.win32-py2.5.exe

- Les binaires Win32 Extensions pour python 2.5
  http://sourceforge.net/project/platformdownload.php?group_id=78018
  pywin32-210.win32-py2.5.exe

4.1.3.2Installation de Python 2.5

Double cliquer sur le fichier python-2.5.2.msi et suivre les instructions. Choisir les options proposées par défaut.

4.1.3.3Installation des extensions win32 pour Python 2.5

Double cliquer sur le fichier pywin32-210.win32-py2.5.exe et suivre les instructions. Choisir les options proposées par défaut.

4.1.3.4Installation de QT 4

Double cliquer sur le fichier qt-win-opensource-4.3.5-mingw.exe et suivre les instructions. Choisir les options proposées par défaut **sauf pour l'installation de mingw** (**veillez à installer le compilateur**). Il sera téléchargé et installé automatiquement dans C:\MinGW.

4.1.3.5Compilation de SIP 4 et de PyQT 4

Décompresser les archives PyQt-win-gpl-4.3.4-snapshot-20080207.zip et sip-4.7.4-snapshot-20080209.zip dans c:\ .

L'arborescence doit ressembler à ceci :

```
C:\PYQT-WIN-GPL-4.3.4-SNAPSHOT-20080207
+---contrib
+---debug
+---designer
...
+---QtXml
+---release
+---sip

C:\SIP-4.7.4-SNAPSHOT-20080209
+---custom
+---doc
+---sipgen
```

```
+---siplib
\---specs
```

Ouvrir une invite de commande (démarrer / exécuter / cmd.exe)

Pour compiler SIP, entrer les commandes suivantes :

```
PATH = %PATH%;C:\MinGW\bin;C:\Qt\4.3.5\bin;C:\Python25
cd C:\sip-4.7.4-snapshot-20080209
python configure.py -p win32-g++ && mingw32-make.exe && mingw32-make.exe install
```

Pour compiler PyQT, entrer les commandes suivantes :

```
PATH = %PATH%;C:\MinGW\bin;C:\Qt\4.3.5\bin;C:\Python25
cd C:\PYQT-WIN-GPL-4.3.4-SNAPSHOT-20080207
python configure.py && mingw32-make.exe && mingw32-make.exe install
```

Ajouter le path suivant au système :
Bouton droit sur Poste de travail, propriétés
Onglet Avancé, cliquer sur Variables d'environnement
Dans la liste du bas, chercher PATH et ajouter à la fin de l'existant :

```
C:\Qt\4.3.5\bin;C:\Python25
```

4.1.3.6Installation de Py2exe

Double cliquer sur le fichier py2exe-0.6.6.win32-py2.5.exe et suivre les instructions. Choisir les options proposées par défaut.

4.1.4Mac OS X 10.4.10

4.1.4.1Pré requis

L'installation nécessite une connexion internet pour télécharger les différents composants ou en utilisant ceux présents dans le cd ANSMO\Dev\MacOSX.

Afin de pouvoir installer l'environnement de construction, il est nécessaire au préalable d'avoir une connexion internet et les fichiers suivants : (livrés dans le cd ANSMO ou téléchargeables sur Internet) :

1 Go d'espace disque sont requis pour l'installation de tous les composants.

- Les binaires macosx de Qt 4.3.5
  http://trolltech.com/developer/downloads/qt/mac
  qt-mac-opensource-4.3.5.dmg (67 mo)

- Les binaires macosx de python 2.5.2
  http://www.python.org/download/
  python-2.5.2-macosx.dmg (17.8 mo)

- Les sources du dernier snapshot PyQT 4.3 (au 12/02/2008)
  PyQt-mac-gpl-4.3.4-snapshot-20080207.tar.gz (5.9 mo)

- Les sources du dernier snapshot de SIP 4.7 (au 12/02/2008) sip-4.7.4-snapshot-20080209.tar.gz (437 ko)

4.1.4.2Installation de Python 2.5

Double cliquer sur le fichier python-2.5.2-macosx.dmg puis dans le volume « Universal MacPython 2.5.2 » double cliquer sur MacPython.mpkg et suivre les instructions. Choisir les options proposées par défaut.

4.1.4.3Installation de QT 4.3.5

Double cliquer sur le fichier qt-mac-opensource-4.3.5.dmg puis dans le volume « Qt 4.3.5 » double cliquer sur Qt.mpkg et suivre les instructions. Choisir les options proposées par défaut.

4.1.4.4Installation de PyQT 4.3 et Sip 4.7

Ouvrir un terminal et se placer dans le dossier contenant les fichiers sip-4.7.4-snapshot-20080209.tar.gz et PyQt-mac-gpl-4.3.4-snapshot-20080207.tar.gz

Décompresser les fichiers :

```
tar xzf sip-4.7.4-snapshot-20080209.tar.gz
tar xzf PyQt-mac-gpl-4.3.4-snapshot-20080207.tar.gz
```

Compiler et installer SIP: (**Attention à ne pas oublier -n dans la ligne de commande**)

```
cd sip-4.7.4-snapshot-20080209
python configure.py -n && make && make install
```

Compiler et installer PyQt :

```
cd PyQt-mac-gpl-4.3.4-snapshot-20080207
python configure.py --confirm-license && make && make install
```

4.1.4.5Installation de Py2app

Lancer un terminal et taper :

```
curl -O http://peak.telecommunity.com/dist/ez_setup.py
sudo python ez_setup.py -U setuptools
sudo easy_install -U py2app
```

4.2Création de la distribution binaire à partir des sources

4.2.1 Ubuntu 7.10

Copier les sources ansmoxxx.zip dans un dossier sur la machine configurée tel que décrit dans le chapitre 4.1.1 ou 4.1.2.

Décompresser les sources et lancer un terminal et taper :

```
cd <path des sources décompressées>
sh ./linux_build.sh
```

Le script effectue les actions suivantes :

- copie des sources dans le chroot ;
- lancement de la commande make dans le chroot ;
- recopie du dossier contenant le binaire et le dossier data hors du chroot ;
- et lancement du binaire pour vérifier le fonctionnement.

Les fichiers linux_build.sh et Makefile présent dans les sources permettent d'obtenir le détail des actions.

4.2.2 Windows XP SP2

Copier les sources ansmoxxx.zip dans un dossier sur la machine configurée tel que décrit dans le chapitre 4.1.3.

Décompresser les sources et double cliquer sur win32_build.cmd

Ce script effectue génère le binaire win32 via py2exe et copie les fichiers nécessaires (dossier data et quelques dll nécessaires à l'exécution sous windows 2000).

Un dossier ansmo_bin_<date>_win32 a été créé contenant les binaires et le dossier data.

4.2.3 Mac OS X 10.4.10

Copier les sources ansmoxxx.zip dans un dossier sur la machine configurée tel que décrit dans le chapitre 4.1.4.

Décompresser les sources et double cliquer sur osx_build.cmd.

```
cd <path des sources décompressées>
sh ./osx_build.sh
```

Ce script effectue génère le binaire osx via py2app et copie les fichiers nécessaires (dossier data).

Un dossier ansmo_bin_<date>_osx a été créé contenant les binaires (le dossier data est inclus dans l'application).