Enregistrement des services dans la plate-forme d’accueil FrameKit

Fabrice Kordon,
Laboratoire d’Informatique de Paris 6,
4 place Jussieu, 75252 Paris Cedex 05,
E-mail: `Fabrice.Kordon@lip6.fr`

Résumé : Framekit est une plate-forme d’accueil dédié au prototypage rapide d'environnements de Génie Logiciel. L’un de ses objectif est d’assurer la correspondance entre des services (fonctions accessibles à un utilisateur) et des outils (les programmes qui assurent ces fonctions). Ce rapport présente les mécanismes de déclaration des services dans la Plate-forme FrameKit et illustre l’utilisation des mécanismes offerts à travers quelques exemples. La particularité de ce mécanisme est d’être complètement dynamique et de permettre la déclaration d’outils sans recompilation de la plate-forme.

Mots clefs : Environnement de Génie Logiciel, Intégration d’outils

Abstract : Framekit is a software platform dedicated to the prototyping of CASE environments. One of its objective is to establish a relation between services (functions provided to a user) and tools (program that execute the corresponding actions). This report presents service declaration mechanisms in FrameKit and illustrates their use by means of examples. One characteristics of the proposed mechanisms is to allow service declaration without any recompilation of FrameKit.

Key-words : CASE environment, Tool integration

1. Introduction

FrameKit **[1, 2, 3]** est une plate-forme d’accueil dédié au prototypage rapide d'environnements de Génie Logiciel. Une plate-forme d’accueil doit permettre la manipulation de modèles graphiques et hiérarchiques, l’application de services sur ces modèles et le stockage des résultats obtenus et dispose de fonctions permettant une administration simple d’entités comme les utilisateurs, les services etc.

Pour assurer ces différents objectifs, FrameKit dispose :

- d’une interface utilisateur paramétrable (Macao **[4]**) pour étudier, mettre au point puis utiliser des formalismes graphiques et hiérarchiques,
- de bibliothèques de composants facilitant le développement d'applications dans le contexte de cette plate-forme afin d’offrir aux chercheurs un environnement de développement cohérent,
- de mécanismes d'intégration permettant d'importer des logiciels développés par d’autres,
- de techniques de développement multi-cibles pour assurer une bonne diffusion du «produit». Cette caractéristique doit concerner aussi bien la

plate-forme elle-même que les composants offerts aux programmeurs d'outils,

- de mécanismes d'administration et de diffusion assurant une administration simple et proposant un gabarit pour faciliter la diffusion d'outils.

FrameKit est offre des services réalisés par des outils. Ces outils travaillent sur des modèkes décrits dans un formalisme (un format de représentation) auquel il faut associer une liste de services. Le menu de services associé à un formalisme est construit à partir de descriptions parcellaires, chacune étant liée à un kit de distribution différent. A cette fin, il existe deux niveaux d'accès aux informations :

- le niveau ***formalismes*** repère un catalogue de services dédiés à un formalisme. La liste est décrite dans un ***fichier de référence racine*** qui contient la liste descriptions de services;
- le niveau ***service*** qui caractérise avec unicité les fonctionnalités d'un outil. La description de ces services est contenue dans un ***fichier de services***.

L'accessibilité des services dans la plate-forme FrameKit peut être calculée en fonction :

- des ***droits d'accès*** déterminent si un usager connecté a accès ou non à un service. Les entrées auxquelles l'usager n'a pas accès n'apparaîtront pas dans le menu de services présenté par l'interface utilisateur Macao;
- des ***conditions*** constituent des gardes inactivant temporairement une entrée du menu de service. Ces conditions permettent par exemple de définir un prérequis pour un service (par exemple, A ne peut être lancé que si B a été préalablement invoqué).

Le premier mécanisme, statique, est évalué à la connexion. Le second, dynamique, est recalculé après chaque invocation de service.

2. Description des menus

2.1. description d'un menu de service

Deux types de fichiers interviennent dans la description des menus de service :

- les *fichiers de services* qui permettent de définir un groupe de services éventuellement décomposable en sous-services;
- les *fichiers de références* qui permettent de définir une liste des fichiers de services à utiliser pour construire un menu de services. Ces fichiers sont utiles pour :
  - définir la liste des services accessibles pour un formalisme donné,
  - définir la liste des entrées composant un menu hiérarchique,

Exemple 1 : Construction d'un menu a l'aide de plusieurs fichiers.

L'enchaînement de fichiers (a) permet de construire le menu (b). Un fichier de référence, associé à un formalisme, permet de référencer les fichiers de services lserv1 et lserv2. lserv2, pour définir les composantes de l'entrée de menu serv3, renvoie vers un fichier de références lserv3 qui permet de pointer sur un fichier de services (on aurait pu en avoir plusieurs).

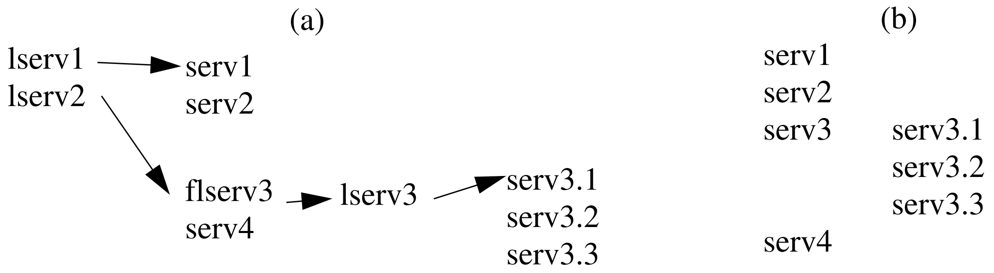

Figure 1 : Construction d'un menu hiérarchique par enchaînement de services.

2.2. Droits d'accès aux services

Il est possible de gérer les accès aux services associés à un formalisme en fonction de différents critères :

- l'architecture de la plate-forme à laquelle se connecte un usager. Cela est utile lorsque plusieurs architectures différentes sont installées sur un même site; on gère l'existence d'un service en fonction du format des exécutables présents;
- le nom de l'usager ou son appartenance à un groupe d'utilisateurs, ce qui permet de différentier les différents rôles des utilisateurs.

2.3. Conditions de services

Outre le mécanisme défini précédemment, FrameKit permet de rendre accessible un service en fonction de l'utilisation d'autres services ou de résultats d'exécution d'autres services. Cela permet de gérer des dépendances entre services.

Exemple 2 : Dépendance entre services.

Considérons un menu composé de deux services : «compilation» et «édition des liens». Le second service ne peut-être invoqué que si le premier a été lancé avec succès. Pour réaliser cela, il faut associer au service «édition des liens» une précondition stipulant que son accessibilité est soumise à une exécution réussie du service «compilation».

Dans FrameKit, ces conditions sont définies au moyen d'indicateurs associés aux services (repérés par leur nom interne) et de variables de session

Indicateurs associés aux services

Trois indicateurs sont associés au nom interne d'un service et prennent la valeur «vrai» ou «faux», permettant ainsi de définir des conditions élémentaires :

- `LAUNCHED_OK` est vrai lorsque le service correspondant a été invoqué avec succès (valeur initiale : faux);
- `LAUNCHED_PB` est vrai lorsque le service correspondant a été invoqué mais qu'une erreur a été signalée à la plate-forme (valeur initiale: faux);
- `NEVER_LAUNCHED` est vrai lorsque le service correspondant n'a jamais été invoqué (valeur initiale : vrai).

Les conditions de services sont mémorisées d'une session à l'autre. Seule la modification du modèle repositionne ces indicateurs à leur valeur initiale.

Les variables de session

Une variable de session contiennent une chaîne de caractère. Par convention, une variable vide n'existe pas (l'affectation d'une chaîne vide correspond à la destruction de la variable). Il existe deux types de variables :

- Les variables permanentes dont la valeur est préservée d'une session à l'autre;
- Les variables temporaires dont la valeur est perdue à la fin d'une session.

La création ou la destruction de ces variables est à la charge des outils(1) qui peuvent ainsi influer sur le menu de service sans en connaître la structure. Les variables de session obéissent aux règles suivantes :

- Le type d'une variable (permanente, temporaire) est défini au moment de sa création par un outil;
- Toute affectation d'une valeur dans une variable entraîne la perte de la valeur précédente;
- Il est possible de changer la valeur d'une variable mais elle conserve le type donné lors de sa création(2).

Il est possible de comparer la valeur de variables de sessions entre elles ou à des constantes dans une condition de service. Deux opérateurs de comparaison sont disponibles : `=` (égalité) et `/=` (différence). On ne dispose pas de tests d'existence mais, comme une variable vide n'existe pas, on peut procéder comme suit :

- `%ma_variable = ''` teste l'absence d'une variable;
- `%ma_variable /= ''` teste l'existence d'une variable.

2.4. Déclaration d'un service

Lorsque l'on intègre un service, un certain nombre d'informations permettant de décrire son interactivité et la manière dont il sera invoqué doivent être fournies à la plate-forme. Ces informations sont indiquées dans les fichiers de services.

Les informations nécessaires sont les suivantes :

- ***Nom du service***

  Il s'agit de définir le service tel qu'il apparaîtra dans le menu présenté par l'interface utilisateur.
- ***Nom interne du service***

  Ce nom interne correspond à un identificateur du service indépendant de la langue de l'utilisateur, contrairement au nom du service.
- ***Le nom du programme associé au service***

(1) Au moyen d'un message CAMI encapsulé par des API pour les langages Ada, C et shell.

(2) Par exemple, si on «recouvre» la variable `X` permanente ayant la valeur «ABC» en demandant la création d'une variable `X` non permanente de valeur «DEF», on se retrouvera avec une variable X permanente de valeur «DEF» (le type de la variable lors de sa création est conservé). Pour obtenir une variable `X` temporaire avec la valeur «DEF», il faudrait préalablement détruire la première variable X.

Il s'agit d'indiquer à la plate-forme le nom du fichier exécutable qui devra être invoqué lorsque le service sera demandé. A partir du nom du fichier exécutable et des informations concernant le formalisme et l'architecture, le chemin absolu[(3)] du programme sera déduit.

- ***Les arguments du programme associé au service***

  Il s'agit de définir les arguments requis par le programme pour réaliser le service considéré. Ces arguments peuvent être indiqués de manière inconditionnelle ou optionnelle. Dans le premier cas, l'argument est toujours transmis lors de l'invocation de l'outil. Dans le second cas, sa présence est liée à la sélection d'une option dans l'arborescence associée au service.
- ***Mode d'interaction de l'outil avec la plate-forme***

  Il s'agit de définir le type de communication entre l'outil et la plate-forme. Les conventions disponibles dans FrameKit sont indiquées page 16.
- ***La «sûreté protocolaire» de l'outil***

  Pour être intégré dans la plate-forme, un outil doit respecter quelques conventions strictes parmi lesquelles figurent l'émission de messages indiquant le type de terminaison de l'outil (prise en charge par les API de FrameKit).

  Pour certains langages de programmation, il est possible de garantir qu'un message de terminaison, même anormale, sera transmis à la plate-forme. Si tel n'est pas le cas, on la déclare «protocolairement non sûre» et FrameKit gèrera de manière particulière l'invocation de l'outil. Le mécanisme mis en œuvre étant coûteux, il est intéressant de différencier les deux types d'applications[(4)].
- ***Informations diverses concernant l'exécution***

  Sont regroupées dans cette rubrique, un certain nombre d'indicateurs liés au mode de fonctionnement de l'outil et à la façon dont son exécution est reportée par l'interface utilisateur :

  - **stop** précise si l'application accepte de recevoir un message particulier lui demandant d'arrêter prématurément son exécution ou non. Si cet indicateur est positionné, FrameKit précisera à l'interface utilisateur qu'elle peut rendre visible un bouton provoquant l'arrêt du service correspondant.
  - **historic** active une fenêtre d'historique au niveau de l'interface utilisateur. Cette fenêtre d'historique contient le bouton «stop» évoqué précédemment (si l'indicateur est positionné) et présente les messages que l'application transmet à l'utilisateur pour indiquer les étapes du traitement en cours.
  - **info_supp** peut prendre trois valeurs; «no_info» précise que le service peut s'exécuter sans information particulière; «info_object» précise que

(3) Au sens Unix du terme.

(4) Actuellement, seules les applications écrite en C sont considérées comme tel. L'exécution des script shell est effectuée d'une autre manière et les API Ada exploitent le mécanisme d'exception pour garantir la sûreté protocolaire.

l'utilisateur doit préalablement au lancement d'un service sélectionner au moins un objet dans le modèle dont le ou les identificateurs sont transmis à l'application; «info_text» précise que l'utilisateur doit préalablement au lancement d'un sélectionner un texte qui sera transmis à l'application.

- **formalism** précise le nom du formalisme du résultat construit par le service. Trois possibilités sont offertes : «no_formalism» signifie qu'aucun modèle n'est créé par l'application; «input_formalism» indique que le formalisme du modèle résultat est celui du modèle en entrée; enfin, il est possible de donner explicitement (sous forme d'une chaîne de caractères) le nom du formalisme résultat.

2.4.1. Un exemple de service et sa description

Considérons le menu de service donné en Figure 2. Il est composé d'une arborescence («le service») contenant l'invocation du service et deux options. On souhaite que l'option 1 soit positionnée par défaut et que l'option 2 ne le soit pas. Par ailleurs, on considère que ce service et ses options sont accessibles par tous les utilisateurs susceptibles de se connecter. On considère que l'exécution du service ou l'activation d'une option n'est pas soumise à une précondition.

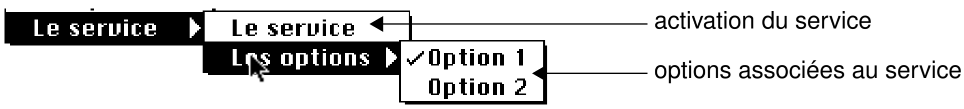

Figure 2 : Exemple de menu associant deux options à un service.

Pour décrire le menu, nous supposons que nous disposons des informations suivantes :

- L'exécutable associé s'appelle `my_tool`. Il est situé dans le répertoire `/MY_TOOL`[5]. et le nom interne associé au service est «`MY_TOOL`»;
- En invoquant cet exécutable, on peut transmettre trois paramètres en ligne de commande. Les deux premiers sont optionnels (ils correspondent aux deux options) mais le troisième est obligatoire. Le schéma d'invocation de l'outil est ainsi : `my_tool [-opt1] [-opt2] -serv`
- Par ailleurs, l'exécutable associé au service :
  - sait communiquer au moyen de tubes nommés,
  - est considéré comme protocolairement sûr,
  - ne sait pas traiter les événements de type «arrêt» transmis par l'interface utilisateur (pas de bouton stop dans la fenêtre d'historique),
  - envoie des informations sur son état d'exécution (création de la fenêtre historique),
  - ne nécessite pas d'informations supplémentaires (texte ou objets désignés dans le modèle),
  - construit un résultat dans le même formalisme que le modèle pour

(5) Nous verrons page 9 comment référencer de manière relative des fichiers exécutables via des variables d'environnement (nécessaires pour référencer un programme exécutable relativement à une architecture).

lequel il a été invoqué (il peut faire partie de menus de services associés à des formalismes différents).

Nous donnons ci-dessous la description de l'entrée de menu terminale associée à l'invocation de l'outil. Les paramètres optionnels sont définis au moyen de variable (ici, P1 et P2) qui seront associées à des options : si l'entrée de menu correspondante est «sélectionnée», la variable sera remplacée par la valeur effective correspondante; dans le cas contraire, la chaîne vide lui sera substituée. Le troisième paramètre, lui, est toujours présent.

```
SERVICE (TERMINAL, 'Le service')
   ACCESS INCLUSIVE
      ALL_USERS
   END_ACCESS
   BEGIN_PRECONDITION
      TRUE
   END_PRECONDITION
   EXECUTABLE ('/MY_TOOL/my_tool', '$P1 $P2 -serv1', COMM_NAMMED_PIPE, 'MY_TOOL')
   PROTOCOL (SAFE)
   QUESTION_INFO (STOP_NOT_ALLOWED, HISTORIC, NO_INFO, INPUT_FORMALISM)
END_SERVICE
```

La description suivante correspond aux options du service. La première est par défaut positionnée, la seconde non. A chaque option, on associe une variable et sa valeur de remplacement.

```
SERVICE (LST_CHECK_MARKS, 'Les options')
   ACCESS INCLUSIVE
      ALL_USERS
   END_ACCESS
   BEGIN_PRECONDITION
      TRUE
   END_PRECONDITION
   SERVICE (CHECK_MARK, 'Option 1')
      ACCESS INCLUSIVE
         ALL_USERS
      END_ACCESS
      BEGIN_PRECONDITION
         TRUE
      END_PRECONDITION
      CHECK_MARK (ON, P1, '-opt1')
   END_SERVICE
   SERVICE (CHECK_MARK, 'Option 2')
      ACCESS INCLUSIVE
         ALL_USERS
      END_ACCESS
      BEGIN_PRECONDITION
         TRUE
      END_PRECONDITION
      CHECK_MARK (OFF, P2, '-opt2')
   END_SERVICE
END_SERVICE
```

Conditions et les droits d'accès peuvent être définis à tous les niveaux d'un menu de services. On peut ainsi facilement activer ou désactiver, statiquement (au niveau des droits) ou dynamiquement (au niveau des préconditions), des entrées de menu (services ou options).

2.4.2. Définition des accès aux utilisateurs

Les droits d'accès sont positionnés pour une liste d'usagers, une liste de groupes (au sens Unix du terme) ou en fonction d'architectures d'exécution de la plate-forme.

Considérons le service ci après :

```
SERVICE (TERMINAL, 'Le service 2')
   ACCESS INCLUSIVE
      USER ('titi')
      USER ('rominet')
      GROUP ('dogs')
   END_ACCESS
   BEGIN_PRECONDITION
      TRUE
   END_PRECONDITION
   EXECUTABLE ('/MY_TOOL/my_tool', '-serv1', COMM_NAMMED_PIPE, 'MY_TOOL')
   PROTOCOL (SAFE)
   QUESTION_INFO (STOP_NOT_ALLOWED, HISTORIC, NO_INFO, '')
END_SERVICE
```

Dans l'exemple ci après, l'accès à «le service» est ici précisé de manière ***inclusive***. Cela signifie que la liste indiquée définir les usagers qui ont accès au services (ici, les usagers «titi», «romninets» et ceux appartenant au groupe «dogs»). Lorsque les accès sont définis de manière ***exclusive***, la liste indiquerait les usagers qui n'ont pas accès au service (ici, il s'agirait de tous les usagers sauf «titi», «rominet» et les membres du groupe «dogs»).

Dans l'exemple ci après, sont défini deux services accessibles à tous les utilisateurs qui se connectent sur une plate-forme. Ces deux services diffèrent par le format du fichier exécutable de l'outil «my_tool» (format Solaris ou format HP-UX). Imaginons que, sur un site, FrameKit soit installé sur trois architectures (Sun/SunOS, Sun/Solaris et HP/HP-UX). Considérons un usager qui se connecte sur :

- la plate-forme Sun/SunOS, il ne verra pas le service «le service» (qui n'est disponible sur aucune architecture dans l'exemple considéré ici);
- la plate-forme Sun/Solaris, il verra le service «le service (sur Solaris)» puisqu'il est disponible pour celle architecture;
- la plate-forme HP/HP-UX, il verra le service «le service (sur HP)» puisqu'il est disponible pour celle architecture.

```
SERVICE (TERMINAL, 'Le service 3 (sur Solaris)')
   ACCESS INCLUSIVE
      ARCHITECTURE ('SUN_SOLARIS')
      ALL_USERS
   END_ACCESS
   BEGIN_PRECONDITION
      TRUE
   END_PRECONDITION
   EXECUTABLE ('/MY_TOOL/my_tool_s', '-serv1', COMM_NAMMED_PIPE, 'MY_TOOL')
   PROTOCOL (SAFE)
   QUESTION_INFO (STOP_NOT_ALLOWED, HISTORIC, NO_INFO, '')
END_SERVICE
```

```
SERVICE (TERMINAL, `Le service 3 (sur HP)')
   ACCESS INCLUSIVE
      ARCHITECTURE ('HP_HPUX')
      ALL_USERS
   END_ACCESS
   BEGIN_PRECONDITION
      TRUE
   END_PRECONDITION
   EXECUTABLE (`/MY_TOOL/my_tool_hp', `-serv1', COMM_NAMMED_PIPE, `MY_TOOL')
   PROTOCOL (SAFE)
   QUESTION_INFO (STOP_NOT_ALLOWED, HISTORIC, NO_INFO, ``)
END_SERVICE
```

Ainsi, on peut installer sur une même machine, plusieurs plate-formes ayant des caractéristiques différentes et qui s'exécutent forcément sur des ***machines*** différentes. Ce système d'architecture permet centraliser les données d'administration d'une plate-forme.

Les deux types de droits d'accès se combinent parfaitement. Dans l'exemple ci dessous, «le service» est accessible pour les utilisateurs «***titi***», «***rominet***» et ceux qui appartiennent au groupe «***dogs***» lorsqu'ils se connectent sur les machines qui exécutent la plate-forme pour les architectures «`SUN_SOLARIS`» ou «`SUN_SUNOS`».

```
SERVICE (TERMINAL, `Le service 4')
   ACCESS INCLUSIVE
      USER ('titi')
      USER ('rominet')
      GROUP ('dogs')
      ARCHITECTURE ('SUN_SOLARIS')
      ARCHITECTURE ('SUN_SUNOS')
   END_ACCESS
   BEGIN_PRECONDITION
      TRUE
   END_PRECONDITION
   EXECUTABLE (`/MY_TOOL/my_tool', `-serv1', COMM_NAMMED_PIPE, `MY_TOOL')
   PROTOCOL (SAFE)
   QUESTION_INFO (STOP_NOT_ALLOWED, HISTORIC, NO_INFO, ``)
END_SERVICE
```

2.4.3. Variables d'environnements pour référencer les fichiers de manière relative

Il est intéressant de référencer un exécutable relativement à une architecture en laissant FrameKit calculer le chemin absolu correspondant selon ses paramètres d'exécution. Pour cela, deux variables d'environnement peuvent être utilisées en paramètres de la commande `EXECUTABLE` dans le fichier de service. Ces variables d'environnements représentent le chemin calculé pour une architecture donnée :

- `FK_TOOLS_ROOT` représente la racine de la sous-arborescence des outils dans l'arborescence de la plate-forme;
- `FK_EXECUTABLE_ROOT` représente le répertoire contenant les exécutables dans l'arborescence de la plate-forme;

Les chemins référencés après une variable d'environnement doivent forcément être relatifs.

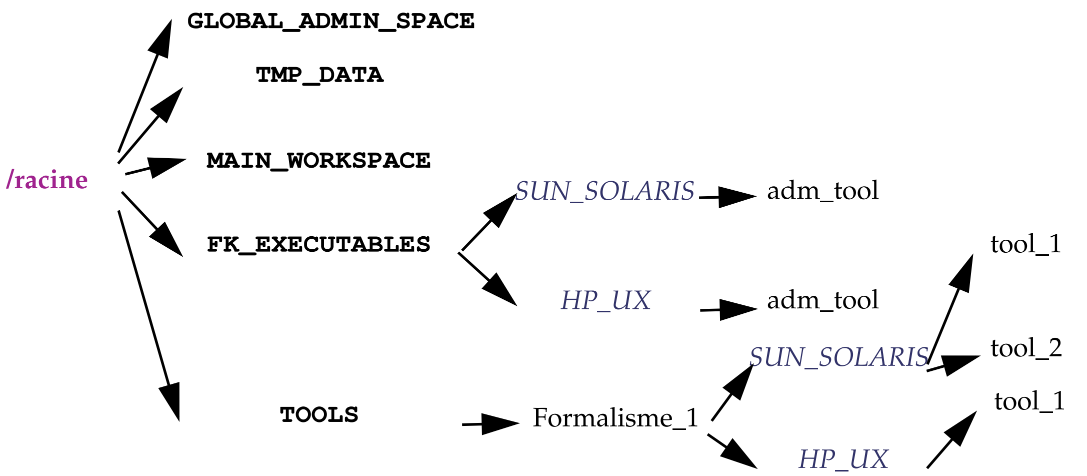

Figure 3 : Exemple d'arborescence pour une plate-forme FrameKit installée pour fonctionner à la fois sur Sun/Solaris et HP/HP-UX.

Considérons l'arborescence d'administration d'une plate-forme FrameKit donnée en Figure 3. La plate-forme fonctionne à la fois sur Sun/Solaris et HP/HP-UX. La référence à l'outil tool_1, qui fonctionne sur les deux architectures, peut se faire de la manière suivante :

```
SERVICE (TERMINAL, 'Le service 5')
   ACCESS INCLUSIVE
      ARCHITECTURE ('SUN_SOLARIS')
      ARCHITECTURE ('HP_UX')
      ALL_USERS
   END_ACCESS
   BEGIN_PRECONDITION
      TRUE
   END_PRECONDITION
   EXECUTABLE ($(FK_TOOLS_ROOT)'tool_1', '-serv1', COMM_NAMMED_PIPE, 'TOOL_1')
   PROTOCOL (SAFE)
   QUESTION_INFO (STOP_NOT_ALLOWED, HISTORIC, NO_INFO, INPUT_FORMALISM)
END_SERVICE
```

L'outil est ainsi accessible pour tous les utilisateurs sur les deux architectures considérées car la valeur calculée pour `FK_TOOLS_ROOT` tient compte du formalisme associé à la session et de l'architecture sur laquelle fonctionne la plate-forme. Si la plate-forme a été configurée correctement, on aura :

- sur Sun/Solaris : `/racine/TOOLS/FORMALISM_1/SUN_SOLARIS/`
- sun HP/HP-UX : `/racine/TOOLS/FORMALISM_1/HP_UX/`

L'outil 2 pourra être référencé de la même manière mais les droits d'accès ne mentionneront que l'architecture Sun/Solaris.

Pour les outils d'administration, le système fonctionne de la même manière. La déclaration de l'exécutable adm_tool (qui fonctionne également sur les deux architectures) se fera de la manière suivante :

```
SERVICE (TERMINAL, 'Le service d administration')
   ACCESS INCLUSIVE
      ARCHITECTURE ('SUN_SOLARIS')
      ARCHITECTURE ('HP_UX')
      ALL_USERS
   END_ACCESS
   BEGIN_PRECONDITION
      TRUE
   END_PRECONDITION
   EXECUTABLE ($(FK_EXECUTABLE_ROOT)'adm_tool', '-serv1', COMM_NAMMED_PIPE, 'XX')
   PROTOCOL (SAFE)
   QUESTION_INFO (STOP_NOT_ALLOWED, HISTORIC, NO_INFO, INPUT_FORMALISM)
END_SERVICE
```

La variable `FK_TOOLS_ROOT` prendra les valeurs suivantes:

- sur Sun/Solaris : `/racine/FK_EXECUTABLES/FK_SOLARIS/adm_tool`
- sun HP/HP-UX : `/racine/FK_EXECUTABLES/FK_HP/adm_tool`

Trois autres variables d'environnement, dont la valeur ne dépend pas de l'architecture, sont également définies :

- `FK_ADMIN_DIR` représente le répertoire contenant les données d'administration de la plate-forme(6);
- `FK_FORMALISM_ROOT` représente la racine associée à un formalisme.
- `FK_MAIN_WORKSPACE` représente le répertoire contenant les données globales de l'outil dans l'arborescence de la plate-forme (indiquée Figure 3). Cette variable est surtout utile pour désigner des paramètres (la référence à un fichier de configuration par exemple);

La dernière variable d'environnement est tout particulièrement dédiée à la référence des outils d'administration.

Si on se réfère à l'architecture, les variables `FK_ADMIN_DIR`, `FK_FORMALISM_ROOT` et `FK_MAIN_WORKSPACE` vaudront respectivement sur Sun/Solaris comme sur HP/HP-UX : `/racine/GLOBAL_ADMIN_SPACE` et `/racine/TOOLS/FORMALISM_1` et `/racine/MAN_WORKSPACE` si on considère une session ouverte pour un modèle de type «formalism_1».

2.4.4. Construction d'une condition de service

Considérons le menu de service suivant :

```
COMPILER          (nom interne CMP)
LINKER            (nom interne LNK)
VOIR LES WARNING  (nom interne VW)
VOIR LES ERREURS  (nom interne WE)
```

L'enchaînement de ces services doit obéir aux règles suivantes :

- L'opération de compilation ne peut se faire qu'une fois (pour relancer une compilation, il faut modifier le modèle);
- Le link ne se fait que si la compilation a été invoquée avec succès;
- on ne peut visualiser les warning que si la compilation a échoué en produisant des warnings;

(6) Utilisé pour désigner les fichiers de services d'administration, indépendants de tout formalisme et localisées avec les données d'administration de la plate-forme.

- on ne peut visualiser les warning que si la compilation a échoué en produisant des erreurs;

Pour que la plate-forme prenne en compte un enchaînement respectant ces règles, on peut se servir des indicateurs de services et de deux variables de session : `WARNING` et `ERRORS`. Ces variables permanentes sont crées par l'outil de compilation lorsqu'il rencontre des problèmes. Le menu ci après décrit un tel enchaînement :

```
SERVICE (TERMINAL, 'Compile')
   ACCESS INCLUSIVE
      ALL_USERS
   END_ACCESS
   BEGIN_PRECONDITION
      NEVER_LAUNCHED (CMP)
   END_PRECONDITION
   EXECUTABLE ($(FK_EXECUTABLE_ROOT)'compile', '', COMM_NAMMED_PIPE, CMP)
   PROTOCOL (SAFE)
   QUESTION_INFO (STOP_NOT_ALLOWED, HISTORIC, NO_INFO, NO_FORMALISM)
END_SERVICE
SERVICE (TERMINAL, 'Link')
   ACCESS INCLUSIVE
      ALL_USERS
   END_ACCESS
   BEGIN_PRECONDITION
      LAUNCHED_OK (CMP)
   END_PRECONDITION
   EXECUTABLE ($(FK_EXECUTABLE_ROOT)'link', '', COMM_NAMMED_PIPE, LNK)
   PROTOCOL (SAFE)
   QUESTION_INFO (STOP_NOT_ALLOWED, HISTORIC, NO_INFO, NO_FORMALISM)
END_SERVICE
SERVICE (TERMINAL, 'Voir les Warning')
   ACCESS INCLUSIVE
      ALL_USERS
   END_ACCESS
   BEGIN_PRECONDITION
      (LAUNCHED_PB (CMP)) and (%WARNING /= '')
   END_PRECONDITION
   EXECUTABLE ($(FK_EXECUTABLE_ROOT)'disp_warning', '', COMM_NAMMED_PIPE, VW)
   PROTOCOL (SAFE)
   QUESTION_INFO (STOP_NOT_ALLOWED, HISTORIC, NO_INFO, NO_FORMALISM)
END_SERVICE
SERVICE (TERMINAL, 'Le service d administration')
   ACCESS INCLUSIVE
      ALL_USERS
   END_ACCESS
   BEGIN_PRECONDITION
      (LAUNCHED_PB (CMP)) and (%ERROR /= '')
   END_PRECONDITION
   EXECUTABLE ($(FK_EXECUTABLE_ROOT)'disp_error', '', COMM_NAMMED_PIPE, VE)
   PROTOCOL (SAFE)
   QUESTION_INFO (STOP_NOT_ALLOWED, HISTORIC, NO_INFO, NO_FORMALISM)
END_SERVICE
```

2.4.5. Cas particulier : l'intégration d'un script shell

Nous avons délibérément prévu l'intégration d'outils simples développés en shell dans FrameKit. Pour cela, on procède de manière particulière en invoquant le script via la commande `fk_invoke_script` (que la plat-forme voit comme un outil). Cette commande particulière s'invoque de la manière suivante :

fk_invoke_script in_dir version copyright info_script [info_itscript] [other]

in_dir	::=	chemin_unix
version	::=	string
copyright	::=	string
info_script	::=	-script file_nale
info_itscript	::=	-it_script file_name
others	::=	-p liste de paramètres

Le paramètre *in_dir* permet de spécifier dans quel répertoire on trouvera le script shell à invoquer et, s'il existe, le script à invoquer en cas d'interruption utilisateur. Les paramètres *version* et *copyright* sont des chaînes de caractères décrivant respectivement une version et un copyright à afficher lors du démarrage. *info_script* permet de spécifier le nom du script à invoquer pour exécuter le service. *info_itscript* permet de spécifier le script à exécuter si l'utilisateur interrompt l'exécution du service (pour faire apparaître cette possibilité, positionner STOP_ALLOWED). Les paramètres situés après *-p* seront transmis tels quels au script.

Remarques importantes:

- le script à exécuter doit écrire ses résultats dans stdout. Le script à invoquer en cas d'interruption ne doit envoyer aucun résultat.
- le script exécuté en cas d'interruption reçoit exactement les mêmes paramètres (utilisateurs) que le script invoqué par fk_invoke_script (les paramètres situés après *-p*).
- si aucun script d'interruption n'est précisé, que les interruptions sont autorisées et qu'un usager interrompt un service, alors aucune action spécifique n'est effectuée (mais les données traitées par l'outil peuvent être dans un état incohérent).

Considérons le script `mon_script` qui, s'il est invoqué avec la commande `-faire`, réalisera les traitements correspondant au service «mon service». La déclaration de ce script dans FrameKit est indiquée ci dessous :

```
SERVICE (TERMINAL, 'mon service')
   ACCESS INCLUSIVE
      ALL_USERS
   END_ACCESS
   BEGIN_PRECONDITION
      TRUE
   END_PRECONDITION
   EXECUTABLE ($(FK_EXECUTABLE_ROOT)'fk_invoke_script',
               'exec_dir "version 1.0" "moi, C1996" -script mon_script -p -faire',
               COMM_STD_OUT, 'INTERNAL_NAME_OF_TOOL')
   PROTOCOL (SAFE)
   QUESTION_INFO (STOP_NOT_ALLOWED, HISTORIC, NO_INFO, INPUT_FORMALISM)
END_SERVICE
```

2.5. BNF d'un «fichier de références»

Dans les fichiers de références comme dans les fichiers de services, les commentaires peuvent survenir n'importe ou. Ils commencent par **/*** et se terminent par ***/**.

qtree_reference_file	::=	**`REFERENCE_FILE`** **(**chaine[7]**)** **`BEGIN`** body_reference_file **`END`**
body_reference_file	::=	one_reference | body_reference_file one_reference
one_reference	::=	**`NEXT_ARE_SUBTREES`** | **`SERVICE`** **(**file_ref[8]**)**
chaine	::=	`'`{any_char_but_quote}`'`

Remarque importante : le mot clef **`NEXT_ARE_SUBTREES`** permet de séparer les outils dont le menu de service associé se résume à une seule entrée des outils dont ce même menu correspond à un sous-arbre plus profond et ne doit se trouver qu'une seule fois dans un fichier de référence. L'idée est d'avoir ainsi un menu plus facile à lire. Cette instruction optionnelle est utilisée par `admin_kit`, l'outil d'installation de la plate-forme.

2.6. BNF d'un «fichier de services»

qtree_service_file	::=	**`SERVICES_FILE`** **`BEGIN`** body_service_file **`END`**
body_service_file	::=	menu_definition | body_service_file menu_definition
menu_definition	::=	**`SERVICE (`** type_service[9]**`,`** chaine**`)`** service_help[10] droits[11] service_precondition service_body[12] **`END_SERVICE`**
service_help	::=	**`HELP (`**chaine[13]**`)`**
droits	::=	**`ACCESS`** access_type[14] access_list **`END_ACCESS`**
access_list	::=	one_user_group_arch {one_user_group_arch}
one_user_group_arch	::=	**`USER (`** chaine[15] **`)`** | **`GROUP (`** chaine[16] **`)`** | **`ARCHITECTURE (`** chaine[17] **`)`** | **`ALL_USERS`**

(7) C'est le nom du menu.
(8) C'est le nom absolu du fichier service
(9) Pour connaître la nature de l'entrée ainsi décrite (entrée de sous-menu, menu terminal, liste d'options, option).
(10) Prévu mais non opérationnel
(11) Identifie les utilisateurs ayant accès au service.
(12) Identifie le service.
(13) A priori, le texte de l'aide... dès que Jean-Luc aura mis cela en œuvre dans Macao.
(14) Permet d'indiquer si la liste qui suit comprend les utilisateurs/groupes autorisés (inclusive) ou exclus (exclusive).
(15) Nom de l'usager.
(16) Nom du groupe.
(17) Le nom donné à une architecture.

service_precondition	::=	**BEGIN_PRECONDITION** precondition_expression **END_PRECONDITION**
precondition_expression	::=	**NOT** precondition_expression | **(**precondition_expression**) OR (**precondition_expression **)** | **(**precondition_expression**) AND (**precondition_expression **)** | service_flag **(**identifier**)** | var_or_cst comp_operator var_or_cst | **TRUE** | **FALSE**
service_flag	::=	**NEVER_LAUNCHED** | **LAUNCHED_OK** | **LAUCHED_PB**
var_or_cst	::=	%identifier | chaine
comp_operator	::=	**=** | **/=**
service_body	::=	**REFERENCE**[18] **(**file_ref[19]**)** | body_service_file | **EXECUTABLE**[20] **(**file_ref**,** chaine[21]**,** comm_behaviour[22]**,** identifier[23]**)** **PROTOCOL (**safety**)** [24] **QUESTION_INFO (**stop[25]**,** historic[26]**,** info_sup[27]**,** form**)** | **CHECK_MARK**[28] **(**flag[29], identifier[30]**,** chaine[31]**)**
file_ref	::=	[**$(**fk_env_variable)] chaine
fk_env_variable	::=	**FK_TOOLS_ROOT**[32] | **FK_EXECUTABLE_ROOT**[33] | **FK_ADMIN_DATA**[34] | **FK_FORMALISM_ROOT**[35]

(18) Le mot clef REFERENCE implique que le menu soit de type `NON_TERMINAL` ou `LST_CHECK_MARKS`.
(19) C'est le nom du fichier de référence.
(20) Les mots clefs `EXECUTABLE` et `QUESTION_INFO` impliquent que le type du menu soit `TERMINAL`
(21) Liste des arguments du fichier exécutable.
(22) C'est le mode d'exécution du programme. Chaque mode définit la convention d'invocation de l'outil et le comportement qu'il doit avoir vis à vis des communications avec la plate-forme.
(23) Le nom interne du service. Ce nom permet de repérer un service de manière unique et indépendante de la langue de communication, ce qui permet leur repérage de manière UNIQUE. Attention, il ne faut pas qu'il y ait de doublons dans un menu.
(24) l'instruction PROTOCOL permet d'indiquer (c'est obligatoire!) si l'executable correspondant est protocolairement sûr du piont de vue de la terminaison. Les programmes écrits en Ada et en Shelle sont considérés comme protocolairement sûr.
(25) Le paramètre précisant si le service est interruptible.
(26) Le paramètre précisant si l'usager a droit à un historique (intéressant pour les services long) ou non.
(27) Le paramètre précisant si le service prend en entrée des paramètres supplémentaires (textes ou objets).
(28) Le mot clef `CHECK_MARK` implique que l'entrée de menu est une option à cocher.
(29) Pour positionner ou non l'option
(30) L'identificateur de l'option (pour la référencer dans la ligne de commande associée à l'invocation du service)
(31) La chaîne correspondant au paramètre associé à l'option. Lorsque $<option_id> sera rencontré dans les options (paramètre `EXECUTABLE`), il sera remplacé par cette chaîne de caractères.
(32) Cette variable représente le chemin absolu de la racine de l'arborescence des outils pour architecture courante de la plate-forme.
(33) Cette variable représente le chemin absolu du répertoire des exécutables de la plate-forme pour l'architecture courante (essentiellement utilisé pour les outils d'administration).
(34) Cette variable représente le chemin absolu du répertoire contenant les fichiers d'administration de la plate-forme.
(35) Cette variable représente le chemin absolu de la racine de l'arborescence dédiée à un formalisme.

comm_behaviour	::=	**COMM_STD_OUT** | **COMM_NAMMED_PIPE**
safety	::=	**SAFE** | **UNSAFE**
flag	::=	**ON** | **OFF**
stop	::=	**STOP_ALLOWED** | **STOP_NOT_ALLOWED**
historic	::=	**HISTORIC** | **NO_HISTORIC**
info_sup	::=	**INFO_TEXT** | **INFO_OBJECT** | **NO_INFO**
form	::=	**NO_FORMALISM** | **INPUT_FORMALISM**(36) | chaine(37)
type_service	::=	**TERMINAL** | **NON_TERMINAL** | **LST_CHECK_MARKS** | **CHECK_MARK**
access_type	::=	**INCLUSIVE** | **EXCLUSIVE**

2.6.1. Conventions sur les communications

Une fois lancé, un outil dialogue avec la plate-forme (au moyen de messages CAMI), pour envoyer des résultats, dialoguer avec l'usager, etc. Les conventions sur les communications correspondent aux différents paramétrages du bus logiciel entre FrameKit et l'outil. Elles sont transmises aux outils au moyen de la ligne de commande par insertion de paramètres avant ceux de l'outil selon la règle suivante :

```
nom_outil <arguments de l'environnement> <arguments de l'outil>
```

Les arguments de l'environnement sont traités pendant l'initialisation des A.P.I. Ensuite, une renumérotation des arguments de la ligne de commandes assure à l'outil des numéros d'arguments commençant à 1 (le premier argument de l'outil est l'argument `N+1` de la ligne de commande si `N` est le nombre d'arguments ajoutés par l'environnement).

Mode «autonome»

Il est vivement conseillé que chaque outil puisse être invoqué hors de la plate-forme (c'est intéressant pour automatiser les tests). Pour cela, l'exécutif de FrameKit respecte la convention suivante(38) : ***«si le premier argument de la ligne de commande d'un outil est '-s', alors l'outil est lancé hors plate-forme»***

En mode autonome, les outils s'affranchissent de l'envoi de messages `CAMI`(39). Par exemple, l'API de manipulation des dialogues dispose d'un mode dégradé assurant les services de communication sur une console (les menus sont simulés sur une interface de type `VT100`).

(36) Cela permet de signifier à la plate-forme que le formalisme résultat est du type du formalisme en entrée.
(37) C'est le nom du formalisme résultat.
(38) Le respect de cette convention est prise en charge par l'environnement de programmation (les API de la plate-forme).
(39) CAMI est un langage de communication permettant de gérer tout ce qui est protocole entre les différentes composantes de la FrameKit et l'interface utilisateur Macao.

Communications de type `COMM_STD_OUT`

L'outil utilisant cette convention dialogue via `STDOUT`. Ce mode de communication ne supporte pas les échanges interactifs (les communications vont de l'outil à la plate-forme seulement). Ce mode est dédié aux outils écrits en shell. L'environnement invoquera l'outil au moyen de la ligne de commande :

`nom_outil p_bus_log(40) -std_out autres_p_env(41) [arg_outil]`

Communications de type `COMM_NAMMED_PIPE`

L'outil utilisant cette convention dialogue au moyen de tubes nommés. L'invocateur d'outil précise en ligne de commande le nom des tubes correspondant aux canaux d'émission et de réception. L'environnement invoquera l'outil au moyen de la ligne de commande suivante :

`nom_outil [p_bus_log] -fifo_in_out name1 name2 autres_p_env [arg_outil]`

name1 correspond au canal d'entrée et ***name2*** au canal de sortie.

2.7. Gestion des caches de menus

Pour accélérer le calcul des menus associés à un utilisateur pour un formalisme donné, FrameKit construit des caches. Ces caches doivent être invalidés lorsque certaines opérations d'administration ont lieu (installation/désinstallation des outils, modification d'un groupe)(42). A chaque menu sont associés :

- un cache contenant les messages CAMI décrivant le menu de services;
- un cache décrivant les structures de données permettant au serveur de sessions de gérer les conditions de services et l'association service/outil.

Il existe une paire de caches par utilisateur, par formalisme et par architecture. Les fichiers de cache sont stockés dans l'espace de travail de l'utilisateur. Leur mise en place a permis de diviser la vitesse de connexion d'un facteur 2.

3. Références

[1] F. Kordon & J-L. Mounier, "FrameKit and the prototyping of CASE environments", 8th IEEE International Workshop on Rapid System Prototyping, Research Triangle Park Institute, IEEE comp Soc Press N° 97TB100155, pp 91-97, June 1997

[2] F.Kordon & J-L. Mounier, "FrameKit, an Ada Framework for a Fast Implementation of CASE Environments", to appear in proceedings of the ACM/SIGAda ASSET'98 symposium, July 1998

[3] F.Kordon, "tha FrameKit Home Page", `<http://www-src.lip6.fr/framekit>`

[4] J-L. Mounier, "the Macao Home page", `<http://www-src.lip6.fr/macao>`

(40) `p_bus_log` représente le paramètre permettant de spécifier le mécanisme sur lequel repose le bus logiciel pour les processus de FrameKit.

(41) Les autres arguments permettent de transmettre le nom de l'usager ainsi que l'idiome utilisé.

(42) Les outils d'administration concernés prennent en charge la destruction des caches au moment opportun. Un outil d'administration permet de les détruire manuellement.