Analyse fonctionnelle de l'environnement système

Table des matières

Introduction

Cette partie détaille l'analyse fonctionnelle de l'environnement système nécessaire au projet MARS.

L'atelier a été conçu de manière hiérarchique. Il peut ainsi être vu comme une encapsulation de fonctionnalités qui contribuent à la prise en charge du travail demandé par l'utilisateur. Dans cette partie, nous définissons tous les objets manipulés dans l'architecture, ainsi que les liens entre ces objets. Cette architecture prend en compte la séparation de l'interface utilisateur et des outils de modélisation, de validation et de génération de prototypes.

Nous présentons l'architecture générale de l'environnement système d'un atelier de spécification supportant le projet Mars.

Nous décrivons précisément les différents éléments permettant la gestion interne de l'interface utilisateur. Nous détaillons les modules de gestion des utilisateurs et des différents services qui leur sont offerts.

Nous introduisons le concept de station de travail virtuelle assurant l'indépendance entre l'interface utilisateur et les services. Cette caractéristique, observée par l'utilisateur comme une gestion de multi-fenêtrages dissimule, au niveau de l'environnement système, des mécanismes de gestion de contextes, soumis à de multiples commutations. L'atelier supporte aussi les contraintes d'un environnement réparti, en particulier, celles liées à la communication d'informations et au stockage distant de données.

Nous présentons une synthèse des différents modules décrits dans les chapitres précédents. En particulier, nous montrons que la solution proposée répond à l'ensemble des besoins exprimés dans le cahier des charges de la partie I.

1. Architecture générale

Actuellement, dans les environnements de génie logiciel, un des principes de base est que l'interface utilisateur et les programmes d'applications sont deux composantes modulaires distinctes [Coutaz, 1990, Karsenty, 1990]. Nous voulons aller plus loin qu'une simple séparation de l'interface utilisateur et des programmes d'application car les programmes d'application ne doivent pas s'occuper de la gestion de l'interface utilisateur, de l'introduction des données, ni de l'édition graphique des résultats.

En partant de la structure d'un atelier de spécification (Partie I, figure 1.2.b Atelier de spécification), nous précisons l'architecture de la structure d'accueil. Nous construisons la structure d'accueil en utilisant les composantes systèmes d'un atelier de spécification (partie I, §5 Composantes systèmes d'un atelier de spécification).

Nous avons mis en valeur des extensions systèmes qui prennent en compte la gestion d'un environnement distribué et du graphique. Pour rendre indépendants l'interface utilisateur (application d'interaction utilisateur) et les outils logiciels (programme d'application) de la plate-forme de l'atelier, nous définissons un niveau d'interface.

Les caractéristiques essentielles d'un atelier de spécification que nous proposons sont: une interface graphique unique, une localisation transparente des données et applications, un langage de description des fonctionnalités des applications ainsi qu'un langage de description des formalismes et un langage de manipulation des objets des formalismes.

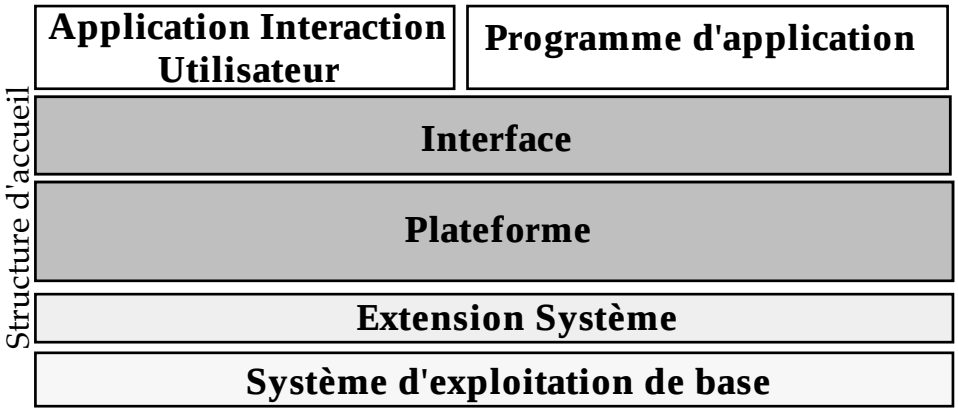


Figure 1.1: Architecture fonctionnelle générale de l'atelier

Ce chapitre présente maintenant l'architecture fonctionnelle d'un atelier de spécification en décrivant les modules associés aux cinq niveaux fonctionnels:

- niveau système d'exploitation de base,
- niveau extension système,
- niveau plate-forme,
- niveau interface,
- niveau application.

Les niveaux s'empilent depuis le niveau le plus physique (ici, celui du système d'exploitation: **système de base**) jusqu'au niveau le plus abstrait, en l'occurrence celui des applications (programme d'application et interaction utilisateur). Pour ces dernières, l'ensemble apparaît bien sûr comme une seule entité, masquant la complexité physique de l'environnement distribué.

Notre architecture repose sur un **système d'exploitation de base** (système de fichiers, système de communication). Le niveau **d'extension du système** ajoute des fonctionnalités graphique et répartition. Le **niveau plate-forme** contient un squelette d'application pour l'interface utilisateur et les gestionnaires de station de travail virtuel, des utilisateurs et des services. Au dessus, du niveau plate-forme se situe le **niveau interface** entre les différentes applications et la plate-forme. Enfin le **niveau application** contient l'éditeur de graphes, des outils d'administration et les applications de calculs. Naturellement chaque niveau intègre un sous-ensemble des objectifs définis dans la partie I.

Cette architecture n'impose aucune contrainte concernant l'implantation de l'application d'interaction utilisateur (AIU) et des programmes d'applications sur des stations de travail.

L'architecture prend en compte le fonctionnement de l'application d'interaction utilisateur en mode autonome. Le **travail en autonome** consiste à utiliser l'application d'interaction utilisateur non connecté à l'environnement système complet de l'atelier. L'utilisateur se limite alors à la mise en forme graphique des spécifications (énoncés) qui feront ultérieurement l'objet de traitements (services).

Le travail en autonome permet de concevoir et de modifier des énoncés avec un environnement supportant uniquement l'interface utilisateur. En mode autonome, l'application d'interaction utilisateur assure la construction graphique et la vérification syntaxique des spécifications de l'utilisateur. Pour cela, elle dispose d'une description locale des formalismes et stocke localement les énoncés de l'utilisateur.

Lorsque l'utilisateur est connecté à l'environnement complet, on parle de **travail connecté**. Dans ce cas, l'atelier de spécification prend en charge la cohérence des données dupliquées (formalismes, énoncés) dans l'environnement distribué.

Cette notion de travail autonome et connecté est un point de notre architecture.

2. Niveau système d'exploitation de base

Ce niveau correspond aux fonctionnalités offertes en standard par un système d'exploitation. Il assure notamment la gestion des processus, de la mémoire et des primitives de communication. Nous ne détaillerons pas les principales fonctions qui sont entre autres la création et la destruction de processus, l'ordonnancement processus, la création, la destruction hiérarchique des fichiers, le traitements des interruptions, le traitements des erreurs et la réalisation entrées sorties.

3. Niveau extension système

Pour réaliser un application d'interaction utilisateur graphique et une plateforme distribuée, le système de base n'est pas suffisant. Nous avons distinguer en ensembles d'extensions systèmes permettant de résoudre ces problèmes. Ces extensions ne sont actuellement que rarement incluses dans le noyau d'un système d'exploitation. Le niveau extension système (Figure 3.0) est composé de différents modules:

Des fonctionnalités spécifiques orientées graphique (**boite à outils de programmation, système de fenêtrage et système graphique**) permettent à l'application d'interaction utilisateur de piloter les dispositifs d'entrée et de visualisation.

Le module **système répartition** offre des services de partage d'informations et des moyens de communication.

Le module **système logistique** est composé de deux catégories de fonctionnalités. La première est composée d'outils de répartition des tâches [Foliot and Ruffin, 1989, Garlick, 1988] permettant la distribution de l'atelier lui-même et la conception applications distribuées. La deuxième catégorie, la gestion de configuration, décrit l'architecture matériel et logiciel du site informatique sur lequel l'atelier est implanté. Ces deux modules réalisent les composantes répartition et système de gestion de fichiers virtuels de la figure 5.1 (partie I Figure 5.1: Adéquation des besoins aux composante).

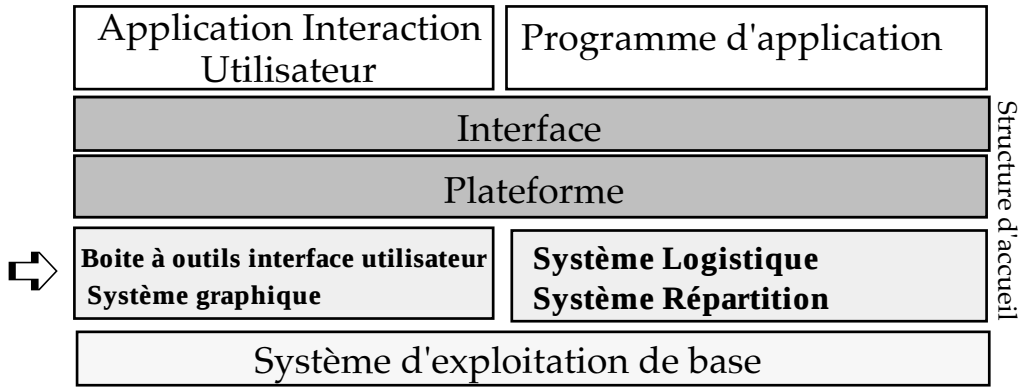


Figure 3.0: Le niveau extension système de l'architecture fonctionnelle

L'architecture fonctionnelle (Figure 3.0) est hiérarchique: chacune des boites décrites dialogue avec la boite supérieure ou inférieure (il n'y a pas de dialogue horizontal).

3.1. Extension pour l'application d'interaction utilisateur

Pour permettre l'interaction avec l'utilisateur, les extensions système nécessaires sont le **système graphique** composé des dispositifs d'entrée et de sortie, le **système de fenêtrage** contrôlant les dispositifs d'entrée et de sortie, le **gestionnaire de fenêtres** responsable du dessin des fenêtres et de leur empilement et la **boite à outils de programmation** offrant des fonctions de communication et des primitives graphiques de haut niveau.

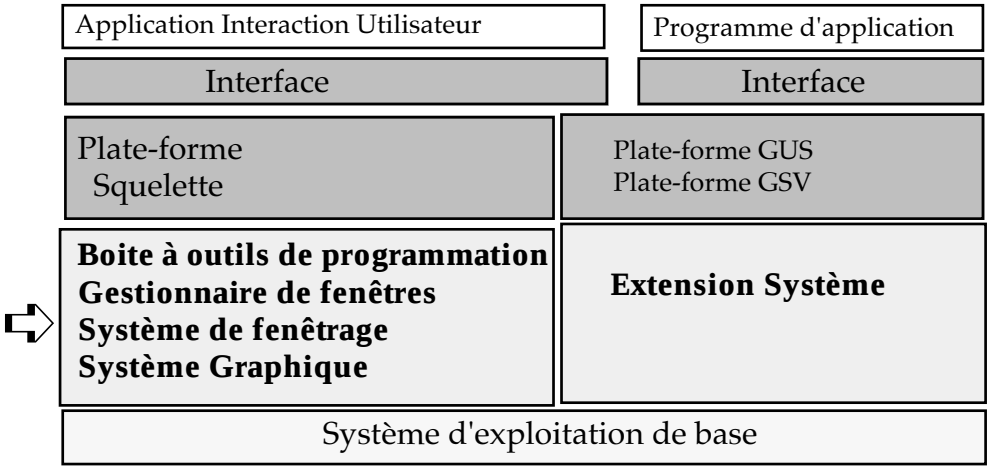


Figure 3.1: Extensions système pour l'application d'interaction utilisateur

Les extensions système sont données à titre indicatif pour le lecteur. Elles font partie des extensions nécessaires pour la constitution d'une interface utilisateur mais en aucun cas l'application d'interaction utilisateur ne fera directement appel à ces extensions, elle passera par la plate-forme squelette que nous décrirons au paragraphe §4 (Niveau plate-forme - squelettes d'interface utilisateur).

Nous nous appuyons sur des exemples concrets actuellement utilisés comme X-Window et l'interface Macintosh.

3.1.1. Système graphique

Le système graphique est le point clef de l'interface utilisateur. Il est composé de deux parties distinctes: le dispositif de sortie permettant de dessiner effectivement des objets graphiques et les dispositifs d'entrée. Nous citons dans ce paragraphe des caractéristiques qui se sont vulgarisées. Elle sont présentées en détail dans [Salmon and Slater, 1987].

Dispositifs de sortie

Les dispositifs de sortie sont actuellement l'écran et les imprimantes. Ces dispositifs supportent des modèles d'imagerie composés d'opérateurs permettant le dessin d'objets graphiques. Le dispositif de sortie est la partie visible par l'utilisateur, il permet de présenter l'aspect lexical du dialogue. Celui-ci représente l'ensemble des éléments graphiques composant le dialogue: icônes, cases à cocher, zone de saisie de texte… (modèle de **Seeheim** [Olsen, 1986]).

Dispositifs d'entrée

Les dispositifs d'entrée permettent la saisie de textes, la sélection et le dessin à l'écran. La saisie de texte est réalisée par le clavier, la sélection et le dessin se font souvent à l'aide d'une souris comportant un ou plusieurs boutons. Pour la saisie de texte, l'entrée est caractérisée par un code de caractère accompagné de modificateurs (touches enfoncées simultanément). Pour le dispositif de pointage, de nombreux périphériques autres que la souris existent (tablette, manche à balai…), ils sont normalisés sous la forme de coordonnées de pointage accompagnés de modificateurs (n° de bouton, …).

3.1.2. Système de fenêtrage

Le système de fenêtrage X-Window permet de gérer en local ou à distance les dispositifs d'entrée et de sortie de l'interface utilisateur: clavier, souris, écran. Une description plus détaillée est faite au paragraphe §6.2.2.a (Etude du système de fenêtrage X11R4). Ce système est un standard de fait dans le monde Unix.

3.1.3. Gestionnaire de fenêtres

Le gestionnaire de fenêtres ("Window Manager" [Stern, 1987]) est responsable du dessin des contours des fenêtres ainsi que de leurs superpositions, déplacements, agrandissements… C'est en particulier lui qui indiquera au niveau supérieur, lorsque l'utilisateur clique à un endroit de l'écran, à quelle fenêtre ou quelle partie de fenêtre cet endroit correspond. Le gestionnaire de fenêtres utilise le système graphique pour dessiner les contours de ses fenêtres.

Une fenêtre est un contexte pour afficher et interagir avec l'utilisateur. Des méthodes sont associés à la fenêtre: création, destruction, déplacement, manipulation, redessiner le contour des fenêtres… Ces méthodes sont pilotés par un gestionnaire de fenêtres. Le gestionnaire de fenêtres gère d'abord les superpositions et les positions des différentes fenêtres afin de conserver une représentation correcte de l'écran.

Des gestionnaires de fenêtres sont livrés en standard dans la distribution X11R4: *twm* le gestionnaire de fenêtre par défaut, *gwm* gestionnaire de fenêtre développé par BULL, *olwm* gestionnaire de fenêtres d'OPEN LOOK développé par Sun Microsystems ou *mwm* le gestionnaire de fenêtres d'OSF/Motif. Tous ces gestionnaires de fenêtres font appel à des points d'entrée de la bibliothèque Xlib de X-Window.

Sur le Macintosh, le gestionnaire de fenêtres (The Window Manager) est unique et fourni en standard avec chaque machine (dans une mémoire morte). Il fait appel aux primitives "QuickDraw" pour le dessin.

3.1.4. Boite à outils de programmation

La boite à outils représente tout ce qui est fourni au programmeur pour créer son interface utilisateur. (exemple: des points d'entrée de la bibliothèque Xlib de X-Window).

Cette boite à outils propose une normalisation du dialogue pour l'accès à différentes fonctions du système d'exploitation comme la gestion des fichiers (gestion graphique de l'interface système) et de communication et l'accès à des fonctions graphiques.

Par exemple, le dessin des graphes de l'utilisateur se fera par appels à des opérateurs (dessin de droite, de cercles, de textes…) qui accéderont au dispositif de sortie du système graphique.

La boite à outils de programmation définit l'interface de programmation (Application Program Interface). L'interface de programmation est formée de l'ensemble des noms de fonctions, de leur séquence d'appel ainsi que des structures de données associées.

Actuellement, les boites à outils sont en voie de **normalisation** (OPEN LOOK [OPEN LOOK, 1989b], OSF/Motif [OSF/Motif, 1989b]). En se normalisant, elles se figent et il devient difficile d'y inclure de nouvelles fonctionnalités. Ceci reste tout de même possible par ajouts de primitives mais ces primitives restent forcément non normalisées pendant longtemps donc peu utilisées si on veut obtenir un logiciel portable !.

Les boites à outils permettent en général des **redéfinitions**. En fait la redéfinition est assez facile lorsque la boite à outils est utilisée dans un langage orienté objets par contre cette redéfinition n'est possible dans les autres environnements que si cela a été prévu à la conception de la boite à outils sous forme de **procédures personnalisées**. La boite à outils **Aïda** [Deuin, 1987], écrite en LeLisp est de la première classe tandis que la boite à outils du Macintosh offre beaucoup de possibilités de procédures personnalisables sous forme de ressources. Il faut noter que la deuxième solution est nettement moins accessible pour le programmeur.

Les boites à outils rendent les interfaces utilisateurs plus homogènes au prix d'une difficulté de programmation (d'apprentissage à la programmation) et souvent d'une redondance de code pour le contrôle des éléments de base. Cette redondance peut être limitée par la construction de squelettes d'applications (§4 Niveau plate-forme - squelettes d'interface utilisateur) et la définition de modules de plus haut niveau prenant en charge un certain nombre d'enchaînements d'appels à la boite à outils.

3.2. Extensions non liées à l'application d'interaction utilisateur

3.2.1. Système - Répartition

Il est nécessaire de posséder des moyens de communications dans notre environnement distribué. De plus, l'exploitation de matériels hétérogènes exige une bonne portabilité de logiciels de communication.

Une bonne répartition des ressources sur différentes machines d'un système distribué est nécessaire pour une meilleure utilisation de ses ressources et un plus grand confort d'utilisation. La mobilité des ressources et des machines dans un système de communication distribué constitue donc une demande importante à considérer.

a. Communication

Dans un environnement distribué, l'absence de mémoire commune oblige à lier la synchronisation aux communications. L'effet de la synchronisation diffère de l'émetteur au récepteur d'un message [Guillemont, 1984].

Il existe quatre modes d'envoi:

- **Asynchrone**
  Lorsqu'un processus envoie un message, il est bloqué pendant un temps plus ou moins long. L'émetteur est bloqué juste le temps nécessaire au système pour prendre en compte sa demande. Cet envoi est appelé asynchrone.
- **Synchrone**
  L'émetteur est bloqué jusqu'à ce que son message soit déposé sur la porte destinataire; c'est l'envoi synchrone.
- **Rendez-vous**
  L'émetteur est bloqué jusqu'à ce que le récepteur reçoive le message. Cette méthode est appelé rendez-vous.
- **Procédure**
  La dernière solution est le blocage de l'émetteur jusqu'à la fin du traitement du message par le récepteur. C'est l'appel de procédure.

Pour le récepteur, la réception d'un message n'est effective que lorsque le processus demande explicitement cette réception. Ainsi, les arrivées de messages ne perturbent pas le déroulement du processus. Ses **messages prioritaires** interrompent l'exécution du processus et déclenchent un traitement particulier.

Dans notre environnement distribué, plusieurs entités d'applications peuvent être activées par des utilisateurs, ce qui implique l'exécution concurrente de plusieurs entités de protocoles dans différentes couches. Pour permettre l'exécution concurrente de plusieurs requêtes utilisateur et donc l'exécution concurrente de plusieurs entités de protocole, il faut que l'environnement fournisse des moyens de **communications asynchrones** entre les entités [Benkiram, 1988]. Chaque entité concurrente s'exécute alors indépendamment des autres et est activée sur une interaction entrante.

Plusieurs protocoles utilisés prévoient l'utilisation de **temporisateurs**. L'environnement doit fournir des outils de manipulation de ces temporisateurs et à l'expiration de ceux ci, permettre l'aiguillage vers des actions prédéterminées.

Nous avons défini les moyens nécessaires à la communication entre les processus du système distribué, il faut maintenant étudier les entités échangées et les mécanismes de désignation nécessaires entre émetteurs et récepteurs.

Dans notre système, comme dans les systèmes à acteurs [Rozier, 1987], tous les processus communiquent par échange de **messages**. Les entités échangées sont donc des messages. Le processus émetteur ne désigne pas directement le processus récepteur. Nous avons besoin d'une désignation homogène du récepteur. Différentes techniques de nommage dans les systèmes d'exploitation répartis sont étudiées dans [Legatheaux Martins, 1988].

Par exemple, cette désignation peut être réalisée en utilisant un nom de "porte" du site sur lequel s'exécute le processus récepteur. Cette désignation semi-directe convient aux applications statiques où les processus ne changent pas de site en cours d'exécution (ce qui est suffisant pour un atelier de spécification).

Pour assurer une communication de messages fiables, une des possibilités consiste à choisir un protocole de **transport en mode connexion** [TCP/IP, 1981], c'est à dire l'établissement d'une communication bidirectionelle entre le processus émetteur et le processus récepteur et transports des messages avec des mécanismes de contrôle d'erreurs et de contrôle de flux.

Par **contrôle de flux**, on désigne les mécanismes offerts par le système pour permettre au processus émetteur de contrôler son débit d'émission. Il se traduit dans notre cas par le ralentissement de l'émetteur et cette gestion est contrôlée par le système. Il fournit aussi une indication en temps réel sur **l'état de la connexion**; en cas de rupture de connexion, le système en avertit l'émetteur et le récepteur.

En conclusion, cette extension système doit offrir un système de nommage pour les processus, un communication par message en mode connecté avec contrôle de flux. Le dialogue par messages et l'utilisation de processus pour chacun des niveaux de l'architecture facilite l'exécution répartie des différents modules non liés à l'application d'interaction utilisateur.

b. Gestion des Fichiers Distants

Dans un environnement distribué, il est nécessaire de gérer l'accès transparent aux ressources "disques" du réseau. La Gestion des Fichiers Distants (GFD) assure un partage d'informations entre sites de manière transparente et rend uniforme l'accès aux fichiers des utilisateurs sur l'ensemble des sites supportant l'atelier. Un utilisateur connecté à l'un des systèmes du réseau local, accède à ses fichiers par la GFD, comme s'il en disposait localement. Les programmes accèdent à des fichiers par la GFD sans modification de code.

Cette gestion est traditionnellement assurée par un serveur de fichiers qui permet l'exportation des fichiers vers d'autres serveurs [Coulouris, 1989]. Par exemple, les interconnexions de systèmes Unix par l'intermédiaire de réseaux locaux à haut débit sont rendues transparentes par le développement de systèmes tels que NFS de Sun [NFS, 1985], RFS d'AT&T [Emrich, 1987], Domain d'Appolo [Leach, 1983]. A titre indicatif, une étude comparative des systèmes NFS et RFS peut être trouvée dans [Vandome, 1986].

Mais ces mécanisme de serveur de fichiers centralisés ne permettrait pas d'être tolérant aux pannes: en cas d'arrêt du serveur de fichiers, il ne serait plus possible de travailler. Il faut donc fournir dans ce cas un fonctionnement en mode dégradé par l'utilisation de serveurs de secours. Cette solution consiste à privilégier certains utilisateurs en leur donnant la possibilité d'accéder aux **serveurs de secours** en dupliquant leurs données. Aucun a priori n'est fait sur ces utilisateurs. L'administrateur système décidera de donner accès aux serveurs de secours à certains utilisateur. Lors d'un travail en mode dégradé sur le système, l'utilisateur accède aux fichiers qui sont placés sur les serveurs de secours. Les copies de ces fichiers sont transmises au serveur **principal**. Nous estimons que les copies doivent être automatiques et non à la charge de l'usager (environnement distribué transparent aux usagers). On augmente pour ces utilisateurs privilégiés leur espace de travail.

La gestion des fichiers distants ne permet que l'accès aux fichiers de manière transparente. Il est nécessaire d'ajouter des mécanismes supplémentaires dans le cas ou des usagers coopèrent pour résoudre des problèmes d'accès simultanés aux ressources (fichier) par l'utilisation de "Lock" par réseau.

3.2.2. Système - Logistique

Le niveau système - logistique est divisé en problèmes de gestion de configuration matérielle, logicielle et en outils permettant la répartition.

a. Gestion de la configuration

Les réseaux d'ordinateurs et leurs applications deviennent de plus en plus complexes à gérer. Notre objectif a été pourtant d'obtenir une **flexibilité** suffisante pour implanter des changements ou de nouveaux logiciels et matériels à la demande, comme par exemple la reconfiguration du réseau en cas de suppression ou d'ajout d'une station de travail. La gestion de configuration décrit l'architecture matérielle et logicielle des sites sur lequel s'exécute l'atelier. Des tables qui contiennent les noms des machines du site (utilisés pour le nommage des machines), la classe de chaque machine (SUN série3, Sun série4, Vax …) et d'autres critères comme la capacité mémoire et disque, ainsi que la "vitesse" de la machine (en Mips par exemple).

D'autres tables définissent l'emplacement des logiciels sur les différentes machines du site. Il faut permettre l'**hébergement d'outils**. Héberger des outils [NSE, 1990] signifie que l'on intègre des outils logiciels existants sans aucune modification, ni de l'environnement ni de l'outil, cette intégration doit pouvoir être faite par l'administrateur de l'Atelier, et parfois par l'ingénieur système. On peut donner comme exemple héberger un nouveau compilateur pour un langage (Ada) ou bien un interpréteur (LeLisp).

La configuration logicielle et matérielle de l'atelier évolue de manière dynamique; par exemple il suffit de rajouter la description d'une machine, pour qu'elle soit utilisable comme machine d'exécution de services.

Cette gestion de configuration prend en compte les problèmes de **protection** de l'atelier proprement dit. C'est à dire que l'écriture de l'atelier est faite de façon à ce qu'aucune application intégrée ne puisse rendre incohérentes les couches de logiciel de l'atelier quelque soit le comportement de ces applications.

Les techniques tables de configuration logicielle et matérielle permettent d'obtenir un atelier efficace [Bourgeois, et al., 1988] ainsi que sa capacité à s'adapter aux changements liés à l'évolution technique et logicielle et à l'augmentation du parc de machines. L'atelier reste stable quelle que soit la dynamique du projet : besoin d'outils ou besoin de machines.

b. Gestion de la répartition

Notre volonté d'exploiter le parallélisme pour augmenter l'efficacité ou la robustesse nécessite la prise en compte des critères de distribution des applications entre les sites. La répartition de charge assure la migration des applications de manière transparente, il est ainsi inutile de modifier le code des applications pour pouvoir entreprendre leur migration. Le module "gestion de répartition" optimise ainsi l'exploitation globale des ressources de l'atelier.

L'atelier fonctionnant dans un système distribué hétérogène, la gestion de répartition permet l'activation d'applications non disponibles sur une machine locale.

4. Niveau plate-forme - squelettes d'interface utilisateur

4.1. Le niveau plate-forme

Nous détaillons le niveau plate-forme qui est constitué de trois modules (Figure 4.1):

- Le squelette.
  Le squelette est un module utilisé par l'interface utilisateur pour contrôler les évènements graphiques du clic de la souris au multi-fenêtrages (boucle d'évènement, gestion des menus, des fenêtres, du dialogue et du graphisme). Il rend indépendant les applications d'interaction utilisateur de l'environnement logiciel et matériel graphique.
- Le Gestionnaire de Station de travail Virtuelle (GSV).
  La plate-forme GSV permet le transport d'informations entre l'application d'interaction utilisateur et les programmes d'application. Elle peut être assimilée à un système de fenêtrage de haut niveau.
- Le Gestionnaire des Utilisateurs et des Services (GUS).
  Le GUS gère l'organisation des données des utilisateurs, l'ensemble des formalismes, des services et des programmes d'application de l'atelier. Cette gestion est assurée dans un environnement distribué.

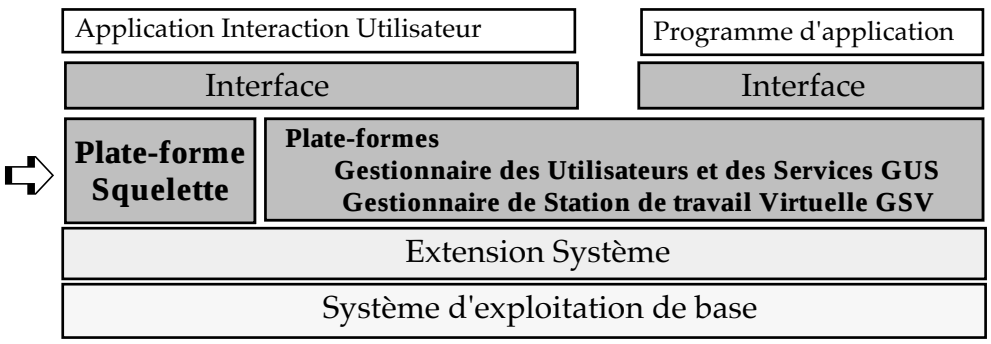


Figure 4.1: Le niveau plate-forme de l'architecture fonctionnelle.

4.2. Le niveau plate-forme - squelette

Nous avons vu précédemment que les boites à outils de programmation permettent d'effectuer des opérations graphiques de base. Or, la constitution d'applications nécessite un assemblage d'opérations de plus haut niveau.

Pour permettre de porter l'application graphique d'un environnement système à un autre, il est indispensable de définir précisément une interface externe. Cette interface externe peut être définie dans un squelette d'application. Un squelette d'application est un programme gérant les éléments de base de l'interaction utilisateur communs dans toutes les interfaces utilisateurs: menus, fenêtres... A chacune des actions de l'utilisateur, le squelette donne un feed-back graphique (exemple: le déroulement d'un menu) et appelle une procédure de l'application à la fin de l'action de l'utilisateur (exemple: l'article du menu sélectionné). Ce squelette sera la seule partie du programme d'interaction utilisateur à modifier lors du portage sur un autre système graphique.

Les squelettes d'application sont actuellement peu répandus dans l'industrie [Coutaz, 1990], nous pouvons tout de même citer MacApp [Schmucker, 1986]: un squelette développé par Apple en Pascal Objet. La littérature fait état de quelques expériences comme Grow [Barth, 1986] et ERZin [Lieberman, 1985].

Nous avons décomposé le niveau squelette en sous parties (Figure 4.2): **squelette d'application**, des **fenêtres**, des **menus**, du **dialogues** et du **graphique**. Chacune de ces parties est responsable d'un certain nombre de messages qu'elle pourra traiter localement ou bien les passer à l'application. Cette stratégie centralise les entrées et simplifie le contrôle de l'application qui ne reçoit seulement que les messages spécifiquement dirigés vers elle par l'utilisateur.

Notre architecture étant hiérarchique, la plate-forme squelette dialogue uniquement avec les les niveaux interface et extension système.

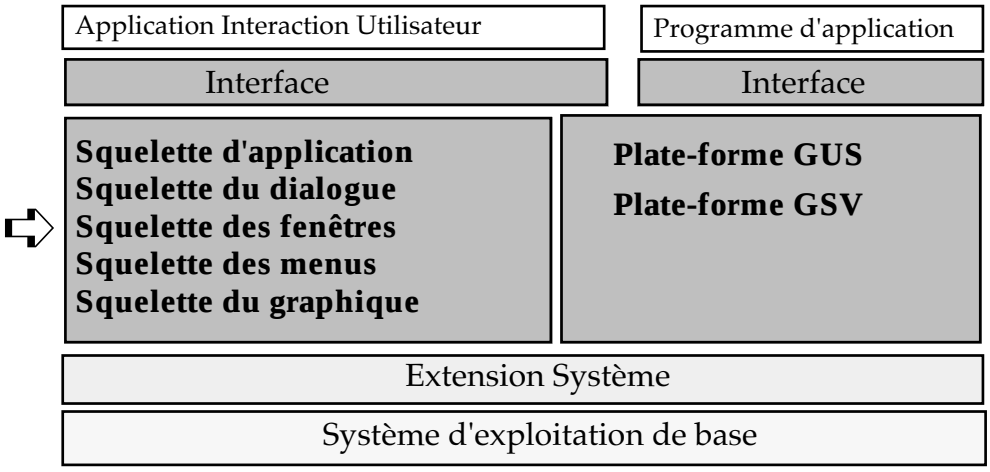


Figure 4.2: Le niveau plate-forme squelette

4.3. Squelette d'application

Un squelette d'application doit être complet: c'est à dire que l'application écrite au dessus de ce squelette ne doit faire appel qu'aux points d'entrée du squelette.

La première tâche d'un squelette d'application d'interface utilisateur est de gérer une boucle d'événements. En effet les interfaces utilisateurs travaillent toutes par boucles d'événements: le squelette attend qu'un message (événement) se présente, il le traite et attend un nouveau message, ainsi de suite jusqu'à ce que le traitement d'un message demande l'arrêt de l'application.

Les principaux messages sont les messages générés par l'utilisateur et les messages systèmes. Quand un utilisateur pointe dans une fenêtre (clic), un message est reçu par le squelette qui le traite à son niveau ou qui le transmet à l'application. Des messages systèmes comme la commutation de fenêtres, la communication entre applications sont aussi traités par le squelette.

4.4. Squelette des fenêtres

Le squelette des fenêtres s'appuie sur le gestionnaire de fenêtres. Il permet de traiter un certain nombre de primitives de bas niveau et prépare le travail pour l'application graphique. Il filtre et décode les messages envoyés par l'utilisateur pour les transmettre à la couche supérieure sous une forme standardisée.

Par exemple, lorsque l'utilisateur change la taille d'une fenêtre, le dessin de la bordure de la nouvelle fenêtre est effectué par le squelette qui envoie à l'application la nouvelle taille de la fenêtre et en cas d'agrandissement l'ordre de redessiner une partie du contenu de la fenêtre et en cas de rétrécissement, il n'envoit à l'application que la nouvelle taille.

Le squelette des fenêtres permet d'unifier les possibilités des différents gestionnaires de fenêtres (§3.1.3 Gestionnaire de fenêtres). Certains gestionnaires de fenêtres prennent en charge complètement les changements de taille de fenêtres et n'envoie un évènement que pour indiquer la nouvelle taille, d'autres considèrent que le changement de taille est du ressort de l'application et envoit un message au début du changement de taille et l'application doit donner l'ordre au gestionnaire de fenêtre de redessiner les parties de nouveau apparentes en dessous de la fenêtre lors de la diminution. (Figure 4.4.a et b).

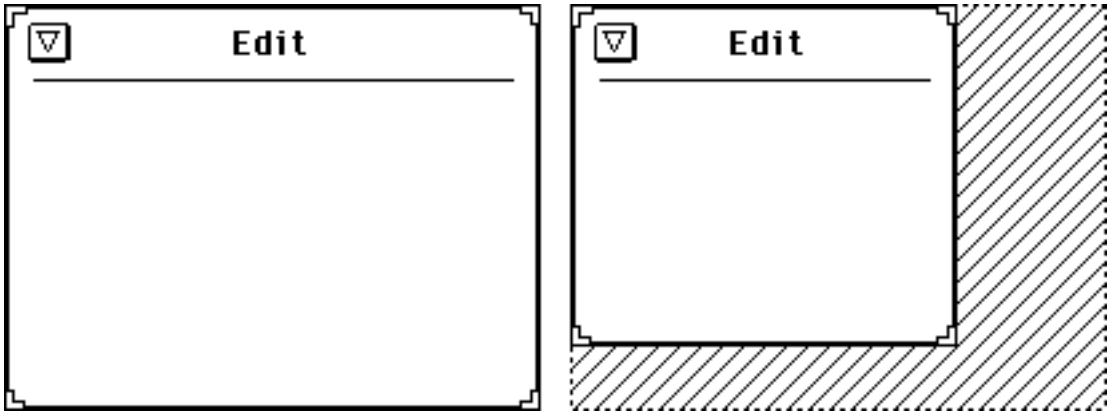


Figure 4.4.a: Diminution de fenêtre sous OPEN LOOK

Sous OPEN LOOK, l'application est prévenue du changement de taille de la fenêtre et le gestionnaire de fenêtre redessine la partie hachurée.

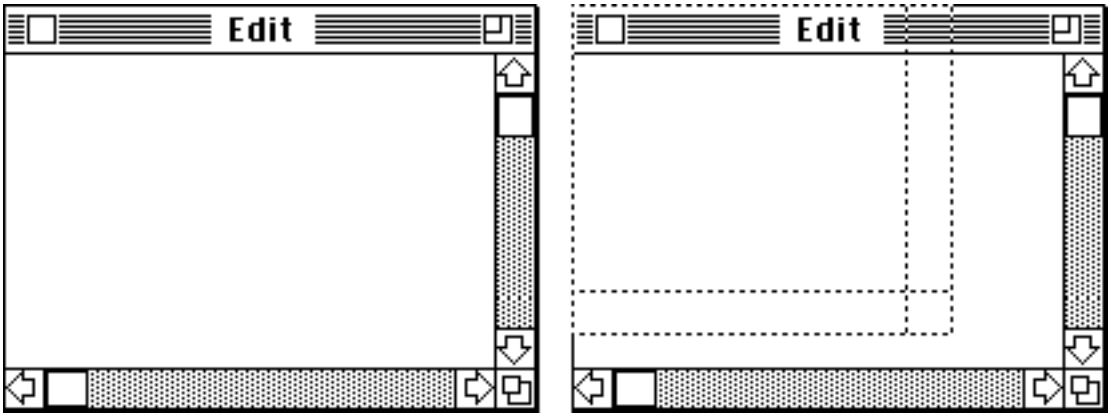
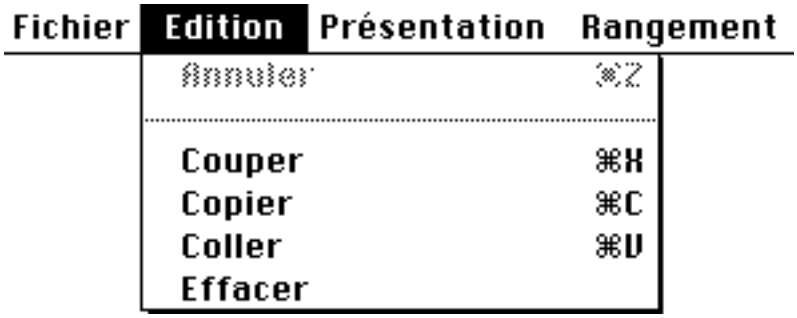


Figure 4.4.b: Diminution de fenêtre sur Macintosh

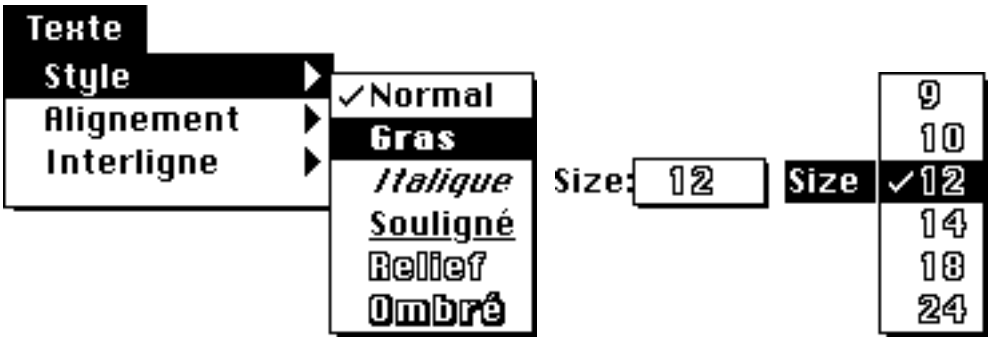
Sur le Macintosh, l'application reçoit un message lors l'utilisateur pointe dans la case d'aggrandissement (en bas à droite de la fenêtre), elle fait appel (appel de fonction) au gestionnaire de fenêtre pour que la taille de la fenêtre suive le pointeur (en pointillé sur le dessin). La fonction lui rend la nouvelle taille ainsi que la zone découverte (hachurée dans la figure 4.4.a). L'application demande au gestionnaire d'une part de fenêtre de changer réellement la taille de la fenêtre et d'autre part de redessiner la zone hachurée. Le gestionnaire de fenêtre enverra des messages de "rafraîchissement" à chacune des fenêtres des applications concernées en leur indiquant quelle zone doit être redessinée.

4.5. Squelette des menus

Le squelette menus permet de s'affranchir de l'ordre et de la position des menus. Il prend en charge l'affichage des menus et ne transmet à l'application graphique que les messages de choix faits par l'utilisateur. Le squelette menus rend transparentes, pour l'application, les différentes formes de menus à l'écran (Figure 4.5.a et b).

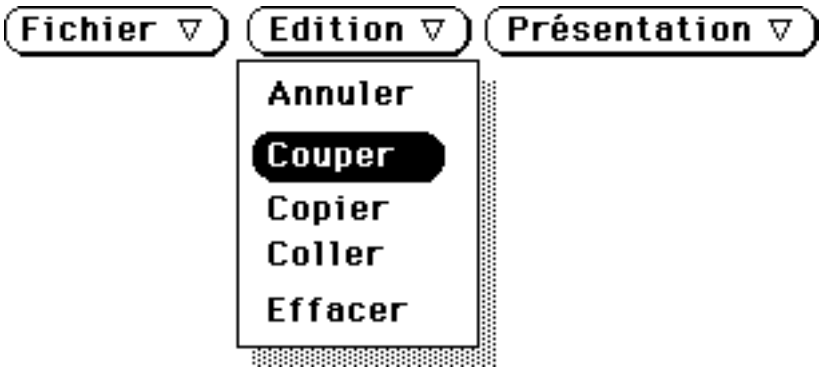


Barre de menus

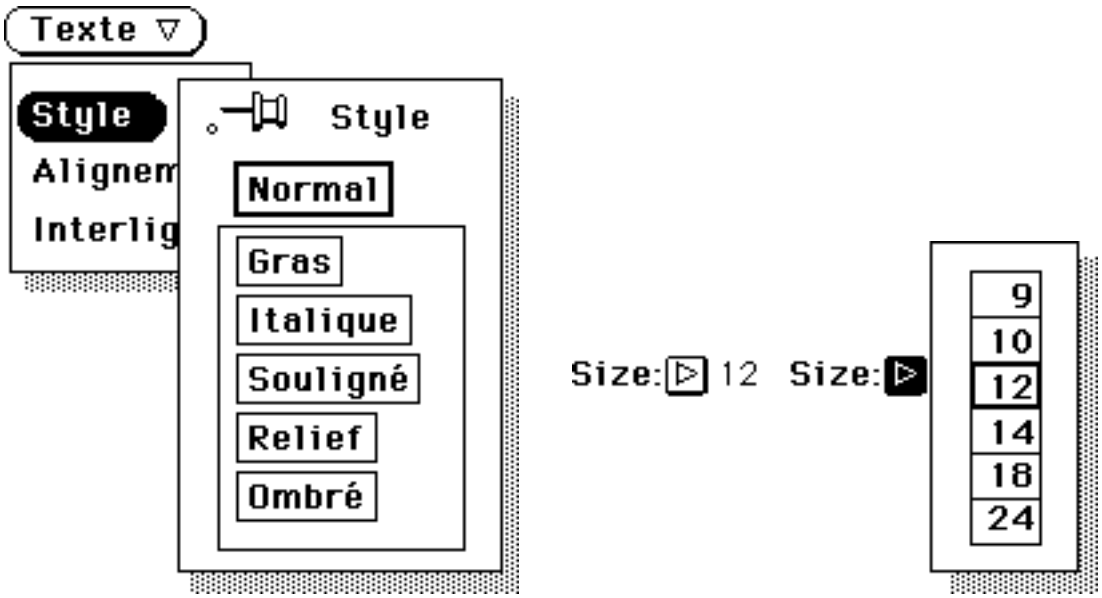


Menu hiérarchique *Pop-up Menu*

Figure 4.5.a: Exemples de menus sur Macintosh



Exemple de menus de la fenêtre de base



Menu hiérarchique avec fenêtre punaisable *Menu avec choix exclusifs*

Figure 4.5.b: Exemple de menus sous OPEN LOOK

L'application fera appel à la boite à outils (§3.1.4 Boite à outils de programmation) pour créer son menu (titre, nom des articles…). Elle transmet ce menu au squelette des menus qui sera chargé de le présenter à l'utilisateur, de le dérouler lorsque l'utilisateur pointera dessus et d'envoyer un message à l'application lorsque l'utilisateur aura choisi un article.

4.6. Squelette des dialogues

Ce squelette permet la création et la gestion des fenêtres de dialogue. Il est responsable, au moment d'une demande d'information à l'utilisateur, par l'intermédiaire d'une boîte de dialogue, de l'activation, la validation et la présentation d'options jusqu'à la terminaison de la réponse de l'utilisateur. Ce squelette du dialogue est mis en place essentiellement pour faciliter l'écriture et le portage des applications d'interaction utilisateur.

Le squelette d'un dialogue fait réagir les différents éléments des fenêtres de dialogue et uniformise la vue depuis les applications. Comme le squelette du dialogue centralise la gestion des fenêtres de dialogue, il suffit de l'informer de certaines préférences de l'utilisateur pour les appliquer à tous les dialogues. Cependant, nous avons constaté que, pour garder une ergonomie générale par rapport aux autres applications de l'environnement de travail de l'utilisateur, il ne fallait pas lui laisser tout loisir de tout transformer. Par exemple, la position des boutons "Ok" et "Annuler" doivent respecter un ordre défini dans le guide de style de l'interface utilisateur et il ne faut pas laisser à l'utilisateur la possibilité de les inverser pour une fenêtre de dialogue ou pour les fenêtres de dialogue de son application !

Il est difficile de concevoir un squelette de dialogue général car il n'existe pas de formalisme décrivant entièrement le dialogue homme-machine. Une enquête auprès de 77 spécialistes dans le domaine des interfaces utilisateurs a conclut à la difficulté de ce probleme [Molich, 1990]. De nombreuses recherches ont été faites et ont abouti à des résultats intéressants (Graffiti [Karsenty, 1987], Masaï [Masaï, 1989], Interface Builder [NextStep, 1989], Commando [Commando, 1989], HyperCard [Goodman, 1987]…) Une étude d'une dizaine de produits de recherches d'universités américaines peut être trouvé dans [Hartson, 1989]. Ces recherches ont spécifié, sous forme de langages, les séquences d'actions et enchaînements autorisés pour l'utilisateur. La plupart permettent une spécification graphique du dialogue.

Le dialogue, bien qu'essentiel, ne représente cependant pas toute l'application graphique et les générateurs d'interface s'arrêtent forcément à ce niveau. Il reste à programmer de manière "traditionnelle" le reste de la sémantique de l'application. Nous présentons cette partie dans la réalisation de l'application d'interaction utilisateur, (III §5 Réalisation de l'application d'interaction utilisateur et §7 Utilisation de l'atelier).

4.7. Squelette du graphique

Un squelette graphique est un ensemble de primitives permettant de dessiner des objets et du texte dans des fenêtres graphiques. Autant il est assez facile de définir des interfaces de dialogues génériques et homogènes, autant il est difficile de fabriquer un squelette graphique générique. En effet, de nombreux squelettes existent mais ils ne permettent pas de faire les mêmes choses. Les langages de descriptions de pages pour les imprimantes laser (Postscript [PostScript, 1986], HP, …) en sont des exemples, les langages de description de dessins (GKS [GKS, 1985], PHIGS [PHIGS, 1986], QuickDraw [APPLE, 1985], langage de X-Window [X-Window, 1990], de MS-Windows, …) en sont un autre.

Si on n'a besoin que d'un petit sous ensemble de primitives graphiques, la définition d'un langage simple permettra au moins de s'affranchir des problèmes élémentaires de coordonnées (positions, définition des écrans). C'est le cas dans le protocole X-Window où l'application manipule des opérateurs graphiques indépendamment des matériels constituant le poste de travail. Cela s'applique bien au cas des dessins de graphes où les nœuds ont des formes simples. Par contre il est impossible de bénéficier de toute la puissance d'expression de la boite à outils d'une machine pour dessiner des nœuds complexes (comme des images complexes) car le squelette graphique sera forcément réducteur pour être compatible avec la plupart des systèmes graphiques. (§3.1.1 Système graphique).

5. Niveau plate-forme - Gestion des Utilisateurs et des Services

5.1. Introduction

La plate-forme GUS gère des formalismes sur lesquels sont définis les services. Le modèle spécifié par l'utilisateur (l'énoncé) est exprimé à l'aide des instances des objets liés au formalisme.

Définition: Un formalisme

Un **formalisme** est l'expression dont la syntaxe (forme) est définie mais dont la sémantique peut l'être incomplètement (exemple SADT). Un formalisme peut être issu d'un modèle théorique (exemple : réseaux de Petri).

Définition: Un énoncé

Un **énoncé** est l'expression, suivant un formalisme, d'un problème devant être pris en charge par l'atelier de spécification. Un énoncé est un modèle dans un formalisme. Un énoncé représente les données de l'utilisateur et parfois des résultats de service.

Définition: Un service

Un **service** est un traitement sur un énoncé. Il permet à un utilisateur de référencer une action spécifique à appliquer sur l'énoncé.

Pour faciliter la représentation des formalismes, nous définissons un méta-modèle basé sur le concept de graphe structuré typé. Nous avons défini un langage de description des formalismes pour exprimer la description syntaxique et lexicale des formalismes, la partie sémantique étant laissée à la charge des programmes d'application.

Un objectif de la plate-forme GUS est d'assurer la gestion des formalismes et permettre ainsi l'ajout dynamique de nouveaux formalismes. En réalité, le GUS ne gére que le stockage des données sans en connaître la sémantique. Ainsi, pour la gestion des formalismes, le GUS stocke les différents formalismes sans en connaître la signification. Nous montrerons que cette politique de gestion des formalismes permet de faire cohabiter différent formalismes au même moment et rend l'atelier neutre vis à vis des formalismes (Objectif 15).

La plate-forme GUS associe à chaque formalisme l'ensemble des services accessible aux utilisateurs. Elle gère la notion de multi-utilisateurs. La sécurité dans un environnement distribué, multi-utilisateurs est un problème fondamental et sera donc développé dans la section §5.2 (Gestion des utilisateurs). Cette plate-forme stocke les données des utilisateurs dans leur espace de travail. La plate-forme intègre ainsi la composante de gestion des formalismes et la composante multi-utilisateurs.

La plate-forme GUS gère des données nécessaires à l'atelier et à la plate-forme GSV. La plate-forme GSV assure le transport d'informations entre l'application d'interaction utilisateur et les programmes d'applications. La plate-forme GSV rend indépendants les programmes d'application de l'interface utilisateur. La plate-forme GSV représente la composante station de travail virtuelle telle que nous l'avons décrite dans la partie I (§5.7 Station de travail virtuelle).

Les plates-formes GUS et GSV sont imbriquées (Figure 5.1), ce qui signifie que le GSV fait appel au GUS pour accéder à ses données. De plus le niveau interface fait lui aussi appel au GUS pour accéder aux données des utilisateurs.

Pour permettre une lecture plus aisée, nous présentons de nouveau l'architecture fonctionnelle sous forme d'une figure simplifiée (Figure 5.2) mettant en évidence les deux plates-formes GUS et GSV.

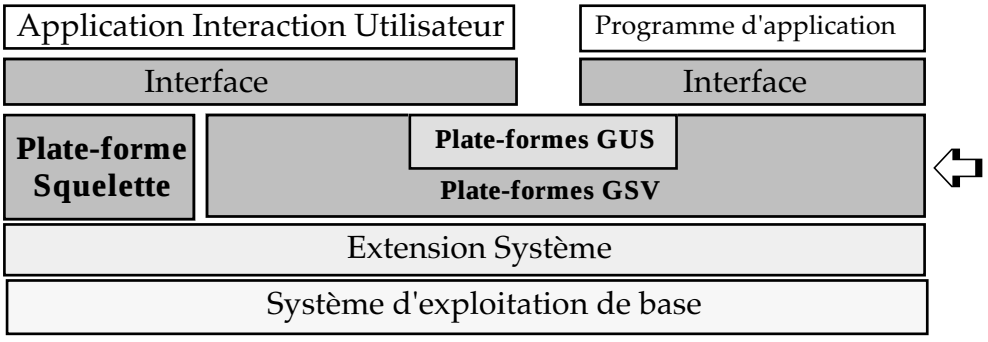


Figure 5.1: Architecture fonctionnelle simplifiée

Les utilisateurs n'ont pas besoin de mécanismes leur permettant de désigner différents types d'objets (applications, fichiers, processus, sites d'exécution…) , puisque notre but est justement de rendre transparent notre environnement distribué. La plate-forme GUS stocke les données de l'atelier en tenant compte de l'hétérogénéité et de la distribution de l'environnement et il offre le moyen d'accéder aux données indépendamment du lieu et de l'organisation du stockage.

L'importance des problèmes à surmonter tient surtout aux difficultés conceptuelles. Un des problèmes fondamentaux des systèmes distribués est l'impossibilité de capter de façon instantanée l'état global d'une application ou d'un système réparti [Helary, 1989]. Cette absence d'état global rend nécessaire la gestion rigoureuse des informations associées aux utilisateurs, aux services et aux sessions de travail.

> **Définition**: Une Session
>
> Une **session** correspond, pour l'utilisateur, à l'exécution d'un service sur un énoncé. La session est une notion dynamique tandis que le service est une notion statique.

Un des buts recherchés est d'assurer la disponibilité de l'atelier et d'exploiter les ressources et le parallélisme de l'environnement tout en étant tolérant aux pannes de machines. Les choix pris pour mettre en place la réalisation de ces objectifs prennent en compte les contraintes de tolérance aux pannes (disponibilité de l'environnement) et de temps d'accès aux données.

Cette partie s'adresse surtout à l'administrateur de l'Atelier; nous allons définir les opérations qu'il pourra effectuer sur les données de l'atelier.

Nous présentons successivement la gestion des utilisateurs, des formalismes, et des services.

5.2. Gestion des utilisateurs

Dans la partie I, nous avons distingué quatre types d'utilisateurs pour un atelier de spécifications (§I.2 Analyse des besoins des utilisateurs): le **modélisateur** (l'utilisateur final), le **théoricien** (administrateur les formalismes), le **concepteur de services** (programmeur) et l'**administrateur** de l'atelier.

Ces quatre types d'utilisateurs ont des droits et des vues différents de l'atelier:

- Les utilisateurs finals introduisent les énoncés de leurs problèmes. La caractéristique essentielle de ces utilisateurs est de ne “voir” que l’interface utilisateur, avec laquelle ils dialoguent dans le but d'utiliser les services offerts par l'atelier. Ces utilisateurs sont notamment très sensibles à la qualité, à la variété et à l'actualité des services de l'atelier. Ils sont les mieux placés pour critiquer l'ensemble des services disponibles dans le but de son amélioration.
- L’administrateur des formalismes et des services est responsable de leur gestion comme l’ajout d’un formalisme ou d’un service et de la cohérence des formalismes. L’administrateur est responsable de l'adéquation des services aux souhaits des utilisateurs.
- Les concepteurs des services développent ou modifient des applications.
- L'administrateur de l'atelier gére les données de l’atelier relatives à l’environnement système (machines et réseaux). Il configure l’atelier pour exploiter au mieux les ressources des sites et suivre leur évolution. L’administrateur est chargé de surveiller**,** de mesurer et d'ajuster le fonctionnement de l'atelier afin de donner satisfaction aux utilisateurs. Il travaille en relation étroite avec l'ingénieur système et réseau.

Un environnement logiciel doit être capable, aujourd’hui, de supporter plusieurs utilisateurs simultanés travaillant sur différents projets. Ce besoin a été identifié dans la partie I (Objectif 5). Nous précisons les politiques d’accès (accès aux données personnelles, aux formalismes et aux services) et des droits liés à chaque type d’utilisateur.

Les données de l'atelier se répartissent en deux catégories: données administratives, données personnelles.

> **Définition**: Données administratives, Données personnelles
>
> Les **données administratives** de l’atelier sont les données internes de l'atelier (les services, les caractéristiques des utilisateurs, les catégories de formalismes).
>
> Les **données personnelles** sont les données externes provenant des énoncés des utilisateurs et des résultats obtenus des exécutions des services.

Ces différentes données coexistantes ont des règles d'utilisation et de fonctionnement différentes. A chaque utilisateur est associé deux contextes: un **contexte statique** lié aux données administratives et un **contexte dynamique** liée aux données personnelles.

Les problèmes à résoudre proviennent essentiellement de l’environnement distribué et de l’utilisation en mode autonome du poste de travail.

L'espace de travail

L'espace de travail des utilisateurs, dans l'environnement distribué, est accessible par le système de gestion des fichiers distants (§3.2.1.b Gestion des fichiers distants) dans lequel certains utilisateurs privilégiés possèdent un espace de travail de secours en cas de panne du site stockant l'espace de travail général des utilisateurs. Le GUS connaît les utilisateurs privilégiés et conserve les adresses de leurs espaces de travail.

La gestion multi-utilisateurs entraîne pour chaque utilisateur un **archivage** de ses **énoncés** et de ses **résultats**. La plate-forme GUS associe dans le contexte statique des utilisateurs l'adresse de stockage de leurs données. Cet emplacement est lié au système de fichiers distants qui rend uniforme l'accès aux fichiers des utilisateurs sur l'ensemble des sites supportant l'atelier (§3.2.1.b. Gestion des fichiers distants). Nous présentons dans le paragraphe "gestion des services", la technique qui permet à la plate-forme GUS d'associer les énoncés et leurs résultats.

Cohérence des énoncés

Dans notre atelier, l'utilisateur travaille sur ses énoncés à partir de n'importe quelle poste de travail. Les résultats générés par les services sont sauvegardés par la plate-forme GUS qui assure ainsi leur accessibilité dans l'environnement distribué. Les énoncés sont sauvegardés de deux manières, soit localement par appel au système d'exploitation, soit dans un espace commun aux utilisateurs gérés par le GUS. Après une sauvegarde locale et lors d'une reconnexion à la plate-forme, le GUS est responsable de la cohérence des résultats associés à cet énoncé. Ce problème est résolu par un **estampillage des énoncés** par la date de dernière modification de l'énoncé par l'application d'interaction utilisateur. Cette estampille ne pose pas de problème de synchronisation des horloges des différentes machines; pour retrouver l'énoncé, le GUS utilise la date donnée par l'application d'interaction utilisateur. Un problème reste non résolu: celui où un même utilisateur modifierait un même énoncé sur deux postes de travail autonomes à la même seconde (date de chacune des stations). La probabilité que ce problème survienne est tout de même négligeable.

Sessions en cours

Un utilisateur peut demander un parallélisme effectif de plusieurs exécutions de services sur des énoncés différents. Un utilisateur peut mettre fin à une connexion en **laissant des services** ou sessions **en cours d'exécution** sur ses énoncés (Objectif 12). Nous sommes ramenés au problème de gestion d'états globaux pour chaque utilisateur dans un environnement distribué. La plate-forme conserve ce contexte dynamique et le restitue lors d'une prochaine connexion. L'information est l'adresse de transport (nom du site et de la porte) d'une partie du GSV qui est présent.

Unicité de connexion

La structure d'accueil est multi-utilisateurs puisqu'elle gére simultanément plusieurs connexions. Tous les services sont accessibles à partir de n'importe quel poste de travail puisque la structure d'accueil rend transparent l'environnement distribué sous-jacent. Il nous semble inutile de gérer des **connexions multiples** d'un même utilisateur à partir de différents postes de travail. En effet cela peut générer des incohérences dans ses données et l'utilisation de plusieurs postes de travail par un utilisateur n'apporte aucune fonctionnalité supplémentaire. Pour refuser une double connexion, la plate-forme GUS garde, dans le contexte dynamique de l'utilisateur, l'indication de connexion.

La sécurité

Dans un environnement multi-utilisateurs, tout utilisateur doit être **identifié** de manière distincte pour assurer la sécurité d'accès aux données. Au niveau de l'atelier, un utilisateur est identifié par son nom. Ce nom est associé à un mot de passe au niveau de l'atelier si l'on veut rendre encore plus "sécuritaire" l'accès à l'atelier. La sécurité des systèmes informatiques a fait l'objet des travaux de nombreuses organisations qui essayent de définir les fonctions de la sécurité et d'élaborer des normes .

Un des ouvrages les plus connus est "l'orange book" [Orange Book, 1985]. Ce livre définit des classes et des niveaux de sécurité, ainsi que les spécifications requises des systèmes informatiques pour s'insérer dans ce classement. Nous nous intéressons à la classe C qui offre une protection discrétionnaire. En effet, ce sont les systèmes dans lesquels le mode de protection est laissé à la discrétion de l'utilisateur. Cette classe C permet l'authentification des utilisateurs dans des systèmes d'exploitation multi-utilisateurs. Chaque classe est divisée en sous-classes. La classe C2 impose des conditions plus restrictives sur le contrôle d'accès et les procédures d'identification, ainsi que l'audit des violations d'accès. Elle requiert surtout une séparation entre les programmes utilisateurs et le noyau du système d'exploitation. Cette classe est considérée comme convenable pour la plupart des applications commerciales (SunOS système V release 4).

Le niveau de sécurité de l'atelier dépend étroitement des systèmes d'exploitation de l'environnement puisqu'il repose sur les systèmes d'exploitation des machines du site informatique. La sécurité est d'autant plus importante que l'application d'interaction utilisateur s'exécute sur postes de travail qui ne sont pas susceptibles d'identifier l'utilisateur [Berger, 1990]. Nous utilisons ici la notion d'administration système et réseau. L'utilisateur doit donc avoir **un "compte"** sur les machines, et ses données stockées dans son espace de travail sur les machines. La sécurité est donc reportée au niveau du Système - Logistique et répartition. Les services sont exécutés dans le contexte système de l'utilisateur. Un utilisateur est donc reconnu par son nom et par le lien entre son nom (atelier) et son nom au niveau du système d'exploitation.

Les données administratives maintenues au niveau de la plate-forme GUS, associées à tout utilisateur sont les données d'identification (nom,…). Ces données sont étroitement liées aux systèmes d'exploitation de l'environnement.

Coopération entre utilisateurs

Dans un environnement multi-utilisateurs, il faut envisager la coopération entre usagers. Cette interaction entre utilisateurs peut être réalisée par des opérations d'importation et d'exportation d'énoncés. L'importation d'énoncés est liée au problème de hiérarchie et au fait qu'un énoncé peut comporter d'autres énoncés. Ce problème complexe est seulement mis en évidence dans ce travail, il fera l'objet de nos futures recherches. Néanmoins notre architecture permettra d'intégrer ces nouvelles fonctionnalités.

5.3. Gestion des formalismes

L'étude des besoins des utilisateurs dans la première partie, nous a montré qu'un atelier de spécification ne doit être couplé à aucun formalisme particulier. Il faut donc, qu'au même moment différents formalismes cohabitent (Objectif 2). En particulier, pour rendre la structure d'accueil indépendante des formalismes utilisés, nous définissons un **méta-modèle** pour permettre la représentation de formalisme, ainsi qu'un **langage de description** des formalismes. Dans cette partie, nous montrons comment notre politique de gestion des formalismes vise à assurer à l'atelier la propriété de neutralité (Objectif 15).

Nous avons localisé la gestion des formalismes à un niveau spécifique de l'architecture. La plate-forme GUS est responsable de cette gestion.

5.3.1. Le Langage de Description des Formalismes (LDF)

De nombreuses méthodes de modélisation de systèmes sont fortement associées au concept de graphe (le **méta-modèle** que nous utilisons). De plus ces méthodes sont souvent associées à des représentations graphiques qui utilisent la notion de graphe (SADT, réseaux de Petri, STP, RPAS, représentation graphique d'ESTELLE…). Ces formalismes constitent pour le moment la bases de toutes les applications d'un atelier de spécification. Nous avons donc défini un Langage de Description des Formalimes (LDF). Ainsi chaque formalisme à base de graphe est exprimé dans le langage LDF.

Le langage LDF comporte une partie de description **syntaxique** et une partie **lexicale** du formalisme; la partie **sémantique** étant laissée à la charge des services.

Le langage LDF doit aussi prendre en compte les problèmes de représentation graphique uniquement pour l'application d'interaction utilisateur; c'est la partie **esthétique** du formalismes. En effet, dans notre architecture, l'application d'interaction utilisateur est la seule application à gérer du graphisme. Elle permet de saisir syntaxiquement les énoncés sans faire appel aux programmes d'application.

Nous introduisons la description des formalismes à l'aide d'une terminologie "objet" regroupant le vocabulaire suivant: classe, attribut, instance, type, méthode, client et héritage [Stroustrup, 1987]. Rapellons que la notion de **classe** regroupant des objets de type similaires permet de décrire ces objets. Les champs des objets sont des **attributs** dans une classe. Une classe, définie par un nom, décrit la structure d'un ensemble d'objets. Tous ces objets sont des **instances** de la classe. Chaque objet est une instance d'une classe, et le nom de cette classe représente le **type** de l'objet. Une classe doit permettre de décrire l'implémentation d'un type de donnée abstrait. Il faut donc ajouter à la définition d'une classe les opérations que l'on peut effectuer sur les objets. Ces opération s'appelle des **méthodes**. Si dans la définition d'une classe, il est fait référence à une autre classe, alors cette classe est appelée **cliente** de l'autre. Si une classe est donc construite par extension d'une autre classe, on parle d'**héritage**.

Pour décrire commodément tout formalisme basé sur les graphes, le langage LDF contient quatre classes pré-définies (la classe nœud, la classe connecteur, la classe information et la classe forme graphique):

- la classe nœud
  La classe nœud décrit les nœuds du graphes. Elle a un attribut qui définit sa forme graphique.
- la classe connecteur
  Pour restreindre les connexions d'objets à objets, la classe connecteur a un attribut complexe (**l'ensemble des liens possibles**) composé de couples (nœud de départ, nœud d'arrivée). Cette classe est donc une classe retardée car l'attribut complexe ne peut être défini qu'au niveau des sous-classes correspondantes. La classe connecteur possède un attribut **connexion** qui décrit la connexion entre des objets issus de la classe nœud.

- la classe information
  La classe information permet d'associer des données valables pour tout le formalisme. Cette classe a obligatoirement deux attributs par défaut, un nom et une forme graphique commune à tous les formalismes. Par formalisme, il n'y a qu'une seule classe information. Il est important de signaler que la classe information ne peut avoir qu'une seule instance par énoncé puisque cet objet donne des informations sur tout l'énoncé.
- la classe forme graphique
  La classe forme graphique décrit la forme des objets d'un formalisme, leur taille et certaines méthodes de dessins associés. Cette classe est utilisée pour la représentation des données à l'utilisateur. Cette classe n'est donc employée que par l'application d'interaction utilisateur. L'application d'interaction utilisateur possède une bibliothèque d'objet de forme graphique. Le langage LDF offre la possibilité de référencer les objets de cette classe. L'utilisation de cette possibilité particulière permet de rajouter de nouvelles descriptions graphiques de manière externe au langage LDF.

Les classes des formalimes sont construites par héritage des classes pré-définies. Le langage LDF permet donc la définition des classes d'objets qui composent les formalismes. Les classes ainsi créées sont complétées en leurs associants des attributs typés. Plusieurs types d'attributs sont définis: le type entier, le type chaîne de caractères... Dans un formalisme un attribut peut représenter une propriété de l'objet dans le modèle ou un moyen de désigner un objet (propriété de nomination).

Définition: Le langage LDF

Le langage LDF comporte une partie de description syntaxique et une partie lexicale du formalisme. La partie sémantique est laissée à la charge des services.

Nous avons défini un Langage de Description des Enoncés LDE dans la partie plate-forme GSV (§6 Niveau plate-forme - Gestion de Station de travail Virtuelle). Ce langage LDE décrit les instances des classes de tout formalisme, il est lié à la définition même du langage LDF. Un énoncé basé sur un formalisme est donc composé des objets (instanciation de ces classes) du formalisme.

a. Le langage LDF

Pour créer une nouvelle classe, il faut donner le nom de la classe (type de l'objet du modèle), le nom de sa classe prédéfinie (NOEUD ,CONNECTEUR et INFORMATION) et enfin le nom de sa classe graphique associée. Et pour chaque nouvelle classe, il faut donner la liste des attributs qui sont décrits de la manière suivante: nom de l'attribut, son type, la valeur par défaut puis la propriété de cette attribut. Cette propriété indique si cet attribut correspond à une propriété de l'objet ou permet de le désigner (par le nom).

L'expérience nous a montré qu'il était difficile de concevoir un formalisme d'un seul coup. Il peut y avoir évolution d'un formalisme lorsque l'administrateur du formalisme se rend compte que tout n'a pas été correctement décrit ou que des modifications doivent être apportées. Pour éviter qu'il ne faille créer un nouveau formalisme de nom différent, nous avons ajouté un **numéro de version** au nom du formalisme.

NOM: *identificateur*
TYPE_ATTRIBUT: entier, chaîne, énuméré, …
PROPRIETE : PROPRIETE,NOMINATION
TYPE_CLASSE : CONNECTEUR,NŒUD,INFORMATION
CLASSE_FORME: un nom de la forme dans la classe FORME_GRAPHIQUE

NomFormalisme(NOM, version)

L'utilisation d'un numéro de version associé à un formalisme est expliqué dans la partie "gestion dynamique des formalismes" par la plate-forme GUS.

CreerClasse(TYPE_CLASSE,NomDeLaClasse ,CLASSE_FORME**)**

Dans le cas d'une classe de type INFORMATION, les paramètres nom et forme ne sont pas utiles. Une classe INFORMATION a obligatoirement un nom et une forme par défaut commun à tous les formalismes.

CreerAttribut(NomAttribut,**TYPE_ATTRIBUT**,ValeurParDéfaut,**PROPRIETE)**

Dans la création d'un attribut, on ne fait pas référence à la classe à laquelle il appartient. L'appartenance est établie automatiquement par la définition de la classe englobante.

Si l'attribut est de type énuméré, le champ valeur par défaut comporte l'ensemble des éléments du type suivit de la valeur par défaut. ex: (AAA\BBB\CCC):AAA.

Connexion(NomDeLaClasse,NomDeLaClasseDepart,NomDeLaClasseArrivé**)**

L'utilisation de Connexion n'est valable que pour un NomDeLaClasse qui est un connecteur.

Le langage LDF ne concerne pas la sémantique d'un formalisme, ainsi il est possible d'exprimer n'importe quel formalisme entièrement textuel. Dans ce cas, il ne sera pas possible lors de l'introduction de cet énoncé de vérifier la syntaxe en temps réel.

Pour expliquer l'écriture d'un formalisme dans le langage LDF, nous allons prendre comme exemple de formalisme les "réseaux de Petri".

b. Exemple de description d'un formalisme: les réseaux de Petri

Les réseaux de Petri [Brams, 1983] sont utilisés afin de modéliser le comportement dynamique de systèmes discrets. Ils sont composés de deux types d'objets: les places et les transitions. L'ensemble des places représente l'état du système et l'ensemble des transitions les événements dont l'occurrence provoque la modification du système. La figure 5.3.a présente une description graphique du formalisme Réseaux de Petri. Dans l'exemple sur le formalisme Réseau de Petri, nous n'avons pas fait apparaître l'objet information pour simplifier l'explication.

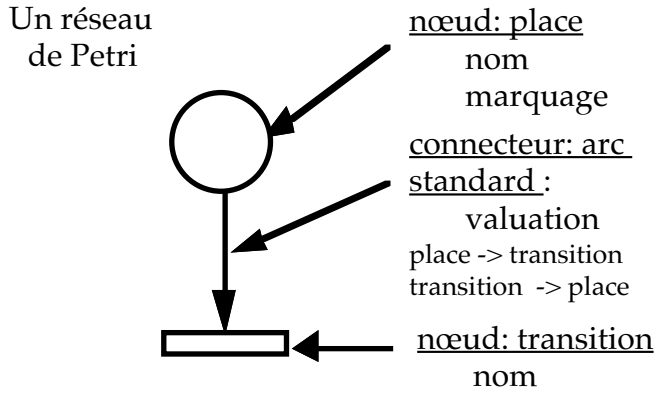


Figure 5.3.a : Exemple du formalisme réseau de Petri

Les classes regroupent des objets de comportement similaires : "les places", les "transitions" et les "arcs". Le langage LDF permet de décrire ces différentes **classes**. Ces classes héritent obligatoirement d'une des deux classes prédéfinies (nœud ou connecteur). La classe "place" a pour **attributs**: nom , marquage. Ces attributs sont **typés** : le nom est un texte, le marquage un entier. La classe "arc standard" a pour attribut, valuation de type entier. Dans le formalisme des réseaux de Petri, un arc ne peut aller que d'une place vers une transition ou que d'une transition vers une place. Il faut donc donner des valeurs initiales aux couples départ-arrivée de l'ensemble des liens possibles. La bibliothèque de forme graphique comporte la description graphique d'un rectangle, d'un rond et d'un arc.

Nous présentons ci-dessous, un schéma (Figure 5.3.b) qui décrit les liens entre les classes pour le formalisme réseaux de Petri.

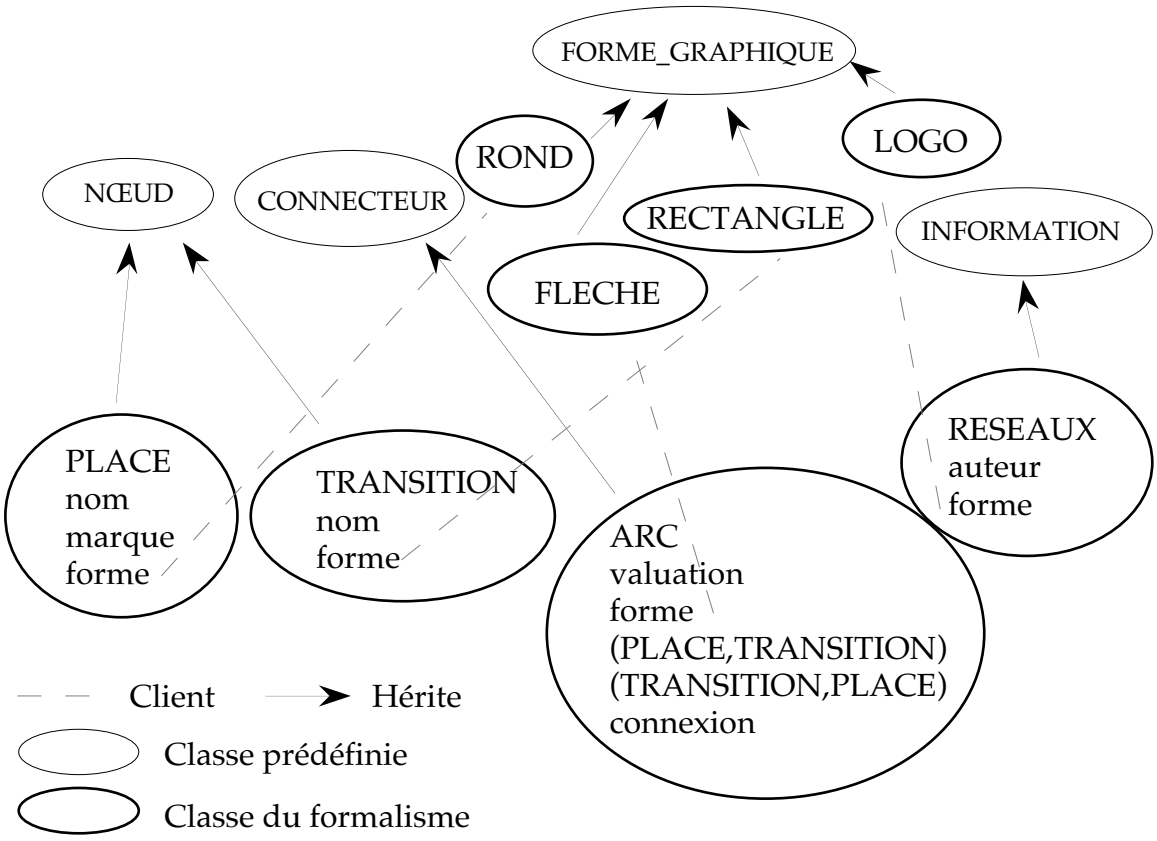


Figure 5.3.b : Les classes du formalisme réseaux de Petri

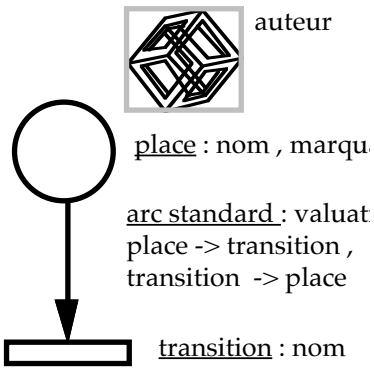
Nous définissons maintenant le formalisme Réseaux de Petri à l'aide du langage LDF. Pour prendre un exemple complet, nous définissons la classe `INFORMATION` comme un objet de forme graphique "spéciale" et comme attribut le nom de l'auteur. Ainsi, dans un énoncé basé sur cette description en LDF, de ce formalisme, l'utilisateur pourra mettre son nom comme valeur pour cet attribut.

```
NomFormalisme(Réseau de Petri,1)
CreerClasse(INFORMATION,,)
CreerAttribut(Auteur,chaîne,"",PROPRIETE)

CreerClasse(NŒUD,place,rond)
CreerAttribut(nom,chaîne,"",NOMINATION)
CreerAttribut(marquage,entier,0,PROPRIETE)

CreerClasse(CONNECTEUR,arc,fléche)
CreerAttribut(valuation,entier,1,PROPRIETE)
Connexion(arc,place,transition)
Connexion(arc,transition,place)

CreerClasse(NŒUD,transition,rectangle)
CreerAttribut(nom,CHAINE,"",NOMINATION)
```



Réseau de Petri

c. Exemple de description d'un formalisme: les réseaux locaux

Pour montrer la pluralité des représentations de formalismes au moyen du langage LDF, nous présentons succintement la description du formalisme "réseaux locaux" décrivant un réseau local simplifié. Ce formalisme est utilisé dans la partie III pour décrire l'architecture matérielle que nous avons choisie pour réaliser l'atelier AMI.

Dans notre exemple de réseau local, les différentes machines sont représentées par des nœuds (Macintosh II, Macintosh Classic, Station Unix) ayant des formes graphiques particulières et des attributs spécifiant le nom de la machine et son type. Nous définissons deux types de réseaux physiques, représentés par des connecteurs, qui relient les machines entre elles: le réseau Ethernet et le réseau LocalTalk. Ces réseaux sont interconnectés par des passerelles représentées par des nœuds dans notre formalisme. Les attributs associés aux connecteurs définissent les protocoles utilisés au niveau de la couche réseau du modèle OSI. Dans notre réseau, il faut représenter les connexions par ligne série entre station Unix et Macintosh.

Nous présentons le formalisme "réseaux locaux" ainsi que la bibliothèque de formes graphiques associées.

```
NomFormalisme(Réseau local,1)
CreerClasse(INFORMATION,,)
CreerAttribut(Localisation,chaîne,"",PROPRIETE)

CreerClasse(NŒUD,Macintosh II,macII)
CreerAttribut(nom,chaîne,"",NOMINATION)

CreerClasse(NŒUD,Macintosh Classic,mac)
CreerAttribut(nom,chaîne,"",NOMINATION)

CreerClasse(NŒUD,Station Unix,unix)
CreerAttribut(nom,chaîne,"",NOMINATION)
CreerAttribut(type,énuméré,(Sun3\Sun4\Sun386i):Sun4,PROPRIETE)

CreerClasse(CONNECTEUR,Ethernet,coaxial)
CreerAttribut(protocole,énuméré,
  (TCP/IP\EtherTalk\TCP/IP+EtherTalk):TCP/IP,PROPRIETE)

CreerClasse(CONNECTEUR,LocalTalk,paireBlindée)
CreerAttribut(protocole,chaîne,"AppleTalk",PROPRIETE)

CreerClasse(CONNECTEUR,LigneSérie,doublePaire)

CreerClasse(NŒUD,passerelle,boite)

Connexion(LocalTalk,Macintosh II,Macintosh II)
Connexion(LocalTalk,Macintosh Classic,Macintosh Classic)
Connexion(LocalTalk,Macintosh II,Macintosh Classic)
Connexion(LocalTalk,Macintosh Classic,Macintosh II)

Connexion(LocalTalk,Macintosh Classic,passerelle)
Connexion(LocalTalk,passerelle,Macintosh Classic)
Connexion(LocalTalk,Macintosh II,passerelle)
Connexion(LocalTalk,passerelle,Macintosh II)

Connexion(Ethernet,Station Unix,Macintosh II)
Connexion(Ethernet,Macintosh II,Station Unix)
Connexion(Ethernet,Station Unix,passerelle)
Connexion(Ethernet,passerelle,Station Unix)

Connexion(LigneSérie,Station Unix,Macintosh II)
Connexion(LigneSérie,Station Unix,Macintosh Classic)
Connexion(LigneSérie,Macintosh II,Station Unix)
Connexion(LigneSérie,Macintosh Classic,Station Unix)
```

L'exemple de réseau local (figure 5.3.c) présente graphiquement un réseau de Macintosh et de stations Unix interconnectés au réseau LocalTalk par une passerelle.

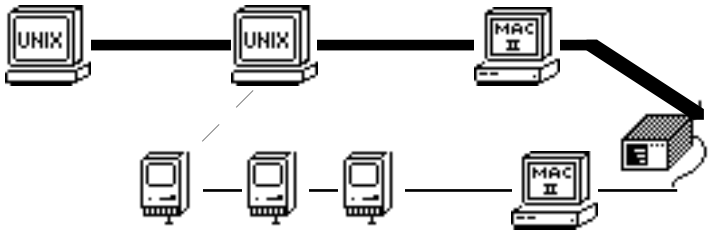


Figure 5.3.c: Un réseau local

5.3.2. La gestion dynamique des formalismes

Nous allons montrer que le GUS permet l'ajout dynamique de formalismes sans interrompre le fonctionnement de l'atelier.

Nous avons voulu que notre atelier de spécification soit ouvert au niveau de la théorie (Objectif 2), par intégration dynamiquement de nouveaux formalismes. Cette intégration est à la charge de l'administrateur des formalismes.

Par l'utilisation du langage LDF, les formalismes sont définis de manière externe à la plate-forme GUS. La plate-forme GUS ne fournit que les fonctions de stockage, de recherche des descriptions LDF des formalismes. La plate-forme GUS gère une liste de tous les formalismes connus de l'atelier, mais en aucun cas elle n'est capable d'analyser le langage LDF. L'ajout de formalisme relève donc d'une application particulière, une application d'administration des formalismes.

La description d'un formalisme définit le vocabulaire d'entrée du modèle sous-jacent. Il est du ressort de l'administrateur des formalismes de vérifier la cohérence des vocabulaires des différents formalismes. Par exemple une "place" en réseaux de Petri ordinaire doit aussi s'appeler "place" en réseaux de Petri colorés. Il faut noter que ces problèmes de vocabulaire ne sont à gérer qu'au sein d'une même catégorie de formalismes. La "place" en réseau de Petri peut s'appeler "sommet" pour la théorie des graphes.

Les vérifications syntaxiques et sémantiques lors de l'ajout de nouveaux formalismes décrits en LDF, sont réalisées par d'applications d'administration des formalismes.

La gestion des formalismes par la plate-forme GUS permet d'ajouter dynamiquement de nouveaux formalismes sans entraver le fonctionnment de l'atelier.

5.3.3. Version multi-linguistiques d'un formalisme

Nous avons mis en évidence dans la partie I qu'un atelier de spécification doit prendre en compte la notion de **multi-linguismes** (Objectif 8). Il est donc nécessaire de prendre en compte les différentes langues des utilisateurs au niveau de la gestion des formalismes. Nous étudierons dans la partie GSV, l'influence du multi-linguismes, sur la gestion des services, des applications et des interactions avec l'utilisateur.

La plate-forme GUS doit gérer l'ensemble des formalismes par rapport aux langues. La solution que nous avons retenue pour gérer les formalismes dans plusieurs langues est de "posséder plusieurs fois" la description en LDF du même formalisme dans plusieurs langues. Cette contrainte nous oblige donc à compléter le langage LDF pour associer au nom du formalisme, la langue dans laquel il est écrit. De plus pour associer des formalismes identiques mais de langues différentes, nous utilisons un **nom interne** pour désigner les formalismes. Nous complétons le langage LDF pour prendre en compte ces deux nouvelles notions:

```
LANGUE:français, anglais, allemand, espagnol…
NomFormalisme(NOM, version, LANGUE, nomInterne)
```

Puisque la plate-forme GUS est indépendante du langage LDF (elle ne gère que des listes de formalismes), il est obligatoire de lui fournir des informations complémentaires lors de l'ajout d'un nouveau formalisme: nom interne et langue utilisée.

Pour un formalisme en plusieurs langues, les seules différences portent sur le nom du formalisme et sur les noms des classes et des attributs. Lors de l'ajout d'un formalisme dans une autre langue, on précise à la plate-forme GUS les associations de noms dans les différentes langues au moyen d'un **dictionnaire de traduction**. Nous montrerons lors de la description de la plate-forme GSV, les besoins d'un dictionnaire de traduction, nous avons simplement abordé ici, la possibilité de créer ce dictionnaire.

Prenons l'exemple du formalisme des réseaux de Petri en LDF en français et en anglais:

```
NomFormalisme(Réseau de Petri,1, français,Rdp)
CreerClasse(NŒUD,place,rond)
CreerAttribut(nom,chaine,"",NOMINATION)
CreerAttribut(marquage,entier,0,PROPRIETE)

CreerClasse(CONNECTEUR,arc,fléche)
CreerAttribut(valuation,entier,1,PROPRIETE)
Connexion(arc,place,transition)
Connexion(arc,transition,place)

CreerClasse(NŒUD,transition,rectangle)
CreerAttribut(nom,CHAINE,"",NOMINATION)
```

↔

```
NomFormalisme(Petri net,1, anglais,Rdp)
CreerClasse(NŒUD,place,rond)
CreerAttribut(name,chaine,"",NOMINATION)
CreerAttribut(marking,entier,0,PROPRIETE)

CreerClasse(CONNECTEUR,arc,fléche)
CreerAttribut(valuation,entier,1,PROPRIETE)
Connexion(arc,place,transition)
Connexion(arc,transition,place)

CreerClasse(NŒUD,transition,rectangle)
CreerAttribut(name,CHAINE,"",NOMINATION)
```

Figure 5.3.d: Description du formalisme de nom interne "Rdp"

Nous avons montré qu'il est nécessaire d'avoir des outils (applications d'administration) de vérification des formalismes lors de l'ajout de nouveau formalismes. Dans la partie "gestion des services" (§5.4 Gestion des services), nous expliquons comment sont conçus de tels outils.

5.3.4. Modifications esthétiques d'un formalisme

Certaines parties esthétiques sont communes à tous les utilisateurs (ex: la forme de chaque nœud) pour permettre la communication d'informations graphiques. Par contre d'autres éléments esthétiques peuvent être **particularisés** (ex: taille de objets, couleurs…). Un problème survient lors de l'utilisation en mode autonome de l'application d'interaction utilisateur, il faut avoir une copie des formalismes nécessaires.

La plate-forme GUS doit conserver ces modifications esthétiques des formalismes pour chaque utilisateur. Dans ce chapitre, nous ne détaillerons pas cette caractéristique pour simplifier les explications car elle ne remet pas en cause le discours.

5.3.5. Les extensions futures du langage LDF

L'atelier gére des objets de différents formalismes. Il serait intéressant d'introduire la représentation des notions d'agrégation, décomposition et de hiérarchie dans le langage LDF. Ainsi, on aboutirait à une structuration plus riche que celle offerte par un modèle issu du modèle entité-relation tel que celui de PCTE.

La hiérarchie de classes de modèles est très utile dans la modélisation de système, elle permet d'une part la modélisation successive d'un système par raffinement et d'autre part la modélisation par l'utilisation de plusieurs formalismes. Une telle méthode de modélisation serait réalisable au niveau de l'édition de l'énoncé. Un énoncé serait alors composé de sous-énoncés.

Mais des problèmes théoriques surviennent lors de la remontée ou exportation des résultats sur chacun des sous-énoncés. Comment interpréter les résultats de chacun des sous-énoncés pour l'énoncé principal ? Actuellement, dans la plupart des cas, les résultats des sous-énoncés sont difficilement exploitables pour l'énoncé principal.

5.4. Gestion des services

L'utilisateur doit pouvoir effectuer des opérations sur ses énoncés au moyen de services. Dans un atelier de spécification la plupart des services offerts aux utilisateurs permettent d'analyser, valider et générer du code à partir de l'énoncé d'un utilisateur. Nous pouvons donner une définition d'un service utilisateur:

> **Définition**: Un Service, Options d'un service
>
> Un **service utilisateur** est un traitement sur un énoncé d'un utilisateur. Le nom d'un service est unique pour un formalisme et permet à un utilisateur de référencer une action spécifique.
>
> Les **options d'un service** servent à préciser son traitement. Un service se décompose donc généralement en options.

La définition d'un service est affinée en introduisant la notion d'**options d'un service**. Ces options précisent l'action du service, par exemple un service X permet un calcul sur un énoncé soit par la méthode A, soit par la méthode B.

Afin d'assurer l'intégrité et la cohérence des données, un service ne modifie en aucun cas les données initiales de l'utilisateur. Il est inconcevable de permettre, par exemple, à un compilateur de modifier le programme source de l'utilisateur. Dans le cas ou un service a pour objet de modifier les données de l'utilisateur comme lors d'une simulation, il faut fournir un moyen de dupliquer de manière automatique l'énoncé initial ou un moyen de revenir à l'énoncé initial. Nous détaillons plus loin les problèmes de description de résultats d'un service (§5.4.3 Représentation des résultats).

La plate-forme GUS est responsable de la **gestion** (ajout, suppression, liste…) **dynamique des services** (Objectif 16) ce qui entraîne une modification dynamique de la classe des formalismes. C'est un problème difficile à résoudre, même en utilisant un langage orienté objet, car la plupart des langages orientés objet n'offrent pas la possibilité de créer une classe dynamiquement [Broussard, et al., 1990]. Dans les langages autorisant la création dynamique de classes cette possibilité repose sur le fait que les classes sont considérées comme des objets.

La plate-forme GUS est uniquement responsable du **stockage et de l'accès aux différentes données** associées aux formalismes, sans en connaître la sémantique, ainsi le GUS ne connaît ni le langage LDF, ni le langage LDE.

Ces contraintes sur le GUS, entraînent l'impossibilité de modifier des données écrites en LDF et entre autres d'ajouter un nouveau formalisme ou un nouveau service… Nous étudions dans la section suivante, la solution consistant à définir des services spécifiques permettant la manipulation de ces différentes données.

Dictionnaire d'un formalismes

La notion de service permet de dégager les concepts théoriques sous-jacents aux formalismes. Un service associe une méthode au formalisme et la liste des services associée à un formalisme (classe) complète la définition du formalisme.

Les noms des objets du formalisme et l'ensemble des noms des services pour chaque formalisme constituent un **dictionnaire** du vocabulaire du modèle. Il comprend éventuellement leurs traductions dans différentes langues.

Les noms des services sont représentatifs des opérations demandées. Pour cela, le dictionnaire doit être créé par un utilisateur qui connaît parfaitement le domaine du formalisme et admis par l'ensemble des concepteurs et modélisateurs. Cet utilisateur est l'administrateur du formalisme.

> **Définition**: Le dictionnaire d'un formalisme
>
> Le **dictionnaire d'un formalisme** contient l'ensemble du vocabulaire lié au formalisme. Il n'est géré que par un administrateur de formalisme.

Par exemple, le dictionnaire des graphes contient les termes: "sommet", "arc" mais aussi "composantes connexes", "centre", "chemin minimal".

5.4.1. Les deux catégories de services

Les données personnelles et administratives gérées par le GUS se répartissent en deux catégories: les données ayant un lien avec le système d'exploitation (login…) et les données purement internes à l'atelier.

Dans le cas des données liées au système, la plate-forme GUS connait le contenu des données et offre des primitives de bas niveau (ex: rechercher le nom de login connaissant le nom de l'utilisateur).

Dans le second cas, la plate-forme GUS localise les données propres à l'atelier sans en connaitre la signification détaillée (exemple: les formalismes, les services…). L'interprétation de ces données est réalisée par la plate-forme GSV et par les applications d'administration. Nous devons donc introduire une nouvelle entité capable de vérifier la cohérence des données des leur création. Cette entité fait partie des outils d'administration. Il existe deux possibilités pour réaliser ces outils: soit créer des outils graphiques spécifiques au GUS pour la saisie, la vérification, etc…, soit se servir de la structure d'accueil de l'atelier pour entrer ces données en utilisant une application d'interaction utilisateur spécialisée (le programme d'Interface Utilisateur d'administration (IU-Adm)). Pour une meilleure intégration des outils et pour bénéficier d'une administration distribuée, nous avons retenu la seconde solution. La figure 5.4.a présente la solution retenue.

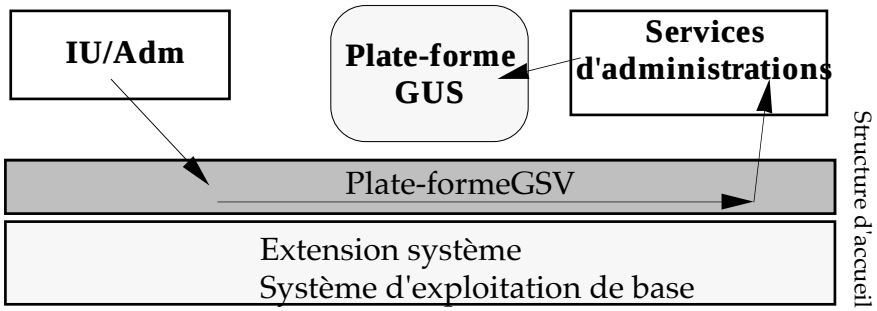


Figure 5.4.a: Administration du GUS

La plate-forme GUS donne la liste des services associés à un formalisme, mais elle ne permet pas la saisie d'un nouveau formalisme (langage LDF). Cette opération est réalisée par un service particulier d'administration des formalismes. Ce service connaît le langage LDF et vérifie le nouveau formalisme et sa cohérence avec les autres formalismes. Ce service utilise une fonction de la plate-forme GUS pour stocker ce nouveau formalisme.

Nous verrons dans la partie GSV, que cette plate-forme connaît, elle aussi, le langage LDF. Par l'utilisation de services particuliers (**services d'administration**), les deux plates-formes GUS et GSV travaillent avec les mêmes classes (formalismes, services…) mais ne se partagent aucun champ des objets manipulés. D'autre part, nous avons montré que les données liées au système d'exploitation n'étaient connues que de la plate-forme GUS, ainsi le GSV n'est pas lié aux données du système.

Nous venons de montrer qu'il existe deux catégories de services: les **services d'administration** et les **services utilisateurs**. Rappelons que le GUS gère les caractéristiques des services, mais ne permet pas leurs exécutions qui sont à la charge de la plate-forme GSV. Dans la partie GSV, nous aborderons le problème de l'accessibilité d'un service à l'utilisateur via les menus de l'interface utilisateur.

Les services d'administration permettent la gestion et l'évolution de l'atelier en incluant de nouvelles fonctionnalités. Une administration d'atelier repose sur le suivi et l'analyse de l'utilisation du système afin de déceler des anomalies et ainsi de tirer des informations utiles à la gestion. Des services donnant l'historique de l'utilisation des services, le nombre de fois que s'est connecté ou non un utilisateur, qu'un service n'a jamais été utilisé, le nombre d'utilisateurs connectés à un moment donné, etc....sont utiles à une bonne maîtrise du système.

> **Définition**: Les services d'administration
>
> Les **services d'administration** liés à des formalismes d'administration offrent un ensemble d'outils de gestion des données du GUS. Ces services gèrent d'une part, des données liées aux systèmes d'exploitation comme l'ajout d'un utilisateur, "l'accounting", et d'autres part les données du GSV comme l'ajout d'un formalisme, d'un service…

Notre étude se rapportant à un atelier de spécification, nous détaillerons donc les services utilisateurs qui permettent d'analyser, valider et générer du code à partir de l'énoncé d'un utilisateur.

Dans la suite de ce paragraphe sur le GUS, nous employerons par abus de langage le terme **service** pour **service utilisateur**. Nous avons défini un service comme un traitement sur un énoncé, nous allons maintenant expliciter le terme traitement.

5.4.2. Les types de traitements d'un service

L'analyse des besoins des utilisateurs et les caractéristiques systèmes d'un atelier de spécification nous permettent de définir les différents types de traitements, c'est à dire les caractéristiques principales des différents outils d'un atelier de spécification (outils d'analyses, de vérification et de génération de prototype).

La plupart des outils réalisent des calculs sur les données d'entrée, les services correspondants seront des **services de calcul**.

> **Définition**: Service de calcul
>
> Un service est un **service de calcul** s'il réalise des calculs sur les données en entrée.

Nous avons montré dans la partie I que les utilisateurs d'un atelier de spécification, emploient des outils permettant le changement de représentation (Objectif 15). Des services doivent donc effectuer des changements de représentation. Dans notre terminologie cela correspond à un changement de formalisme. Un service peut donc produire un résultat, un nouvel énoncé, dans un formalisme différent. Nous venons de définir le deuxième type de traitement d'un service: la **transformation** d'un énoncé dans un autre formalisme.

Définition: Service de transformation

Un service est un **service de transformation**, quand il prend en entrée un énoncé et rend en sortie un autre énoncé dans un formalisme différent.

Certains outils demandent de la part de l'utilisateur une action, comme par exemple la sélection d'objets de l'énoncé ou comme un choix durant le traitement. Le service a donc un traitement **interactif**.

Définition: Service interactif

Un service est un **service interactif** s'il demande une information supplémentaire à l'utilisateur.

Nous avons défini auparavant les données d'entrée nécessaires à un service; il faut maintenant préciser les sorties d'un service, c'est à dire les résultats qu'il fournit. Ces résultats vont nous permettre de dégager une représentation commune de données.

5.4.3. Représentation des résultats

L'analyse des besoins des utilisateurs et les caractéristiques systèmes d'un atelier de spécification nous permettent de présenter les différents types de résultats fournis par les outils d'un atelier de spécification (outils d'analyse, de vérification et de génération de prototypes). Les résultats des outils d'analyse peuvent avoir plusieurs formes depuis la plus simple, un résultat textuel jusqu'à la plus élaborée, un nouvel énoncé. Nous détaillons maintenant les différentes formes.

La plupart des outils d'analyse génèrent des **résultats textuels** c'est à dire des informations dans la langue de l'utilisateur exprimant ce résultat. Cette forme est sémantiquement la plus complexe à analyser mais la plus simple à stocker.

Rappelons que l'atelier fournit des moyens de communication entre services (Objectif 14), pour cela nous avons défini une représentation commune des résultats des services. Cette représentation est plus élaborée que la simple forme textuelle sans syntaxe ni sémantique. Nous définissons ainsi deux autres formes: le **langage LDE** de description des énoncés et le **langage LDR** de description des résultats.

Le langage LDE déjà utilisé en entrée, permet à un service de générer un nouvel énoncé.

Le langage LDR supporte des méthodes de représentation graphique de haut niveau pour visualiser les résultats. (Objectif 10). Par exemple, lorsqu'un service à besoin de mettre en évidence certains objets de l'énoncé. Comme pour le langage LDE, la plate-forme GUS ne connait pas la sémantique du langage LDR. Le langage LDR est décrit dans la partie plate-forme GSV (§6.4.3 Couche présentation).

Nous pouvons donner une définition plus précise d'un service:

Définition (complète): Un service

Un **Service** est un traitement effectué sur un énoncé. Selon sa nature, il peut produire un résultat (un nouvel énoncé) dans un formalisme différent. Si ce résultat doit être réutilisé pour des traitements ultérieurs, il constitue un nouvel énoncé. La plate-forme GUS est responsable de l'archivage de cet énoncé et de son association avec le nouveau formalisme.

Cohérence des données

Le GUS garde **l'association énoncé - service - résultat** ainsi il inclue des **d'informations d'historique** afin d'éviter une ré-exécution systématique des services lorsqu'un énoncé résultat a déjà été produit à partir de l'énoncé de départ. La plate-forme GUS gère dynamiquement l'ensemble des services déjà exécutés sur un énoncé. Nous avons déjà signalé dans le paragraphe §5.2 Gestion des utilisateurs que ces informations sont stockées dans le contexte dynamique de l'utilisateur.

Le GUS contruit une arborescence (Unix) pour les données personnelles (énoncé et résultats) de chaque utilisateur. Cette arborescence contient les informations de tous les énoncés d'un utilisateur. Ainsi, à chaque énoncé est associé un nœud de l'arbre. Le GUS offre des méthodes pour accéder aux nœuds de l'arbre. Les résultats des services sont stockés dans une sous-arborescence libellée par le nom de service. L'emplacement de la racine de l'arbre pour chaque utilisateur est déterminé par la gestion des utilisateurs du GUS qui permet de connaître l'emplacement des données de l'utilisateur (contexte statique) dans le système de fichiers distants. La gestion des fichiers distants rend uniforme l'accès aux fichiers des utilisateurs sur l'ensemble des sites supportant l'atelier (§3.2.1 Système/Répartition).

Nous avons montré dans la partie §5.2 du GUS (§5.2 Gestion des utilisateurs), qu'il est nécessaire d'**estampiller les énoncés** par la date de dernière modification de l'énoncé, par l'application d'interaction utilisateur et de stocker une copie de l'énoncé dans le langage LDE. L'estampille et l'énoncé sont stockés par le GUS dans l'arborescence de l'énoncé correspondant.

Nous avons signalé que le GUS conserve dans le contexte dynamique de l'utilisateur **l'adresse de transport** d'une partie de la plate-forme GSV. Cette adresse est stockée à la racine de l'arborescence.

La figure 5.4.b présente l'arborescence des données sur les énoncés, gérée par le GUS (Objectif 6).

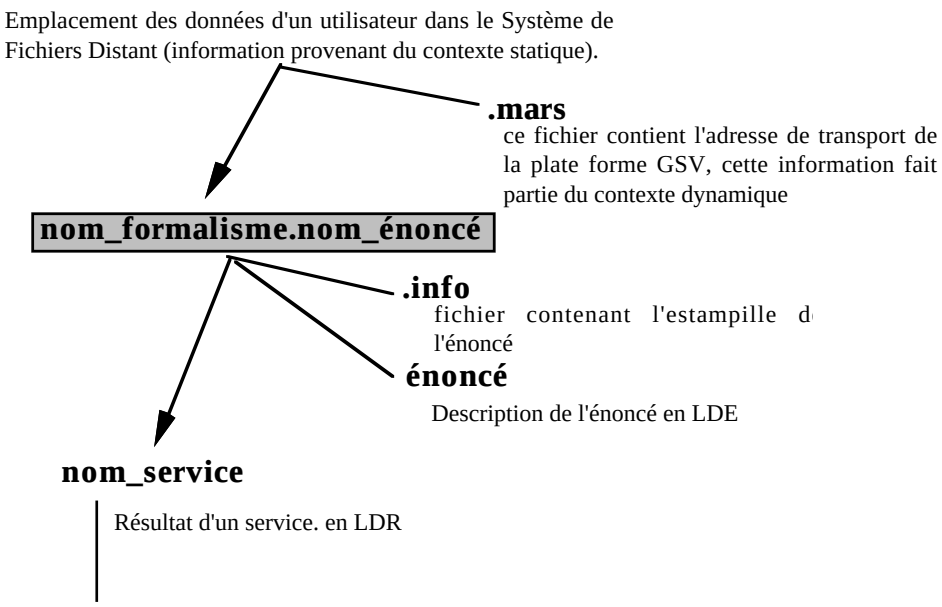


Figure 5.4.b: Gestion d'un historique sur un énoncé

Pour les services interactifs et de calcul, cette gestion des historiques est suffisante; en effet, par un parcours de cet arborescence, le GUS peut connaître pour un énoncé l'ensemble des services qui ont été demandés.

Dans le cas d'un service de transformation, ce service génère un nouvel énoncé; pour rester cohérent avec la gestion d'un historique cette transformation va créer une nouvelle sous-arborescence sous la racine. Mais le GUS doit garder le lien entre énoncés d'entrée et de résultats. Pour cela, nous complétons notre politique de gestion d'un historique en rajoutant le fait qu'un service de transformation peut générer une sous-arborescence historique d'un énoncé. Cet énoncé est aussi ajouté à la racine de l'arborescence. On obtient ainsi, une gestion complète d'un historique sur un énoncé (figure 5.4.c). Le GUS est capable de connaître les liens entre les énoncés (Objectifs 4, 6 et 7).

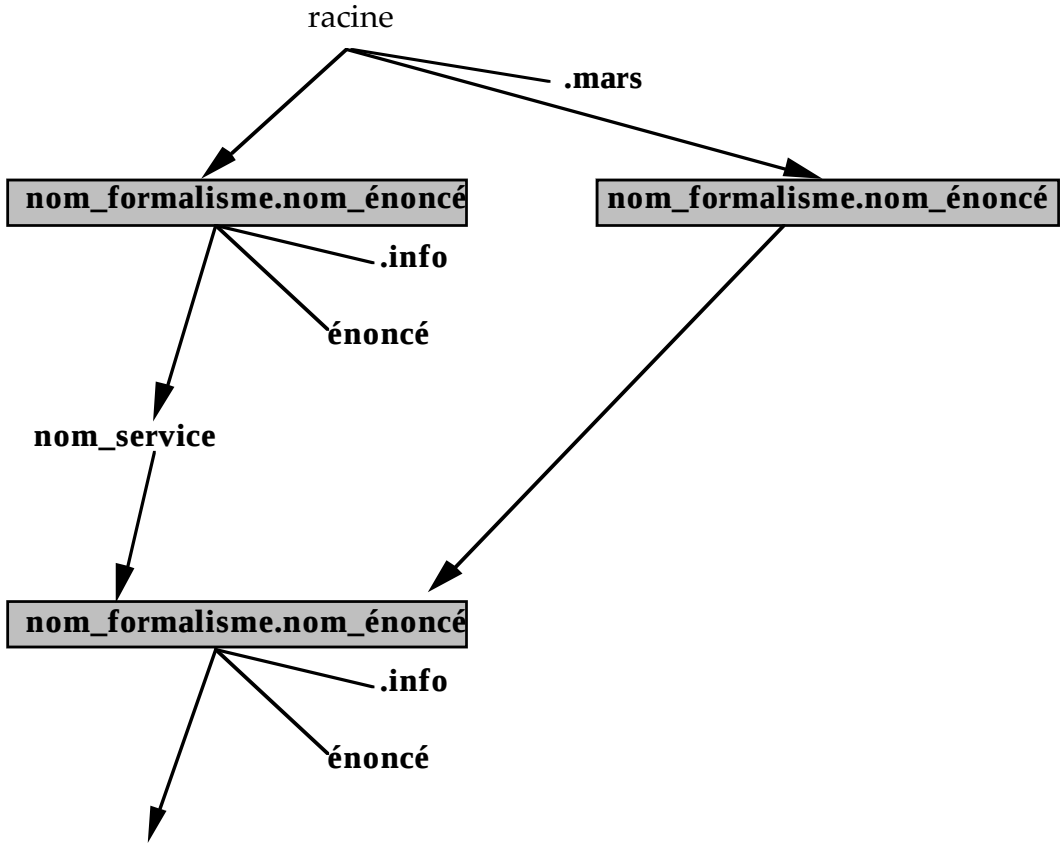


Figure 5.4.c: Gestion complète d'un historique sur un énoncé

5.4.4. Communication inter-services

Pour effectuer un communication inter-services (Objectifs 14, 22), le service demandeur indique le nom du service demandé, le formalisme des données en entrée ainsi que le type de résultats. Le GUS résout le problème de nommage de service dans notre environnement distribué: l'appel d'un service peut être assimilé à un appel de procédure distante (RPC) [Nelson, 1981] sans connaître la localisation du service demandé. Le passage des paramètres de la procédure et la récupération des résultats sont réalisés par l'utilisation des langages LDE et LDR (qui peuvent être assimilés à XDR [Sandberg, 1987]).

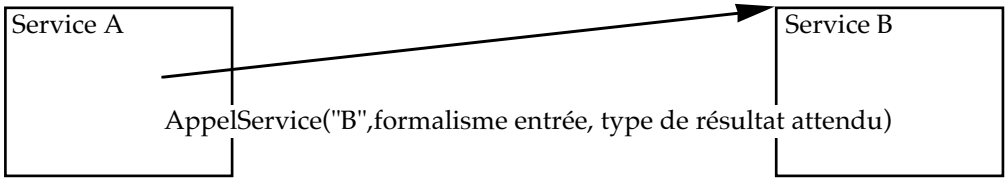


Figure 5.4.d: Appel de service

A l'inverse des appels de procédures distantes, l'appel de service n'est pas un appel procédural (synchrone) puisque après avoir appelé le service B, le service A continue sont exécution avant que le service B n'ait effectué le traitement. Le service A fournit les données d'entrée pour B (un énoncé LDE) puis attend les résultats fournis par B. Le problème de communication et d'échange des données est résolu par le module de communication inter-services. Nous détaillons le fonctionnement de cet appel dans le paragraphe §7.3 Interface de Communication. Nous pouvons dès lors envisager qu'un service appelle en parallèle plusieurs autres services.

Le GUS permet, par une méthode de gestion des historiques, de connaître à tout instant les services demandés sur des énoncés. Le service est une notion abstraite qui ne tient pas compte de la réalisation du traitement.

L'atelier propose deux solutions pour réaliser un service: un **programme d'application** et l'utilisation d'un système expert intégré. Dans la première solution, tout programme exécutable réalise un ou plusieurs services. Dans la deuxième solution, on associe à un service un paquet de règles (**application paquet de règles**). Le système expert se chargeant de réaliser le service avec ce paquet.

Nous étudions maintenant la gestion du système expert et son influence sur la communication inter-services.

Nous montrerons les influences de la gestion des applications sur la communication inter-services.

5.5. Gestion des programmes d'applications

Nous nous intéressons maintenant à la réalisation d'un service. Dans le domaine du logiciel, la seule notion pour réaliser (exécuter des actions) est un programme.

5.5.1. Les programmes d'application

Un programme d'application représente la **partie exécutable d'un service**. Un programme d'application respecte un protocole de dialogue avec l'atelier (protocole **D**ialogue **A**pplication **A**telier) qui normalise l'enchaînement des traitements des applications. Nous détaillerons ce protocole dans la partie GSV (§6.4.3 Couche présentation) qui permet l'exécution des applications.

Définition: Protocole de Dialogue Application Atelier

Le **protocole** de Dialogue Application Atelier **DAA** normalise l'enchaînement des traitements des applications. Il définit le dialogue des applications avec l'atelier. Ce protocole sera décrit dans la couche présentation de la plate-forme GSV (§6.3.4.a Scénario du protocole DAA)

Une application est soit un programme **compilé** ou soit un programme **interprété** soit un **ensemble de règles** exploitées par un moteur d'inférence. Dans le cas d'un programme interprété, il suffit que ce langage interprété autorise une communication inter-applications selon le protocole DAA à l'aide de ce langage. Notre atelier de spécification n'est donc pas lié à un type de langage. Il est capable d'accepter aussi bien des langages compilés qu'interprétés (Objectif 20).

Il est fréquent que plusieurs services soient regroupés dans une même application. Un service est défini dans un formalisme et une application ne regroupe que des services **d'un même formalisme**.

Il faut prévoir aussi qu'un service soit réalisé par plusieurs programmes d'applications. Dans ce cas, il est nécessaire que l'administrateur des formalismes indique à la plate-forme GUS l'application à utiliser par défaut pour la réalisation du service.

Pour la plate-forme GUS, les applications ne sont considérées qu'à un fort niveau de granularité; la plate-forme ne gère pas les différents programmes sources des applications, en effet, ceci est du ressort d'un AGL. Le GUS établit les liens entre les services et les applications.

Définition:

Un **Programme d'application** (ou Application) représente la partie concrète (l'exécutable d'un service), le service étant une notion abstraite. Plusieurs services d'un même formalisme peuvent être regroupés dans un programme d'application. Inversement, un service peut être réalisé par plusieurs programmes d'applications donc plusieurs codes exécutables.

5.5.2. Placement des programmes d'application

Vis à vis du système, un programme d'application est un **code exécutable**. Il dépend donc du type de machine et du système d'exploitation. Pour utiliser efficacement une application sur l'ensemble du site informatique supportant l'atelier et exploiter efficacement les ressources du site, nous devons résoudre le problème de localisation des codes d'une application.

Notre expérience en administration de systèmes, nous a montré que certaines applications ont besoin de fichiers de configuration. Il faut donc fournir un moyen de gérer le **code exécutable** des applications ainsi que leurs **fichiers de configuration**. La difficulté provient de l'hétérogénéité matérielle des machines supportant l'atelier (processeur RISC, 68000...), A chaque programme d'application, nous assoçions les sites sur lesquels il peut s'exécuter. De plus certains programmes ne sont exécutables que sur certaines machines qui possèdent l'environnement logiciel ou matériel adéquat. Par exemple, un programme ayant besoin de l'accès au compilateur Ada n'est exécutable que sur les machines du réseau pour lesquels un droit d'utilisation de ce compilateur a été acquis. La plate-forme GUS gère donc les **restrictions d'exécution** d'une application.

Les caractéristiques des sites sont gérées par la "gestion de configuration" (§3.2.2.a Gestion de la configuration) du niveau extension système. Le GUS doit donc prendre en compte les différents types de code exécutables d'une application. Pour chaque application, le GUS, en relation avec le système de configuration, fait des associations: application - type de machine, puis application - liste de machines. Il est possible de posséder plusieurs codes exécutables d'un programme d'application. Le GUS se charge alors de stocker ces différents code pour la même application. Le GUS associe à chaque programme d'application l'emplacement dans le système de fichiers (§3.2.1 Système - Répartition) du code ou de ces codes exécutables. On obtient ainsi plusieurs "localisations ou implantations" de l'application. On aboutit donc à la notion de copie d'une application. En fait, il y a des copies permanentes pour sécurité et des copies temporaires pour exécution sur une machine particulière.

Par cette gestion des codes exécutables, la plate-forme GUS résout les **problèmes de portages** des applications sur un type de machine; puisque l'application est toujours accessible à l'utilisateur.

Nous montrerons dans le paragraphe §5.7.3 (Renseignements sur les applications), que toutes les caractéristiques sur les applications sont regroupées dans des fichiers de renseignements gérés par le GUS.

5.5.3. Communication inter-applications

La communication inter-services définie précédemment implique une **communication inter-applications.** Cette communication est possible si le protocole d'appel de service est compatible avec le protocole DAA. Il n'est pas suffisant de permettre la communication entre applications: il faut l'administrer. Le GUS gère un double lien entre les applications clientes. Pour une application, on connait ainsi la liste de tous les services qu'elle appelle et en cas de modification d'une application, on connait la liste de celles qui l'utilisent. En effet cette communication va rendre dépendante les applications entre elles. Si, à la suite par exemple d'un changement important de système d'exploitation, une application ne fonctionne plus, il faut prévoir une solution de remplacement.

La notion d'application ne remet pas en cause celle de service. Nous avons montré que l'utilisation de la communication inter-services rend très complexes les liens entre les applications ce qui nécessite l'utilisation de services d'administration. Ces services d'administration permettent de parcourir les connaissances de l'atelier, pour une meilleur réutilisation des services. Ces outils de navigation sont appelés des "**browsers**" en génie logiciel; il permettent de se promener à travers les différentes entités gérées par le GUS comme dans Smalltalk 80 [Goldberg, 1984], le browser visualise graphiquement les données de tout le système. En quelque sorte, il faut pouvoir fournir aux utilisateurs des **catalogues des connaissances** de l'Atelier, aussi bien aux concepteurs de services et à l'administrateur. L'utilisateur final, quant à lui, doit avoir accès à degrés de précision et avoir une vision plus globale du catalogue des connaissances de l'atelier.

5.6. Gestion des applications paquets de règles

Nous avons vu dans la partie I que l'intégration d'un système expert (Objectif 21) à l'atelier de spécification permettait la programmation logique et assurait ainsi des extensions plus rapides et plus simples des outils. Le concepteur d'un service peut utiliser un système expert intégré pour écrire la réalisation de son service sous forme de paquet de règles. Dans ce cas l'application "paquet de règles" ne représente pas la **partie exécutable d'un service**, mais définit les paquets de règles associés au service. On a donc équivalence entre paquet de règles interprétable par un moteur d'inférence et programme d'application.

L'architecture fonctionnelle de l'atelier (Figure 5.6.a) prend en compte les applications écrites sous forme de paquets de règles et les autres applications. Un système expert sert d'interface entre les applications paquets de règles et le niveau plate-forme de l'atelier.

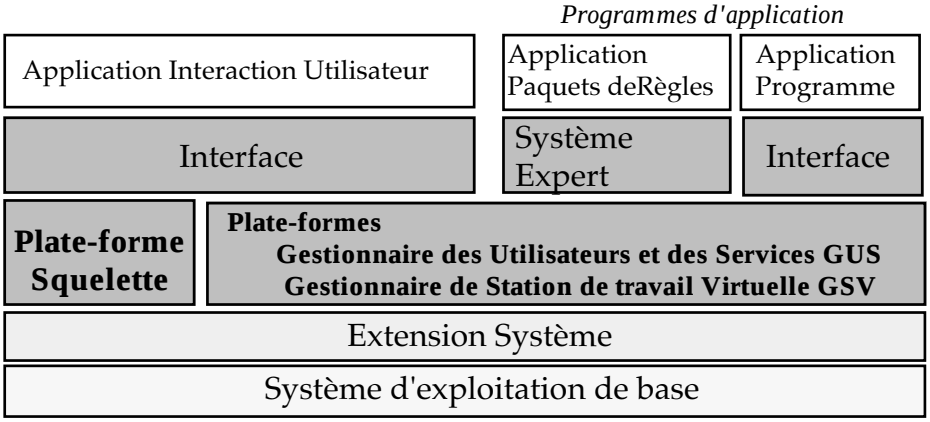


Figure 5.6.a: Les types de programmes d'application

Le GUS associe une information particulière sur le service traité par un paquet de règles, pour indiquer qu'il faut utiliser le système expert pour exécuter le service.

> **Définition**: But, Paquet de règles
>
> Un **but** est une propriété qu'on cherche à vérifier sur un ensemble d'objets. Pour atteindre un but, on fait appel à un ou plusieurs paquets de règles.
>
> Un **paquet de règles** est un ensemble de règles qui est associé à un problème donné. C'est donc la transcription d'une méthode de résolution à l'aide de règles.
>
> Naturellement plusieurs paquets peuvent être associés à un même but. Nous associons le nom du but du paquet de règle et le nom de service.

La plate-forme GUS gère l'association entre noms de services et noms de paquets de règles tout en mémorisant le fait que la réalisation du service s'effectue à l'aide du système expert.

La plate-forme GUS assure la localisation et le stockage des paquets de règles comme pour les codes des applications.

Nous expliquerons dans la partie GSV comment est réalisée l'exécution d'un service puisque maintenant il existe deux techniques.

Mais, il faut vérifier que l'utilisation d'une application paquets de règles ne remette pas en cause la communication entre services.

Nous avons pris comme option d'intégrer le système expert dans l'atelier. En conséquence, il est capable de recevoir un énoncé et de charger un paquet de règles correspondant au service demandé puis de fournir des résultats compatibles avec les représentations de l'atelier.

En partant des principes que nous venons de donner, nous allons montrer que la communication inter-services s'effectue aussi bien dans le sens système expert à un service réalisé par un programme d'application que dans le sens programme d'application à système expert et dans le sens système expert à système expert. Nous en déduisons des caractéristiques techniques que nous avons imposées au système expert avant de l'intégrer dans l'atelier.

5.6.1. Communication système expert à système expert

Pour réaliser des services, le moteur d'inférence du système expert applique des paquets de règles successifs associés aux divers buts qu'il choisit au fur et à mesure de ses inférences. Le système expert en cours d'inférence sur un paquet de règles a donc besoin de charger un autre paquet de règles. Ce chargement correspond pour l'atelier a un appel de service. Mais cet l'appel de service est particulier: le système expert exécute la fonction "**AppelService**" en donnant le nom du paquet et celui du formalisme associé au paquet. La communication entre le paquet déjà présent et le paquet à charger est réalisée en interne par le système expert. La localisation du paquet et son chargement éventuel s'il n'est pas déjà présent est à la charge de la plate-forme GUS.

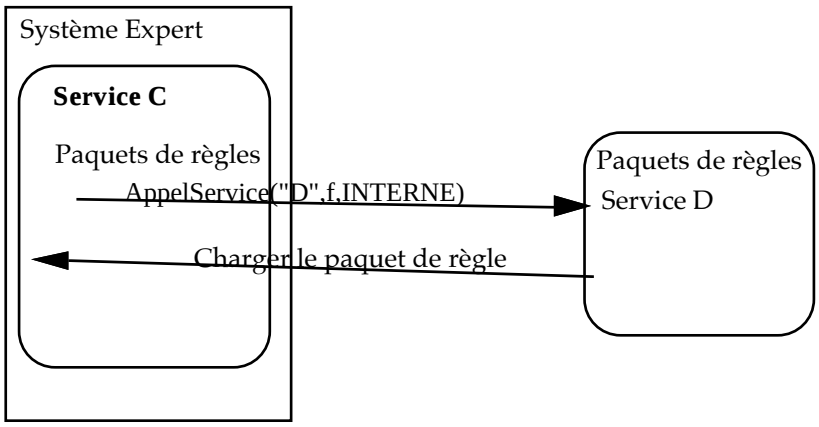


Figure 5.6.b: Communication système expert à système expert

5.6.2. Communication entre système expert et programme d'application

Le système expert doit être capable, en cours d'inférence, de demander et de recevoir des faits issus d'autre applications en particulier d'application d'interface utilisateur, cette communication consiste, dans certains cas, à recevoir un nouvel énoncé. Pour cela, le système fait un appel de service comme défini dans le paragraphe communication inter-services en fournissant l'énoncé d'entrée pour ce service, puis récupére les données résultat suivant le type spécifié dans l'appel.

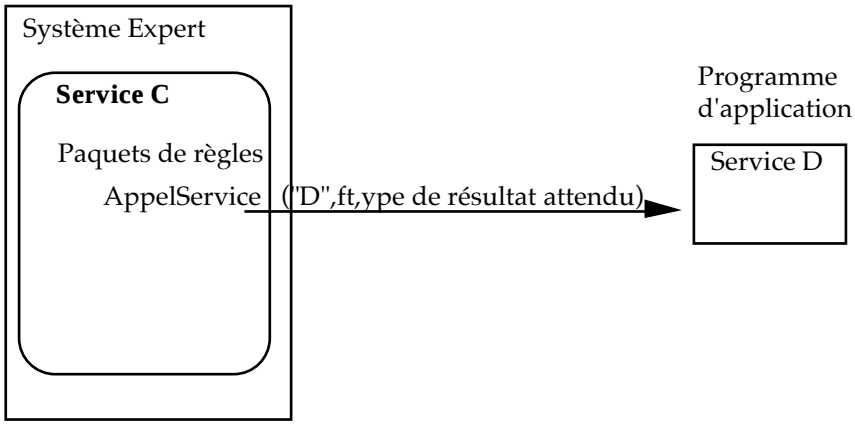


Figure 5.6.c: Communication entre système expert et programme d'application

La communication programme d'application à système expert ne pose aucun problème puisque nous sommes partis du principe que le système expert était intégré à l'atelier, ainsi le protocole d'exécution d'un service est compatible avec le protocole d'appel de service.

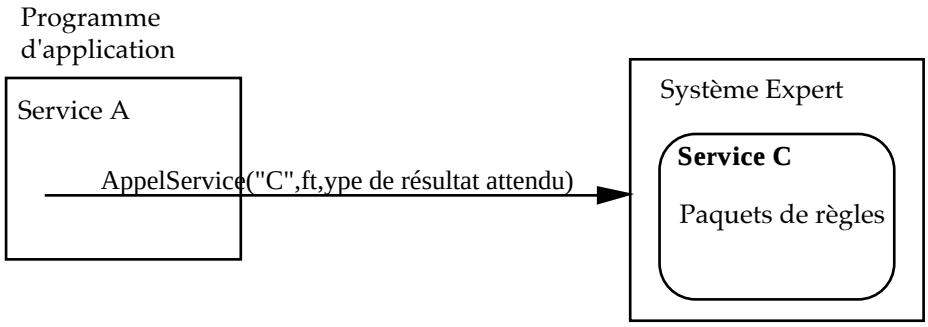


Figure 5.6.d: Communication entre programme d'application et système expert

5.6.3. Conclusion

Nous avons homogénéisé la notion de programme d'application (code exécutable) et celle de paquets de règles (code interprété). La **réalisation d'un service** peut être traitée soit par un programme classique (programme d'application) soit par l'association système expert et paquet de règles.

Nous avons ainsi introduit la notion de **machine virtuelle** capable d'exécuter du code ou d'interpréter des règles. Par abus de langage, nous utiliserons le terme **application** pour regrouper les programmes d'application et les programmes paquets de règles. Nous allons maintenant mettre en évidence les différents états d'une telle application au cours de son cycle de vie.

5.7. Les différents états d'une d'application

5.7.1. Cycle de vie d'une application

Les différents types d'utilisateurs (administrateur, concepteur, modélisateur) développent ou utilisent des applications qui passent par différents stades d'élaboration ou de développement (cycle de vie du logiciel). On distingue trois états possibles d'un programme d'application en **développement**, en **test**, en **exploitation**. Cette distinction aboutit à organiser le développement en différents **espaces de travail** [Bourgeois, et al., 1988]. L'**espace privé** correspond à l'espace propre à chaque développeur, l'**espace d'intégration** concerne l'administrateur de l'atelier et l'**espace officiel** contient l'ensemble des services validés placés sous la gestion de configuration. Cette démarche usuelle est conforme au Plan Qualité logiciel [Galinier, 1988].

Une application en **développement** n'est utilisable que par l'administrateur de l'atelier ou par son concepteur. Nous avons défini des méthodes de développement à respecter dans AMI; mais sans fournir un environnement de développement (Standardised Interface for Integrated Programming).

Un tel type d'environnement permet au programmeur de gérer un projet, une méthode (Merise, SADT …), la construction du programme, sa documentation. Ces types d'environnement ne sont pas liés à un type particulier de système d'exploitation ou même de machine et pourrait s'intégrer dans notre atelier.

Dans un environnement distribué hétérogène, le but d'un développeur est que la présentation et l'utilisation des outils soient identiques, en regard du système d'exploitation et des architectures matérielles des machines. Nous laissons le choix libre au concepteur de service d'employer n'importe quel type d'environnement et aussi de langage comme Ada, Eiffel, et même des langages interprétés comme Lisp.

Une application en **test** est dans un état intermédiaire et son utilisation est réservée à son concepteur, à l'administrateur de l'atelier ou à quelques utilisateurs choisis auxquels sont donnés les droits d'accès. L'objectif des tests est de déceler qu'un programme ne marche pas. Le concepteur du service doit publier ces tests pour permettre de certifier ce service, le service est donc fiable (Aptitude du système à assurer exactement ses fonctions, définies par le cahier des charges et la spécification) [Meyer, 1988]).

Les applications en **exploitation** sont des programmes ou des paquets de règles qui dont le développement est achevé. Leur utilisation est la plus large possible.

5.7.2. Evolution des droits

Nous montrons comment évoluent les liens entre les états des applications et les droits des utilisateurs. L'accès des applications dépend des droits des utilisateurs (§5.2 Gestion des utilisateurs). Il faut donc définir une hiérarchie de niveaux de modification des objets. A chaque niveau est associé un espace de travail. Le premier niveau correspond à l'espace de modification de l'utilisateur, les modifications sont locales… Dés que le concepteur veut intégrer son service; une coopération entre le concepteur et l'administrateur permet d'intègrer son service dans l'atelier. Ce service passe dans l'état test et est dans l'attente d'une **officialisation** sous le contrôle de l'administrateur. Si cette application est valide (passage de tests) alors l'administrateur la fait passer dans l'état exploitation et elle devient accessible par tous les utilisateurs. L'application devient une **référence** commune.

La plate-forme GUS associe à un utilisateur les applications auxquelles il a le droit d'accèder. Les règles de droits des utilisateurs correspondent à des attributs supplémentaires dans la classe utilisateur. A chaque type d'utilisateur est associée une liste des services accessibles. Les droits d'accès des utilisateurs aux services sont définis par les règles suivantes:

- L'administrateur de l'Atelier accéde à tous les services utilisateurs, et aux services d'administration.
- L'administrateur des formalismes accéde à l'ensemble des services du formalisme qu'il administre.
- Un concepteur de services accède aux services qu'il développe (état en développement ou en test).
- Un utilisateur final n'accède qu'aux services de modélisation et d'analyse et à certains services d'administrations.

5.7.3. Renseignements sur les applications

Nous étudions les conséquences de la définition d'un programme d'application par rapport à la notion de service.

Rappelons que GUS ne permet pas l'exécution d'une application mais seulement la gestion des informations qui décrivent cette application.

Nous venons de déterminer certaines des informations de description sur le comportement des programmes d'applications qui sont stockés par le GUS dans des fichiers de renseignements.

Des services d'administration permettent de mettre à jour ces fichiers et d'extraire les données nécessaires.

Définition: Renseignements sur les applications

Les **renseignements** associés à une application contiennent des descriptions de son comportement. Ces renseignements sont deux types, les informations relatives à l'environnement de l'atelier (état de l'application, droits d'accès) et les informations pour l'exécution du programme (type de code, restriction d'exécution, utilisation d'un interpréteur ou du système expert…).

La plate-forme GSV et des outils d'administration gère le contenu de ces fichiers de renseignements. Nous complèterons la description du comportement des applications vis à vis de la structure d'accueil dans la partie GSV (§6.4.3 Couche présentation) qui permet l'exécution d'une application.

5.8. Fonctionnalités d'une application et arbre de questions

Un de nos objectif essentiel (Objectifs 23, 24) est que la notion de **service** (données-résultats) soit **indépendante de l'environnement hétérogène** tandis que la partie dynamique d'un service (**programme d'application**) reste **liée** fortement à cet environnement.

Dans tout environnement logiciel, les applications offrent des fonctionnalités aux utilisateurs. Dans les environnements possédant une interface utilisateur conviviale, ces fonctionnalités sont accessibles par des menus [APPLE, 1985]. C'est pour cela que nous avons intégré la présentation des menus dans l'application d'interaction utilisateur et que nous avons définit la notion d'arbre de questions ansi que son langage pour permettre de décrire de manière externe les fonctionnalités des applications.

Définition: Fonctionnalités d'une application

Une application offre un ensemble de **fonctionnalités** accessibles à l'utilisateur. Ces fonctionnalités correspondent aux questions que l'on peut poser à l'application. L'exécution d'une fonctionnalité d'une application correspond à une réalisation d'un service. A chaque fonctionnalité correspond un nom de service.

L'ensemble des questions d'une application est regroupé dans un **arbre de questions**. Cet arbre correspond à l'ensemble des réalisations de services que compose l'application.

Définition: Arbre de questions

Un **arbre de questions** est un regroupement des fonctionnalités d'une application: les questions et leurs options.

On en déduit qu'un arbre de questions est une vue de l'ensemble des services et de leurs options pour une application. Cette vue des services par un arbre de question est cohérente avec la communication inter-services telle qu'elle est décrite au paragraphe §5.4.4 Communication inter-services.

5.8.1. Le Langage d'Arbre de Questions

L'arbre de questions qui décrit le comportement de l'application fait partie des renseignements sur l'application. Sa description est externe aux formalismes et aux applications.

Nous avons défini un langage **LAQ** (Langage d'Arbre de Questions) qui permet de structurer de façon hiérarchique les questions que l'on peut poser à un programme d'application. Ce langage est évidemment caché par un service d'administration permettant de le produire graphiquement.

Définition: Le langage LAQ

Le **langage LAQ** permet de décrire de façon hiérarchique, les questions que l'on peut poser à un programme d'application. Les questions représentent les services que réalise ce programme.

```
nom : identificateur
NomArbre(nom_application)
CreerQuestion(nom_service)
CreerOption(nom_service, nom_option_service)
```

Il faut autant de CreerOption que d'options du service.

Nous allons affiner la description du langage LAQ au fur et à mesure que nous introduirons ces différentes options. Nous verrons que le langage LAQ est extensible ce qui permet de le faire évoluer. Une description complète du langage LAQ se trouve dans la partie III (§4 Les langages de communication).

5.8.2. Les types de questions

Nous avons caractérisé trois types de services (§5.4.2 Les types de traitements d'un service): les services de calcul, de transformation et interactif. Pour les applications qui réalisent les traitements de calcul, la définition de l'arbre de question est suffisante. Pour celles qui réalisent des transformations, il faut rajouter le nom du formalisme de sortie des résultats. Le langage LAQ inclut donc une option spéciale pour cela:

```
OptionTransformation(nom_option_service, formalisme)
```

Pour les services interactifs, les applications qui réalisent ces traitements ont besoin d'un dialogue beaucoup plus complexe comme par exemple une sélection d'objets, de multiples objets ou l'utilisation d'une fenêtre de dialogue proposant des choix non déterministes (calculés en fonction des données d'entrée).

Le langage LAQ permet le choix d'une sélection d'objets quelconque associée à une question:

```
TYPE_DE_SELECTION: [au plus un, un et un seul, au moins un,
  indifférent]
OptionSelection(nom_option_service, TYPE_DE_SELECTION)
```

Pour permettre une sélection plus fine des objets de l'énoncé, le langage LAQ permet de désigner les types des objets pris en compte.

```
OptionSelectionFine
                (nom_option_service,TYPE_DE_SELECTION,nom_classe_objet)
```

a. Remarque

Pour obtenir un arbre de questions en plusieurs langues, il faut définir le même arbre de questions en langage LAQ dans chacune des langues en respectant l'ordre de définition des différentes questions. L'association se fait par lignes et on ajoute au langage LAQ la nom de la langue utilisée comme pour la gestion multi-linguismes des formalismes (§5.3.2 Gestion dynamique des formalismes).

NomArbre(nom_application, LANGUE)

5.8.3. Les résultats

Nous avons défini précédemment les résultats d'un service (§5.4.3 Représentation des résultats), or la représentation des résultats produite par un programme d'application peut être dépendante de l'architecture physique de la machine sur laquelle l'application s'est exécutée. Pour rester cohérent avec la définition des types de résultats d'un service, dans tous les cas un programme d'application fournit des résultats dans les langages LDR (Langage de Description des Résultats) ou LDE (Langage de description des Enoncés). Toutefois il peut par exemple, stocker sur disque ces données internes sous forme de structures (dépendantes du processeur de la machine). Ceci oblige, lors de l'exécution ultérieure de cette application sur le même énoncé, de l'exécuter sur le même type de machine, pour qu'elle puisse récupérer ces données. Le GUS garde donc dans **l'historique d'un énoncé** les caractéristiques des résultats non pas des services mais des programmes d'application. Il ajoute dans l'historique d'un énoncé les applications qui ont travaillé sur cet énoncé; en effet, nous avons montré (Figure 5.4.b : Gestion d'un historique sur un énoncé) que le GUS conservait un historique des services. Puisqu'une application peut réaliser plusieurs services, nous structurons l'arborescence d'un historique sur un énoncé, en ajoutant un niveau supplémentaire qui représente les applications qui ont travaillés sur l'énoncé. Nous complétons la Figure 5.4.b.

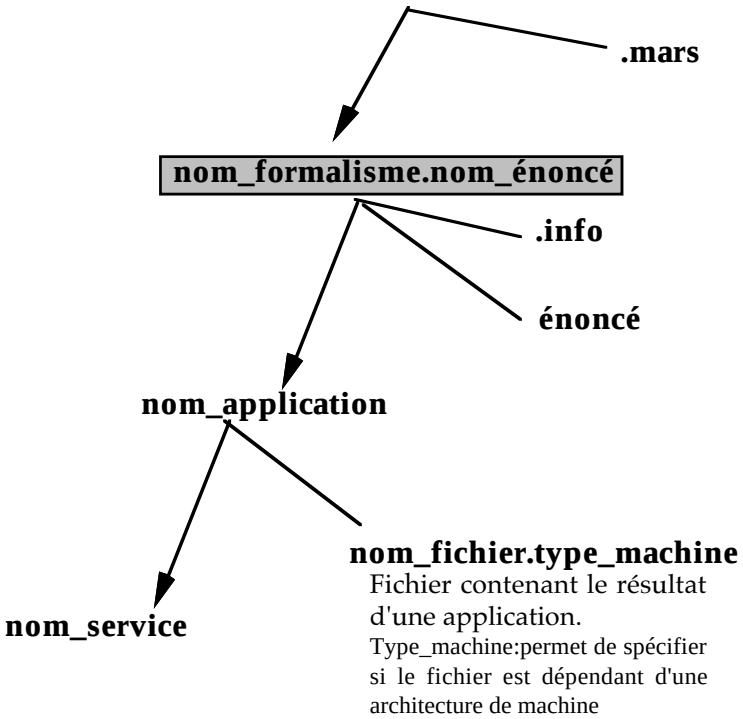


Figure 5.8: Gestion d'un historique sur un énoncé

Cette structure de l'arborescence est indépendante de la gestion des fichiers relatifs à une application.

5.9. Synthèse sur les liens entre formalismes, services et applications

Nous pouvons maintenant présenter sur un exemple, un schéma complet qui montre les liens entre les formalismes, les services et les applications. La figure 5.9 présente un exemple d'arbre de question dans le cadre des réseaux de Petri.

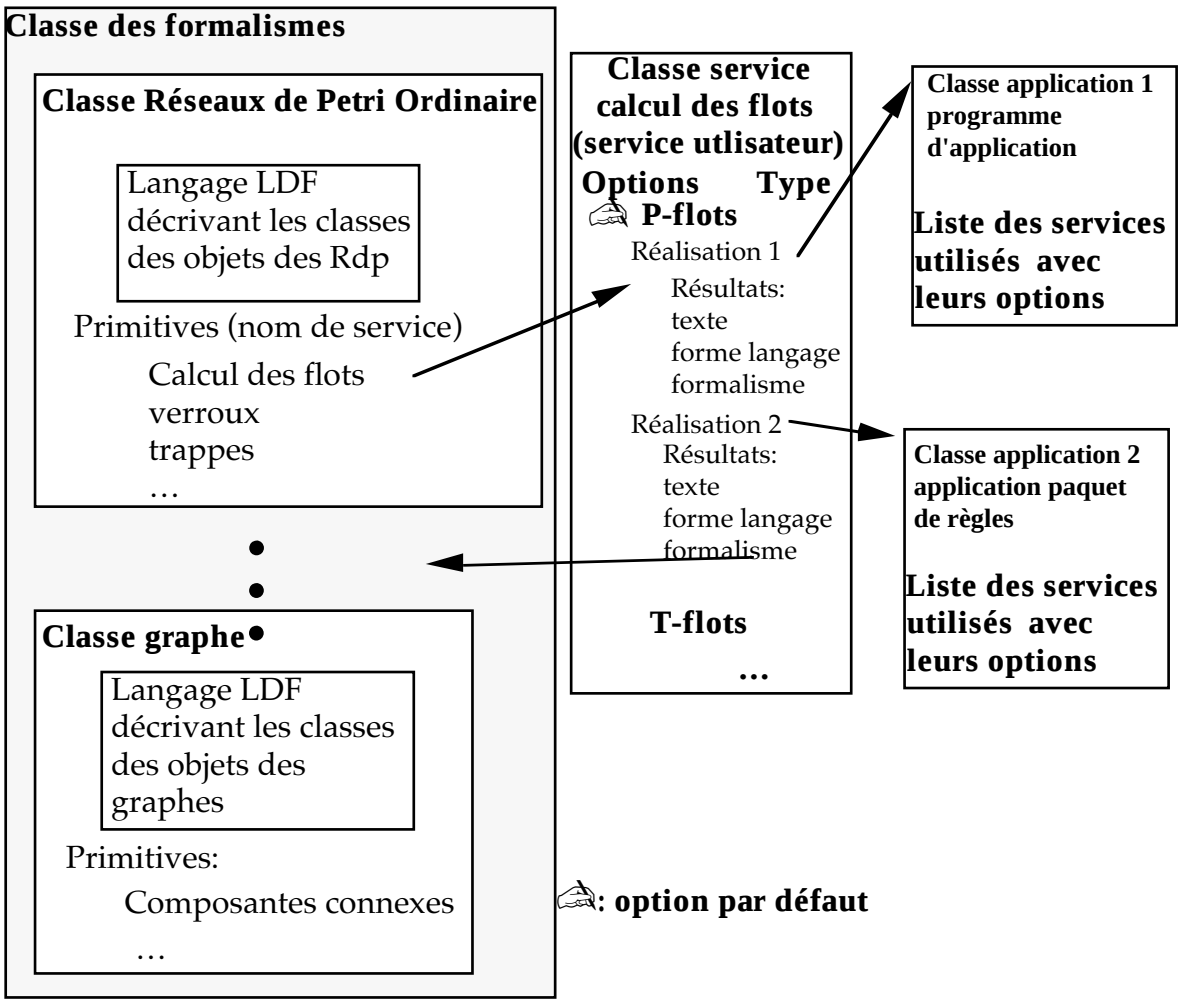


Figure 5.9: Les liens entre formalismes, services et applications

Dans notre discours, nous nous sommes volontairement limité à l'arborescence des options d'un service à un niveau mais il est possible de créer des options d'options ce qui complique les explications du GUS sans en changer le principe.

5.10. Architecture du GUS dans un environnement distribué

Nous avons présenté de manière détaillée l'ensemble des données et primitives de la plate-forme GUS. Nous avons montré que ces données du GUS se répartissent en deux catégories les **données administratives** et **personnelles**. Nous avons montré que certaines de ces données dépendaient d'informations liées au système d'exploitation. Le GUS connaît en détail certaines données, y compris leur sémantique, et n'a besoin de connaitre pour d'autres que leur localisation.

Après cette description des données, nous présentons une architecture logicielle pour le GUS qui satisfait aux contraintes de notre environnement distribué et à la gestion des données telle que nous l'avons décrite.

L'hétérogénéité et la distribution de l'atelier la mise en place d'une organisation pour accéder aux données de l'atelier à tout moment sans qu'une quelconque défaillance survenant au niveau d'un site perturbe le bon fonctionnement global de l'atelier. Le problème à traiter est ici l'accès à des données et leur disponibilité dans un environnement distribué. Il faut donc trouver des solutions de stockage et d'accès aux données tolérantes aux pannes.

Par nature, les **données administratives** ont une **fréquence de mise à jour peu élevée**. Dans ce cas, nous pouvons nous contenter d'une consultation minimale (même en mode dégradé). Ainsi en cas de panne majeure, il est acceptable de renoncer momentanément à intégrer un nouveau service ou un nouveau formalisme.

Toutefois il reste indispensable d'accéder aux données d'administration afin de continuer à assurer des service. Nous avons adopté une solution classique pour gérer le problème de disponibilité des informations: la duplication des données sur différents sites. Un des problèmes de cette duplication de données est la cohérence des données sur les sites; il faut avoir la même information partout. La mise à jour de données dupliquées [Gardarin, 1989] est un sujet complexe. Les **données personnelles** ont une fréquence de **mise à jour plus élevée**, par exemple on ne peut pas interdire à un utilisateur de travailler avec l'atelier du fait qu'il est impossible de mettre à jour son contexte dynamique.

En analysant les différentes données gérées par le GUS, on remarque que l'application d'interaction utilisateur ne stocke pas de données personnelles et n'a besoin que de consultations de données. Nous avons donc séparé la plate-forme GUS en un GUS-client et un GUS-serveur. Ainsi les données à mettre à jour sont du ressort de GUS-serveur et le GUS-client ne fait que des requêtes pour lire des données et si necessaire pour demander la mise à jour de données personnelles au GUS-serveur. Ainsi, les problèmes d'accès à des données dans notre environnement distribué avec une forte disponibilité sont reportés du seul côté GUS-serveur.

Nous présentons maintenant une architecture plus détaillée sur les liens entre les deux plate-formes GUS et GSV. La figure 5.10 montre le découpage de la plate-forme GUS. Les requêtes du GUS client sont transmises au GUS serveur par l'intermédiaire de la plate-forme GSV. Le serveur après avoir récupéré les informations demandées transmet au client les réponses.

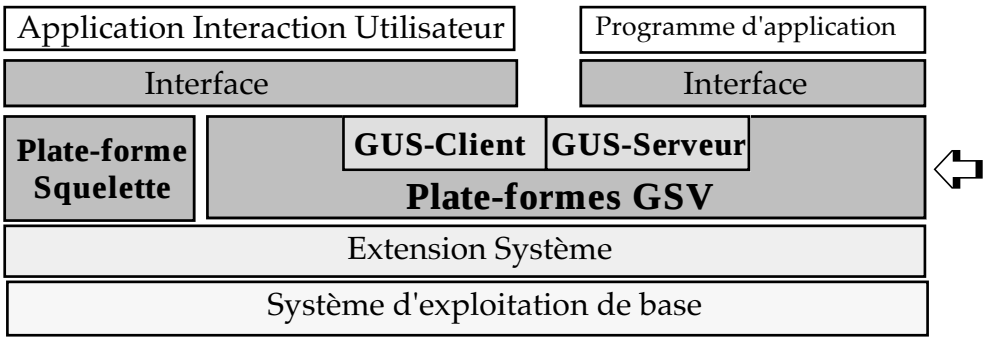


Figure 5.10: Architecture fonctionnelle simplifiée

Un de nos objectifs (Objectif 18) est que la station utilisateur locale puisse être utilisée de manière autonome, déconnectée des serveurs. Pour permettre une utilisation autonome de l'application d'interaction utilisateur, le GUS-client **intégre la partie serveur correspondant à la gestion des formalismes**. Les descriptions des formalismes sont dupliquées sur les stations offrant la possibilité de travailler en autonome. Le GUS-client possède des primitives pour obtenir la liste des formalismes locaux. En conséquence, l'administrateur des formalismes (du GUS-serveur) ne peut en aucun cas intervenir dans la gestion des formalismes locaux. Le GUS-client compare les formalismes locaux et distants et en cas d'anomalie,le GUS-client en informe l'administrateur des formalismes, par exemple par l'utilisation du courrier électronique. Il faut prévoir la transmission d'une version correcte du formalisme, soit manuellement par l'intermédiaire d'un serveur de fichiers, soit automatiquement par un service d'administration, chargé du téléchargement de formalismes et de la mise à niveau d'énoncés construits avec d'anciennes versions du formalisme.

Une autre conséquence d'avoir séparé le GUS en deux parties nous permet d'associer la **gestion de la classe forme graphique** (§5.3 Gestion des formalismes) uniquement au GUS-Client.

Une conséquence directe de cette architecture du GUS est que la totalité des données de l'atelier est traitée par le GUS-serveur. Il est donc envisageable de posséder ces données dans plusieurs langues (par exemple formalisme en anglais, français, allemand…). Le multi-linguismes est donc possible grâce à cette architecture.

Les problèmes de stockage et d'accès aux données tolérantes aux pannes sont ainsi traités au niveau du seul GUS-serveur. Nous avons défini dans cette partie les contraintes que doit respecter le GUS-serveur. L'architecture du GUS-serveur pour être tolérante aux pannes et permettre l'accès aux données et la mise à jour dépend de l'architecture matérielle et du système de l'environnement. Nous donnerons dans la partie III, la solution que nous avons retenue pour la réalisation de notre atelier AMI.

6. Niveau plate-forme - Gestion de Station de travail Virtuelle

6.1. Introduction

Le Gestionnaire de Station de travail Virtuel (GSV) est une plate-forme d'accueil d'applications (Figure 6.1) qui joue un rôle primordial pour l'ouverture de l'atelier. La plate-forme GSV représente la composante station de travail virtuelle telle que nous l'avons décrite dans la partie I (§5.7 Station de travail virtuelle). La plate-forme GUS gère des données nécessaires à l'atelier. La plate-forme GSV assure le transport d'informations entre l'application d'interaction utilisateur et les programmes d'applications. La plate-forme GSV rend indépendants les programmes d'application de l'interface utilisateur.

Pour permettre une lecture plus aisée, nous présentons de nouveau l'architecture fonctionnelle sous forme d'une figure simplifiée (Figure 6.1) mettant en évidence les deux plates-formes Gestion des Utilisateurs et des Services (GUS) et Gestionnaire de Station de travail Virtuelle (GSV).

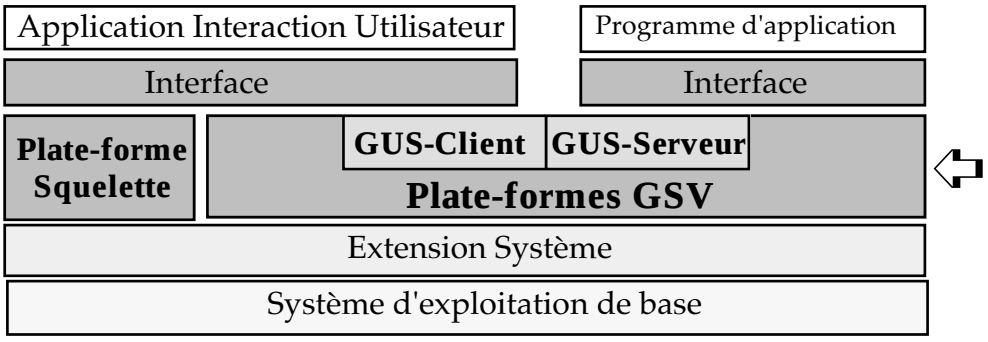


Figure 6.1: Architecture fonctionnelle simplifiée

Dans l'environnement multi-sessions, plusieurs services peuvent être activés simultanément. Cette caractéristique, si elle est observée par l'utilisateur comme une simple gestion de multi-fenêtrages, dissimule, au niveau du gestionnaire de station de travail virtuel (GSV), des mécanismes de gestion de contextes, soumis à de multiples et intempestives commutations en intégrant les contraintes d'un environnement réparti.

Un des rôles du GSV consiste à gèrer entièrement les fenêtres et leur contenu pour diminuer les temps de réactions et alléger le travail des programmes d'applications. Les événements de bas niveaux sont traités localement et les événements transmis aux applications sont donc de plus haut niveau. Ainsi, les applications ne manipulent pas de données graphiques, elles n'ont pas la connaissance de fenêtres associées aux données graphiques en entrée et en sortie, elles ne gèrent pas les menus. L'application d'interaction utilisateur prend en charge la gestion des fenêtres, des menus, du graphisme et des sessions multiples (plusieurs applications accessibles à un instant donné à partir de l'interface utilisateur).

La plate-forme GSV gère donc l'accès concurrents à l'interface utilisateur et l'indépendance des applications vis à vis d'un système de fenêtrage, la synchronisation application-utilisateur, ainsi que le nommage des services et la tolérance aux pannes dans notre environnement distribué. Ainsi, l'utilisateur n'a pas à connaître la topologie du réseau ni le type des machines et l'accès aux services.

Après une étude des systèmes de fenêtrage, nous décrivons les techniques que nous avons choisies pour résoudre les problèmes d'une architecture en couches pour réaliser cette plate-forme GSV et les objets associés à chacune des couches. Nous en déduirons les événements (messages) qui circulent entre l'application d'interaction utilisateur et les programmes d'applications de l'atelier.

6.2. Le GSV: Un système de fenêtrage de haut niveau

Les systèmes de fenêtrage existent sur la plupart des postes de travail. Ils sont devenus l'interface utilisateur de base des systèmes d'exploitation multi-tâches. Dans ce type de configuration, les systèmes de fenêtrage font face à deux problèmes: le partage de l'écran entre les fenêtres et l'accès à une fenêtre par plusieurs processus. Le système de fenêtrage affecte directement le dialogue homme-machine.

Le GSV peut être vu comme un système de multi-fenêtrages . Nous examinons l'architecture possible d'un tel système de fenêtrage dans un environnement distribué hétérogène. Les **problèmes** majeurs sont liés à la **communication** dans un tel environnement et à la **synchronisation** des applications qui accèdent simultanément à l'interface utilisateur.

Nous montrons d'abord ce qu'attendent les utilisateurs d'un système de fenêtrage, nous en déduisons des caractéristiques sur les systèmes de fenêtrage.

Nous présentons ensuite l'architecture des système de fenêtrage actuels utilisant la notion de réseau de machines. Nous montrons leur faiblesse pour travailler dans un environnement distribué hétérogène. Nous appuyons notre travail par l'étude du système de fenêtrage qui nous a semblé le plus représentatif: le système X-Window.

Enfin, nous en déduirons les caractéristiques et les propriétés que doit comporter notre système de fenêtrage GSV.

6.2.1. Que doit respecter un système de fenêtrage ?

Les utilisateurs d'un système de fenêtrage souhaitent [Prosser, 1985] une compatibilité, une inter-communication, une portabilité, l'utilisation d'un réseau de machines et une boite à outils de programmation.

- Compatibilité
  Les anciennes applications doivent pouvoir être utilisées mais cela doit aller plus loin, cet environnement doit pouvoir en améliorer l'utilisation (utilisation d'une interface graphique).
- Inter-communication
  Chaque fenêtre possède un contexte d'exécution et il doit être possible de partager de l'information entre fenêtres. Par exemple le couper/coller d'une fenêtre à une autre.
- Portabilité
  Les applications ne sont pas liées à un type de système de fenêtrage.
- Système distribué
  Le système de fenêtrage utilise de manière élégante la notion de réseau de machines.
- Boite à outils de programmation
  Un programmeur a besoin d'outils pour exploiter les capacités du multi-fenêtrages aussi bien pour créer de nouvelles applications que pour modifier d'anciens logiciels. Cette boite à outils doit être fournie avec un guide pour la construction d'applications.

6.2.2. Architecture des systèmes de fenêtrage actuels

Une fenêtre est un contexte pour afficher et interagir avec l'utilisateur. Des mécanismes sont associés à la fenêtre: création, destruction, déplacement, manipulation, redessiner le contour des fenêtres… Ces mécanismes sont pilotés par un gestionnaire de fenêtres (Window Manager (WM)). Le WM gère d'abord les superpositions et les positions des différentes fenêtres afin de conserver une image correcte de l'écran. Il résout les problèmes de protection (une application ne peut pas dessiner dans la fenêtre d'une autre) pour exécuter différentes applications s'ignorant les unes des autres. Par contre, le contenu de la fenêtre est souvent à la charge de l'application.

Dans un environnement distribué, chaque fenêtre d'un système de fenêtrage peut être supportée par un processus séparé (**client**) [Gosling, 1985, X-Window, 1989]. Les clients s'exécutant en parallèle, invoquent le système de multi-fenêtrages pour manipuler les fenêtres et effectuer des opérations graphiques dans les fenêtres. Des problèmes de synchronisations apparaissent d'une part quand l'utilisateur manipule une fenêtre et qu'un client accède à cette fenêtre, et d'autre part, lorsque plusieurs applications accèdent en même temps au même écran chacune dans ses fenêtres. Pour résoudre les problèmes de synchronisation, les clients doivent passer par un unique **serveur** de fenêtres (le gestionnaire de fenêtres) pour accéder à l'écran. Dans un environnement distribué, les clients utilisent des messages pour dialoguer avec le serveur et la synchronisation provient de la sérialisation des messages des clients. Le serveur représente une partie du système de fenêtrage, il assure la gestion des fenêtres, l'accès à l'écran et les opérations graphiques.

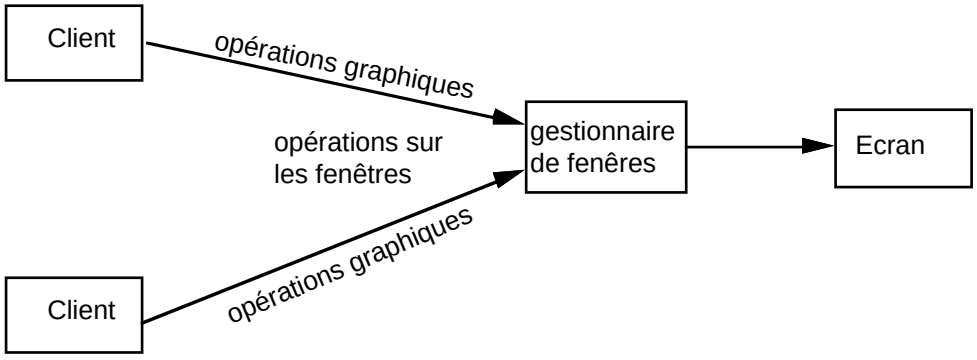


Figure 6.2.a: Architecture des systèmes de fenêtrage

Pour obtenir une interactivité suffisante, il est nécessaire de disposer d'un moyen de communication rapide entre les clients et le serveur.

Le modèle **client-serveur** dans un système de fenêtrage incorpore la notion de réseau de machines. Une application cliente n'a pas besoin de fonctionner sur le même type de système que celui qui supporte l'écran de visualisation (station de travail). Pendant que des applications locales s'exécutent sur la station de travail, d'autres applications distantes envoient par le réseau des demandes à l'écran de visualisation, et reçoivent les événements provenant de la souris et du clavier de la station.

Le programme qui contrôle chaque écran est appelé **serveur**. Il sert d'intermédiaire entre les clients, programmes des utilisateurs locaux ou distants, et les ressources du système locale: écran, clavier, souris….

a. Etude du système de fenêtrage X11

Après avoir décrit les principes d'un système de fenêtrage, nous étudions maintenant le système de fenêtrage X11 le plus employé actuellement et considéré comme un standard de fait (OSF, Unix International). Nous en déduirons les avantages et les inconvénients dans un environnement distribué vis à vis de l'utilisateur.

Après un bref historique, nous présentons l'architecture du système X11. Nous nous intéressons plus particulièrement aux aspects communications entre client et serveur, ainsi que l'utilisation du fenêtrage dans un réseau de machines.

Le système de fenêtrage X-Window, souvent dénommé X, a été développé dans le projet ATHENA du Massachussets Institut of Technology et par DEC avec la contribution d'autres compagnies dont IBM. La version la plus populaire est la version 10 release 4 de 1986. La version actuelle est X-11 release 4. X est un système de multi-fenêtrages pour des stations de travail avec écran bit-map. Le système X est basé sur le modèle client-serveur, il supporte actuellement les réseaux TCP/IP et DECnet. X supporte l'utilisation d'au moins un écran de visualisation. X a été conçu pour être facilement portable d'un type de machine à un autre.

X n'est pas une interface utilisateur, mais il propose une vaste étendue de gestionnaires d'interface utilisateur en laissant le choix de la politique de l'interface utilisateur aux développeurs d'applications. Cette particularité lui a permis de devenir un standard de fait mais actuellement l'industrie cherche un standard de plus haut niveau pour obtenir une consistance de l'interface utilisateur [OPEN LOOK, 1989a, OSF/Motif, 1989a].

X apparaît ainsi comme une plate-forme. Alors les programmes X sont eux aussi, portables mais c'est à l'utilisateur final de s'adapter à différentes politiques d'interface utilisateur. Jusqu'à ce qu'un standard d'interface apparaisse sur le marché, les développeurs X doivent rester vigilants dans l'écriture de leurs programmes. Leurs programmes doivent donc fonctionner avec différents gestionnaires des fenêtres (Window Manager) et conventions d'interface utilisateur.

Nous présentons maintenant l'architecture de ce système de fenêtrage. Le système X possède un serveur d'affichage par station bitmap. Il est le plus indépendant possible du type de matériel de la station de travail, mais il inclut une partie spécifique pour gérer l'écran. Les applications font appel au serveur par l'intermédiaire d'une bibliothèque Xlib.

Le serveur X exécute les fonctions suivantes:
- permettre l'accès à l'écran de visualisation par des clients,
- interpréter les messages des clients provenant du réseau,
- passer les événements (souris, clavier) aux clients en leur envoyant des messages,
- maintenir des structures complexes, incluant fenêtres, curseurs, caractères, et des contextes graphiques comme des ressources partageables par les clients.

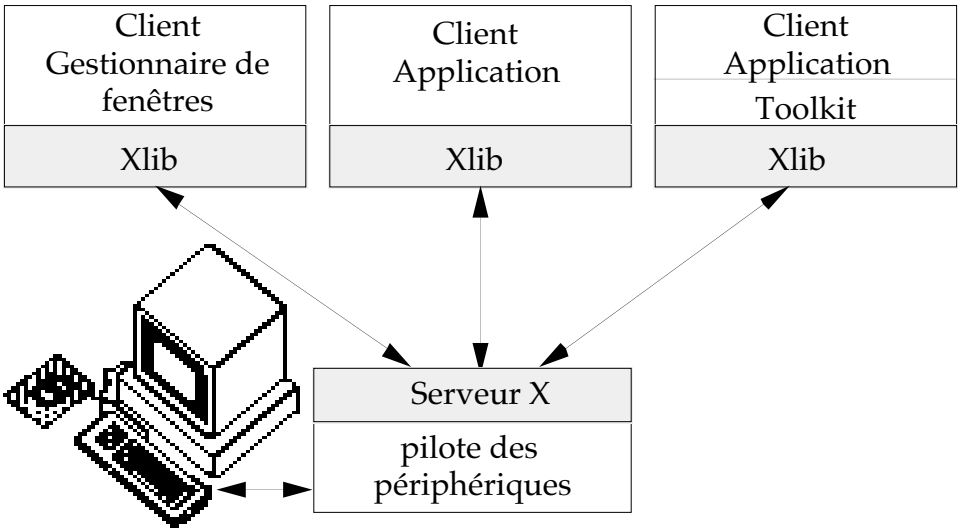


Figure 6.2.b: Architecture client/serveur X11

Gestionnaire de fenêtres et bibliothèque XLib

Pour que les clients ne soient pas dépendants d'une configuration particulière des fenêtres. La disposition des fenêtres à l'écran et les types d'interaction de l'utilisateur avec le système sont laissés à un programme séparé: le gestionnaire de fenêtres, un programme écrit à l'aide de librairie X; il a une autorité sur la gestion et le contrôle des fenêtres. Il permet de changer la taille des fenêtres et de les déplacer, de lancer une nouvelle application et de contrôler l'empilement des fenêtres. Le gestionnaire des fenêtres intercepte des événements (requêtes) des clients. Par exemple, l'application cliente doit attendre que la fenêtre soit visible à l'écran avant toute opération graphique.

La bibliothèque Xlib permet de gérer la connexion à un serveur (avec fonction de transcodage suivant l'architecture des machines), création de fenêtres, dessin graphique, réponse à des événements. Toutes les opérations sont orientées bitmap. Les applications spécifient les opérations graphiques en terme d'adresse de pixel à l'intérieur des fenêtres. Toute opération graphique d'une application ne peut intervenir que dans son espace de travail, c'est à dire dans ses propres fenêtres. Certaines applications sont écrites au dessus d'un toolkit (OSF/Motif, OPEN LOOK…) qui implémente un ensemble de primitives pour la gestion de l'interface utilisateur, comme les menus, les boutons.

Nous étudions maintenant la gestion des événements. Un événement est une action de l'utilisateur sur la souris ou sur le clavier (un périphérique). Les événements sont mis dans une file d'attente dans l'ordre d'arrivée. Un client X doit être préparé à répondre à n'importe quel événement asynchrone. On obtient une programmation par **boucle d'événements**, c'est l'utilisateur qui contrôle le programme et non le contraire.

Expliquons par un exemple simple le passage des événements entre le serveur et le client. Quand un utilisateur clique sur un bouton, le serveur envoie un événement au client correspondant qui va, par exemple, mettre en valeur (noircir) le bouton, pour montrer à l'utilisateur qu'il a cliqué dans le bouton. On voit à ce niveau, l'interaction directe du programme client sur l'interface utilisateur et le passage d'information. Cet exemple révèle déjà les **problèmes de synchronisation** entre les actions de l'utilisateur et leurs traitements par les applications et vice versa. Un traitement local de certaines actions de l'utilisateur nous semblerait mieux adapté.

Protocole X

Les appels des clients à une fonction de la bibliothèque Xlib sont traduits en des requêtes du protocole X. Le serveur exécute les opérations graphiques provenant des clients, aussitôt qu'il les reçoit.

Le protocole X permet le transport de l'information entre le serveur et Xlib. Il existe quatre types de paquets: requête, confirmation, événement et erreur, à chacun de ces types correspond un sens de transfert, un traitement et un stockage particulier de Xlib vers le serveur:

- Paquets requêtes
  Ces messages d'information transportent différentes ordres: dessiner une ligne, changer la couleur d'une cellule. Généralement une action sur l'écran nécessite une série de tels ordres donc un paquets. Certaines des routines de Xlib génèrent des requêtes. Une application fait un appel asynchrone à Xlib, qui construit une requête, la bufferise. Puis quand la file est pleine, envoie au serveur un paquet de requêtes, afin d'optimiser le transfert via le réseau. Dans le vocabulaire de X, envoyer un paquet de requêtes s'appelle "flusher" le buffer. Cette méthode, la plupart du temps, est transparente pour le programmeur de l'application. Elle optimise l'utilisation du réseau mais génère une désynchronisation entre l'application et le serveur et par conséquence avec l'utilisateur.
- Paquets confirmations
  Les requêtes ont, en général, besoin d'un paquet de confirmation, mais il existe des requêtes qui n'en ont pas besoin, comme dessiner une ligne.
- Paquets événements (du serveur vers Xlib)
  Les paquets événements sont des informations sur l'action d'un périphérique, conservées dans l'ordre de génération.
- Les paquets erreurs
  Les paquets erreurs sont des événements de Xlib particuliers, ils sont envoyés directement aux routines de gestion d'erreurs, quand la connexion est flushée. On obtient encore dans ce sens une désynchronisation entre le serveur et le client, ce qui complique la mise au point. Par exemple, le paquet d'erreur indiquant que l'on vient de dessiner dans une fenêtre inconnue ne parvient à l'application qu'après un temps assez long; pendant ce temps elle a pu faire d'autres choses et l'erreur ne correspond pas à l'endroit où l'on se trouve dans l'application.

L'application envoie des requêtes mises dans une file au niveau de Xlib, qui les envoie ensuite au serveur. Par contre dans l'autre sens, le serveur ne bufferise pas les événements, il les envoie immédiatement à la vitesse du réseau. Quand un événement arrive à Xlib provenant du serveur, Xlib envoie les requêtes en attente au serveur (il "flush"' la connexion). Les actions de l'utilisateur sont normalement les déclenchements, mais, pour forcer une synchronisation, Xlib ou l'application peut initier la transaction pour flusher la connexion. Cette méthode est utilisée quand l'application à besoin d'une confirmation ou d'information après l'envoi d'une requête.

Utilisation de X dans un environnement distribué

Le lecteur pourrait croire que ce système rend l'accès au réseau transparent pour les utilisateurs et fournit ainsi un **système d'exécution distribué**, permettant de faire coopérer différents types de machines. Mais en fait ce système n'est pas suffisant car si un utilisateur connecté sur sa station de travail (le serveur X) peut lancer d'autres clients sur d'autres machines, il est obligé de **connaître** le nom des machines ainsi que la liste des clients potentiels. Il doit connaître la **topologie** du réseau ainsi que la **localisation** des clients. Un tel système ne suffit donc à pas rendre transparent un système distribué. C'est ainsi que les terminaux X vendus depuis peu, sont utilisés pour être connectés à un site central. Une analogie apparait ainsi entre ces terminaux X et, il y a quelques années, les terminaux alphanumériques connectés à un site central. Cette utilisation se comprend car il est actuellement plus facile d'administrer un site central qu'un ensemble de machines hétérogènes sur un réseau.

b. Conclusion

Le système de fenêtrage X11 reflète en grande partie l'état de l'art dans ce domaine, mais surtout fait apparaître les problèmes encore cruciaux qu'il reste à résoudre. L'utilisation de l'architecture du système X (bibliothèque Xlib) apporte un avantage important pour développer et maintenir de telles applications graphiques puisque cette méthode permet un bon niveau d'abstraction du type de matériel et des systèmes d'exploitation sous-jacent. Il faut donc retenir du système de fenêtrage X les idées suivantes: l'utilisation d'un **réseau** de machines hétérogènes (système d'exploitation et architecture) pour permettre à des programmes d'accéder à un **écran graphique** distant de manière transparente par l'utilisation d'un **protocole** permettant le **transport** et la **gestion des informations** graphiques. Le système X est multi-fenêtres, mais il appartient aux applications de gérer leur contenu graphique.

Toutefois ce système souléve des problèmes importants qui restent à résoudre. Ces problèmes proviennent du **goulot d'étranglement** constitué par la communication client-serveur et des **synchronisations** entre les applications clientes et le serveur et surtout de la **transparence du réseau**.

Pour réduire les coût de communication, il faut que les clients manipulent des objets de haut niveau comme des cercles, droites, boutons, menus .. et non du bit-map. Ceci est le cas du système de fenêtrage NeWS (Network extendable Window System) [NeWS, 1986] basé sur le modèle client serveur, PostScript et conçu pour être portable. Le serveur News implémente une partie de l'interface utilisateur, des morceaux des programmes clients sont téléchargés et interprétés au niveau du serveur.

Le serveur ne doit pas seulement prendre les événements générer par les interactions de l'utilisateur et les envoyer passivement aux clients; mais surtout il doit pouvoir en **traiter** un nombre important **localement** ceci pour améliorer la vitesse de réaction du dialogue ("feed back") avec l'utilisateur.

La technique client-serveur est actuellement une des plus employées (X11 et News), sa conception reste **centralisée** tout en fonctionnant dans un environnement **distribué**. De ce fait, elle ne profite pas complètement de l'environnement réseau dans lequel elle s'exécute.

6.2.3. Répartition du fenêtrage du GSV en systèmes local et distant

L'analyse des avantages et des inconvénients des systèmes de fenêtrages actuels, nous a amenés à choisir une architecture de la plate-forme Gestion de Station de travail Virtuelle (GSV) à base de deux systèmes de fenêtrages: un local et un distant. Nous allons montrer que nous avons retenu les "bons" principes de X-Window tout en éliminant ses faiblesses.

L'avènement de stations de travail puissantes à bas prix incite à profiter de leur puissance de calcul pour améliorer l'interface utilisateur, mais l'utilisation de telles stations en réseau connectées à des serveurs de calcul doit être transparente pour les usagers qui ne doivent pas connaître l'emplacement physique des applications (§I.2.3.2 Structure d'accueil). Cette transparence du réseau n'est pas intégrée dans X. Pour cela, il nous faut gérer les problèmes de nommages d'application (client) dans un environnement distribué hétérogène.

La technique de client-serveur a montré sa souplesse d'utilisation dans un environnement distribué, nous allons baser notre système sur ce principe.

Toute application d'interaction utilisateur est basée sur une gestion d'événements associée à un protocole de communication.

Il doit y avoir indépendance entre application/système graphique et pourtant l'interface utilisateur doit permettre le couper/coller entre fenêtres d'une même application comme le couper/coller entre applications.

a. Coût des communications

Le problème de goulot d'étranglement au niveau du réseau provient du fait que beaucoup d'informations transitent entre l'interface graphique et l'application. Ce goulot d'étranglement entraîne aussi une désynchronisation entre l'utilisateur et l'application. Comment résoudre ses problèmes? Quoi que l'on fasse, il faut gérer l'interaction avec l'utilisateur: lorsqu'il clique sur un bouton une réponse graphique doit être immédiat.

Dans la plupart des cas, un ensemble d'interactions élémentaires ont pour but la création d'un événement global et seul cet événement de haut niveau intéresse l'application. De plus, la gestion du graphique et du contenu des fenêtres charge énormément le réseau de communication alors qu'on pourrait profiter de la puissance de la station de travail pour y exécuter l'application gérant le graphique. Cette application d'interaction utilisateur ne transmet aux programmes d'applications qu'un minimum d'informations condensées. Elle filtre les événements et envoie les événements de haut niveau aux applications: il s'agit d'un serveur intelligent.

La figure 6.2.c présente l'architecture de notre système de fenêtrage réparti qui prend en compte un système de fenêtrage local (système de fenêtrage et gestionnaire de fenêtres) et la plate-forme GSV assurant le transport d'évènements de haut niveau vers les programmes d'application. Notre approche reprend le principe client-serveur défini dans le système X-Window. Les clients sont des programmes d'application ne connaissant pas le graphique et le serveur est une application graphique basée sur un système de fenêtrage local.

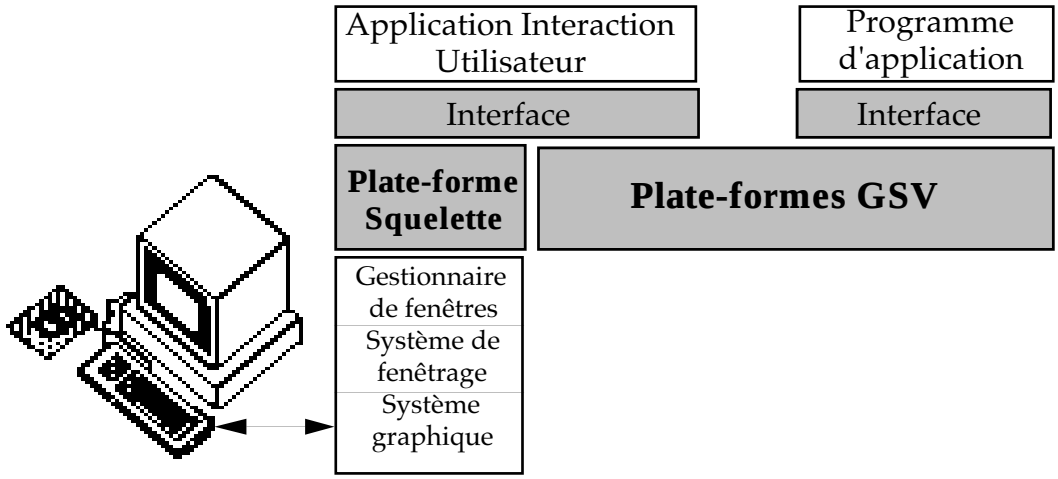


Figure 6.2.c: Architecture de notre système de fenêtrage réparti

Ces constatations nous ont amené à concevoir un double système de fenêtrage: un système local (graphisme, fenêtres, menus) et un système distant (services, programmes d'applications). Il existe le même principe de communication entre les deux systèmes de fenêtrage: un protocole, un ensemble de messages mais ces informations sont de plus haut niveau ce qui sépare encore plus l'application finale du graphisme. La séparation est telle que l'application finale ne sait même pas qu'il y a du graphisme, des fenêtres ou des menus!

b. Système de nommage

L'utilisateur final attend un certain nombre de services qui lui sont proposés sans qu'il se préoccupe de la localisation effective des programmes d'applications ni des machines de calculs [Coulouris and Dollimore, 1989]. Pour cela la plate-forme Gestion de Station de travail Virtuelle propose à l'utilisateur l'ensemble des noms des services accessibles. La plate-forme GSV interroge la plate-forme GUS pour connaitre les services et les applications qui les réalisent. La plate-forme GSV est responsable de l'exécution des programmes d'application dans l'environnement distribué. Le nommage dans un environnement distribué est un problème fondamental et sera développé dans la section (§6.4.3.b Le gérant des services).

c. Désynchronisation

Des désynchronisations entre nos deux systèmes de fenêtrages peuvent intervenir et le GSV doit gérer les synchronisations. En effet, les commutations de fenêtres sur le système de gestion de fenêtre local peuvent entraîner des commutations de session du système de gestion de fenêtre distant et les messages en cours de transfert doivent être routés vers les bonnes fenêtres sans que l'utilisateur ne s'aperçoive de rien. Cette simulation est effectuée au niveau local et la nouvelle fenêtre devient immédiatement active sans attendre de confirmation. Si des commandes arrivent pour les fenêtres en arrière plan, leur traitement peut être modifié en conséquence. En effet on ne peut pas interrompre l'utilisateur par une question venant d'une application en arrière plan. Lorsqu'il y aura confirmation de commutation, c'est le gestionnaire de fenêtre distant qui n'enverra peut-être plus de messages en attendant une réactivation.

Cette double gestion locale et distante doit être tolérante aux pannes car l'un des gestionnaires de fenêtres peut tomber un panne. Cela permet aussi au gestionnaire local de demander l'exécution d'application, de se déconnecter et de se reconnecter plus tard pour obtenir les résultats. (§6.4.2.e Terminaison de la couche session).

Nous allons montrer l'architecture en couches pour la conception du gestionnaire de station de travail virtuel.

6.3. Architecture en couches de la plate-forme GSV

En raison de la complexité de la gestion de l'hétérogénéité des systèmes d'exploitation et des contraintes des environnements distribués, nous nous sommes naturellement inspirés de l'architecture hiérarchisée. Notre architecture de la plate-forme Gestion de Station de travail Virtuelle est structurée en couches, dans un esprit semblable à celui du modèle de référence OSI de l'ISO du CCITT, car une telle architecture permet de spécifier les échanges entre les différentes composantes du système (entités). Une telle structuration est dans le domaine des réseaux hétérogènes, reconnue comme une des voies permettant de maîtriser des système de haute complexité [Diaz, 1989].

Nous examinons aussi les avantages du modèle OSI du point de vue génie logiciel.

6.3.1. Architecture en couches d'un système distribué

La normalisation de l'interconnexion de systèmes hétérogènes constitue un guide indéniable pour le développement d'un environnement logiciel dans une architecture distribuée. Cette **décomposition hiérarchique** permet de s'affranchir progressivement d'un ensemble de contraintes systèmes mais la conception d'applications n'est pas toujours totalement indépendante des caractéristiques des systèmes d'exploitation utilisés.

Le modèle OSI [Rose, 1990, Zimmerman, 1980] propose une décomposition fonctionnelle du système de communication en couches, chacune étant caractérisée par des fonctionnalités précises et les services qu'elle offre à la couche supérieure. Une entité de la couche n sur une machine dialogue avec une entité de la couche n d'une autre machine [Tanenbaum, 1981]. Les règles et les conventions utilisées pour le dialogue sont appelées protocole de la couche n. En réalité, les données ne sont pas directement transférées de la couche n d'une machine à la couche n d'une autre (sauf pour la couche la plus base de l'architecture). Chaque couche passe ses données et ses informations de contrôle à la couche immédiatement inférieure, jusqu'à la couche la plus basse du modéle. Une couche peut rajouter de l'information au message, ou bien réguler le flots de données, fragmenter le message pour le transmettre à la couche inférieure. Chaque protocole doit contenir suffisamment d'informations pour permettre l'implémentation de chaque couche. L'implémentation et la spécification des interfaces font partie de l'architecture. Il n'est pas nécessaire que les interfaces soient identiques sur toutes les machines, mais que chaque machine utilise correctement le protocole.

Les systèmes ouverts échangent des informations à travers un ou des systèmes intermédiaires. Ces derniers représentent le réseau de communication permettant d'interconnecter les machines de l'environnement distribué.

6.3.2. Architecture en couches et génie logiciel

L'architecture d'un atelier logiciel [XICH, 1988] peut être inspirée des modèles en couches. Elle consiste à identifier des niveaux de service dans un atelier logiciel et à associer une couche de "boîte à outils" à chacun de ses niveaux. Les outils d'une couche s'appuient alors sur ceux des couches inférieures. Les différentes boîtes à outils sont constituées d'outils de l'environnement de programmation, d'outils de l'environnement de projets, d'outils de l'environnement méthodologique. Ce type d'architecture répond aux critères des **ateliers ouverts, stables**. Cette technique de structuration en couches permet de partitionner en petits problèmes, un problème difficile à gérer.

Une architecture d'atelier logiciel répartie [Bourgeois, et al., 1988] doit reposer à la fois sur une architecture matérielle de type réseau et sur une architecture **logicielle en couches**.

Nous montrons la complémentarité d'une architecture en couches du GSV avec des concepts de génie logiciel nécessaire à une méthode de conception orientée objet [Meyer, 1988].

Dans le modèle OSI, une couche n est cliente de la couche immédiatement inférieure. Ce type d'architecture satisfait au principe de **protection**. Chaque couche communique avec un nombre restreint de couches; elle ne communique que par le biais des interfaces.

La notion d'**encapsulation** est inhérente au modèle OSI; ce masquage de l'information se retrouve dans les concepts objets.

Le concept de couche respecte le principe de **décomposabilité**: Une méthode de conception satisfait au principe de décomposabilité si elle aide à décomposer les systèmes complexes. Cette approche confére à la plate-forme GSV la **modularité**.

Au niveau d'une couche, nous utilisons une méthode satisfaisant au principe de **composabilité**: combiner des éléments de logiciel pour construire un nouveau système. Des changements de spécifications n'ont pour résultat que des changements limités dans chacune des couches de notre architecture: notre architecture du GSV satisfait donc au principe de **continuité.;**

6.3.3. Conclusion

L'architecture en couches reprend les principaux avantages des systèmes de fenêtrages par réseau que nous avons d'étudiés (protocole réseau, niveaux hiérarchiques). Les applications se situent sur la couche la plus externe du modèle et sont donc les plus indépendantes possible du système et de l'interface utilisateur.

Notre architecture du GSV inspirée du modèle OSI satisfait bien les principes de base d'une méthode de conception orientée objet. Mais cette technique de conception en couche a malheureusement un inconvénient: le **problème d'efficacité**. La superposition de couches et la communication pourraient entraver l'efficacité de notre architecture. Pour réduire les coûts de communication, les couches doivent limiter l'échange d'informations sur les objets manipulés et il faut privilégier les traitements locaux.

6.4. Architecture du GSV

L'environnement multi-sessions de la plate-forme Gestion de Station de travail Virtuel (GSV) induit une gestion complexe de l'exécution concurrente de plusieurs entités de protocole dans différentes couches. Une telle approche repose en particulier sur le concept de services et de protocoles.

> **Définitions**: Entité locale, Entité distante.
>
> Par convention, une entité est **locale** lorsqu'elle est liée à l'application d'interaction utilisateur, **distante** sinon.

L'entité d'une couche coté application d'interaction utilisateur (local) dialogue avec son entité homologue coté distant (Figure 6.4.a). Les deux cotés manipulent les mêmes données mais les traitements sont différents. Les communications entre couches de même niveau sont asymétriques. En effet, pour laisser l'initiative à l'utilisateur, l'application d'interaction utilisateur joue un rôle de maître au niveau de chaque couche (Cf. modèle client-serveur). L'atelier fonctionne dans un environnement distribué hétérogène; les incertitudes et perturbations dues au milieu de communication ou aux défaillances éventuelles des sites imposent la conception de protocoles robustes [Estraillier, 1986].

La communication horizontale entre les entités locale et distante est réalisée par envoi de messages. Les entités distantes dialoguent entre elles (verticale) par messages en utilisant les moyens de communication offerts par le module extension système (§3.2.1 Système - Répartition). Cette caractéristique facilite la distribution des entités distantes sur l'ensemble des sites supportant l'atelier.

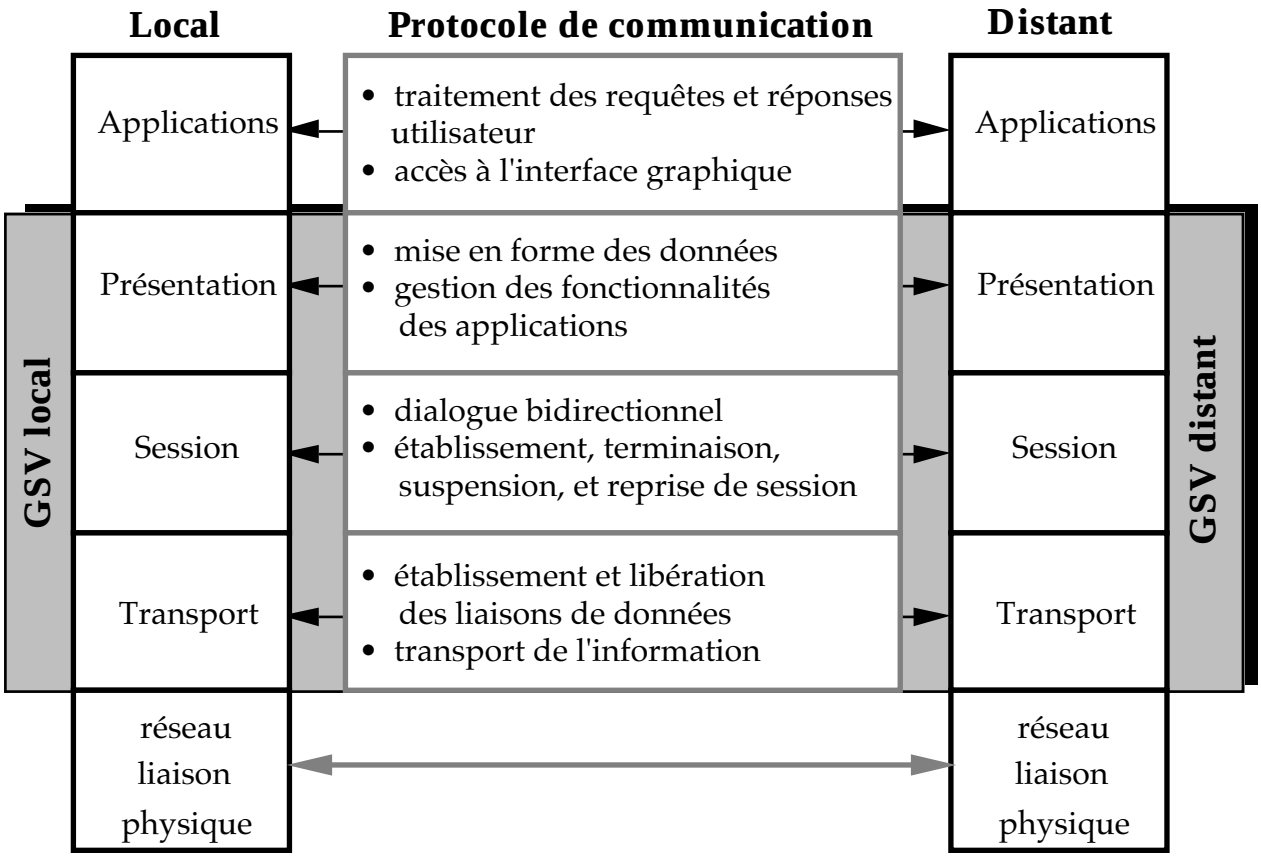


Figure 6.4.a: Architecture en couche de la plate-forme GSV

La couche **physique** a pour rôle de véhiculer l'information en une suite de bits sur le support de transmission. La couche **liaison** a pour objectif de gérer la liaison de données, et pour fonction de délimiter et de reconnaître les séquences de bits (trames) de la couche physique et de les maintenir en séquences. La couche **réseau** est principalement responsable de l'adressage des systèmes de communications, du routage des unités de données (paquets) et du contrôle de flux d'information à travers le réseau. Nous supposons ces couches déjà présentes dans notre environnement distribué (§3 Niveau extension système).

La couche **transport** de la plate-forme GSV intègre un mécanisme d'établissement et de libération de connexion comme dans le modèle OSI. Nous y avons ajouté la connexion dans un environnement distribué, l'identification de l'utilisateur et l'unicité de connexion. La connexion dans l'environnement distribué est assurée par différentes techniques rendue homogènes pour l'utilisateur.

La couche **session** est responsable de la gestion des sessions. Les sessions correspondent au travail d'un utilisateur sur un énoncé défini dans un formalisme. Pour la plate-forme GSV, la couche session de est donc responsable de la gestion multi-sessions et des formalismes. Elle intègre de plus un mécanisme de poursuite de session lors de la déconnexion de l'utilisateur.

La couche **présentation** dans le modèle OSI, est responsable de la mise en forme des données pour l'application. Dans notre architecture, elle rend indépendante la couche application, des données de la plate-forme GSV. Ainsi, la description des fonctionnalités des applications, des énoncés et des résultats est gérée par la couche présentation qui les trancrit pour la couche application. De plus, elle met en œuvre des algorithmes d'exécution des entités distantes de la couche application dans l'environnement distribué.

La couche **application** correspond, dans l'architecture fonctionnelle de l'atelier de spécification aux niveaux interface et application (Figure 1.1 Architecture fonctionnelle générale de l'atelier). L'entité locale correspond à l'application d'interaction utilisateur et les entités distantes aux programmes d'application. L'application d'interaction utilisateur est donc basée sur deux plate-formes (plate-forme squelette et plate-forme GSV), elle est responsable du routage des informations vers l'une ou l'autre des plate-formes pour traiter localement les données graphiques et diminuer le traffic sur la plate-forme GSV.

Nous proposons dans la suite, pour chacune des couches, une description succincte par rapport au modèle OSI. Nous pouvons partager l'architecture en deux: les couches physique à réseau qui concernent le transfert de l'information et les couches transport, session à application pour le traitement de l'information. Nous n'insistons que sur ces dernières en décrivant les services ainsi que des objets manipulés, leur initialisation, leur traitement et leur fermeture.

6.4.1. Couche transport

Dans le modèle OSI, cette couche est considérée comme une couche tampon entre les couches supérieures de traitements de l'information. Elle permet une connexion point à point et elle définit un ensemble d'adresses de transport. La couche transport est responsable de l'acheminement (en "duplex intégral" et sans erreurs) de blocs d'informations appelés messages. Les messages transmis contiennent des commandes interprétées par les couches supérieures. Ces messages sont de taille variable. Cette couche effectue un contrôle du flux et bufferisation. Elle résout les problèmes de synchronisation dûs à des délais d'acheminement différents de la couche réseau.

La couche transport de la plate-forme GSV est composé de deux sous-couches (Figure 6.4.b) :

- La sous-couche transport d'information
  Cette sous-couche offre des moyens de connexion/déconnexion pour mettre en communication l'utilisateur avec la structure d'accueil. Le transport de la plate-forme GSV est ainsi établi. Cette sous-couche résout les problèmes de nommage de machines dans le réseau (environnement distribué hétérogène).
- La sous-couche gérant des utilisateurs.
  Nous avons rajouté cette sous-couche au modèle OSI. Elle assure la sécurité d'accès d'une part au système d'exploitation et d'autre part à l'atelier en identifiant l'utilisateur dès l'ouverture de la couche transport.

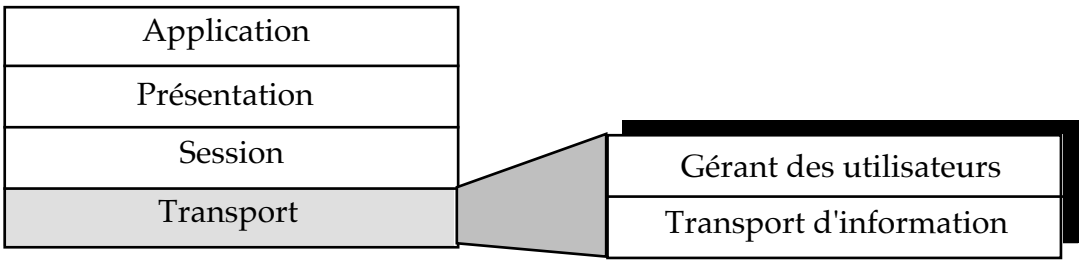


Figure 6.4.b: La couche transport

a. La sous-couche transport d'informations

Le protocole de transport réalise une connexion point à point entre les entités locale et distante. La sous-couche transport d'informations assure le transport de messages entre les entités locale et distante. Elle fournit aussi des procédures et des moyens fonctionnels pour établir et libérer des connexions de liaisons de données. Des entités distantes de la couche transport sont présentes sur plusieurs machines pour assurer une tolérance aux pannes. Elle assure ainsi une connexion dans un environnement distribué: l'entité locale se connecte à une des entités distantes disponible. Nous résolvons ainsi le problème de nommage de machines dans un environnement distribué. Le système distribué et la topologie du réseau sont donc rendus transparents aux utilisateurs.

Cette sous-couche garantit le séquencement correct des messages ainsi qu'une utilisation optimale de la connexion. Dans le cas d'erreurs irrécupérables (plus de réponse du récepteur…) la couche supérieure est informée. Cette sous-couche effectue un contrôle de flux entre l'émetteur et le récepteur. Un tel contrôle est nécessaire en raison du volume et de la durée du traitement graphique des résultats des services. Le contrôle de flux est uniquement dans cette couche pour éviter de l'effectuer pour chacune des sessions.

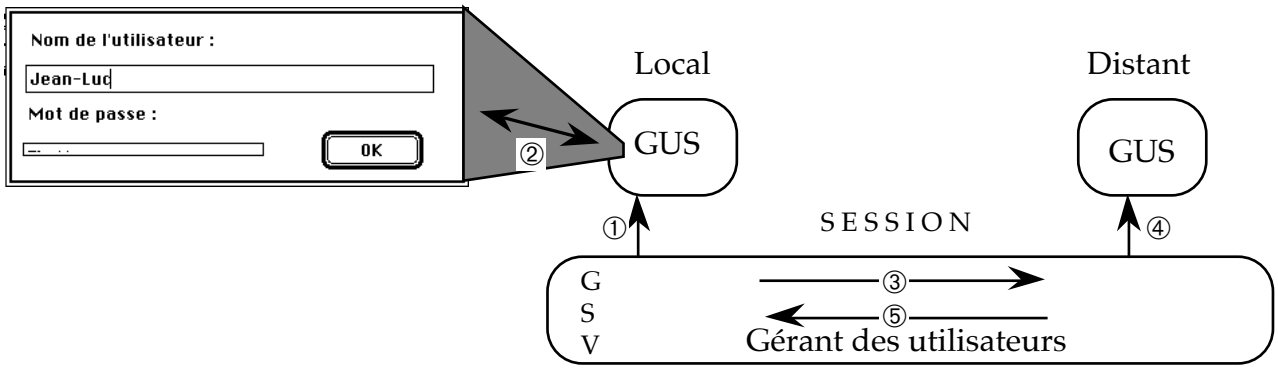
b. La sous-couche gérant des utilisateurs

La sous-couche gérant des utilisateurs stocke des informations concernant l'utilisateur pour les transmettre ultérieurement aux couches supérieures.

Lors de l'établissement d'une connexion de la couche transport, la sous-couche gérant des utilisateurs identifie l'utilisateur par son nom et son mot de passe. Il transmet ces informations à la plate-forme GUS qui lui confirme l'autorisation de connexion (§5.2 Gestion des utilisateurs).

Dans le cadre d'une réalisation de l'atelier dans un environnement distribué, plusieurs serveurs de communications peuvent effectuer la gestion des communications avec l'interface utilisateur. Il est alors nécessaire de veiller à la cohérence des contextes associés aux utilisateurs. Comme nous l'avons signalé (§5.2 Gestion des utilisateurs), le GSV doit interdire des connexions multiples d'un même utilisateur. Pour cela le gérant des utilisateurs informe le GUS de la connexion de l'utilisateur, en retour, le GUS peut interdire cette connexion s'il trouve une indication de connexion dans le contexte dynamique de l'utilisateur.

La réponse du GUS, lors de l'autorisation de connexion correspond à deux réponses dans un même message: le GUS à reconnu l'utilisateur et a vérifié qu'il peut se connecter.



① demande d'identification de l'utilisateur
② identification de l'utilisateur
③ transmission de l'identification et de la langue utilisée
④ élaboration du contexte dynamique de l'utilisateur
⑤ confirmation d'autorisation de connexion

Figure 6.4.c: Ouverture de communication

6.4.2. Couche session

Dans le modèle OSI, la couche session permet la gestion du dialogue. Elle fournit à la couche application un ensemble de mécanismes qui permette de définir les règles du dialogue (échange bidirectionnel…), des méthodes de synchronisation entre entités distantes, les services de connexion/déconnexion sans perte d'information et le transfert de données de contrôle. Une entité de la couche session se base sur les services rendus par la couche transport pour communiquer avec l'entité de session distante.

La couche session offre à l'utilisateur la possibilité de créer des énoncés dans les formalismes disponibles, à chaque énoncé est associé une session. Une session correspond à l'ouverture d'un énoncé dans un formalisme, la session est donc mono-formalisme. L'atelier de spécification étant multi-formalismes, la couche session propose à l'utilisateur les formalismes disponibles. Une session se traduit à l'utilisateur par l'ouverture d'un ensemble de fenêtres (lié au système fenêtrage local), lui permettant de visualiser et de saisir son énoncé.

La couche session est composée de deux sous-couches (Figure 6.4.d):

La sous-couche **gérant des formalismes** est responsable de la présentation de ces formalismes à l'utilisateur.

La sous-couche **multi-sessions** assure le travail simultané de l'utilisateur sur plusieurs énoncés grâce à un système de fenêtrage virtuel (§6.2.3 Répartition du système de fenêtrage du GSV en systèmes local et distant).

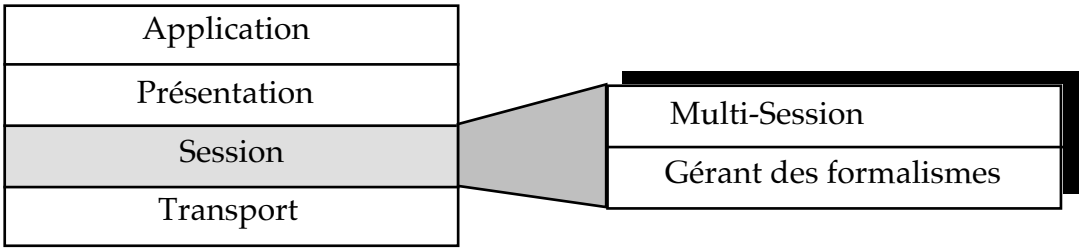


Figure 6.4.d: La couche session

Nous montrons dans ce paragraphe que la couche session joue un rôle majeur dans la manière de travailler de l'utilisateur en lui offrant la possibilité de travailler sur plusieurs énoncés à la fois. Cette couche est responsable du maintient des liens avec le système de fenêtrage locale et filtre ainsi les événements générer par les actions de l'utilisateurs sur les fenêtres de l'application d'interaction utilisateur. Cette caractéristique, si elle est observée par l'utilisateur comme une simple gestion de multi-fenêtrages, dissimule, au niveau de la couche session, des mécanismes de gestion de contextes, soumis à de multiples commutation. Lors de la déconnexion de l'utilisateur, la couche session assure la poursuite des sessions dans le cas ou des services sont actifs et un mécanisme d'enquête quand l'utilisateur se reconnecte.

Nous présentons un scénario de dialogue de cette couche détaillant l'ouverture de session, la gestion de sessions multiples, la fermeture et la terminaison de la couche session.

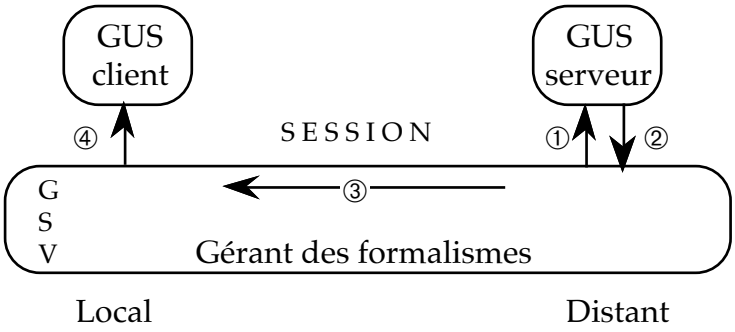
a. Initialisation de la couche session

L'application d'interaction utilisateur peut fonctionner en autonome, pour cela nous avons des copies locales de cette application. Les évolutions de l'application d'interaction utilisateur et de la structure d'accueil se font en parallèle et peuvent amener des incompatibilités avec d'anciennes versions. Pour augmenter la robustesse de l'atelier de spécifications, nous incluons dans les fonctionnalités de son architecture un protocole chargé de contrôler l'adéquation des différentes versions de l'application d'interaction utilisateur et du reste de l'atelier. Ce protocole détecte les versions compatibles des réalisations de l'application d'interaction utilisateur et de la structure d'accueil. Nous montrerons dans la partie III (§5.4.4 Les couches session, présentation) les avantages pour les utilisateurs de l'atelier d'avoir intégré ce protocole.

A l'initialisation, la couche session hérite (de la couche transport) du nom de l'utilisateur. Elle associe un contexte dynamique à l'utilisateur (§5.2 Gestion des utilisateurs). Ce contexte est particularisé dans la mesure où l'administrateur de l'atelier, lorsqu'il ajoute un utilisateur, définit ses droits d'accès aux données [Belkhatir, 1986] et aux services.

Pour compléter les informations sur le statut de l'utilisateur, la couche session détermine la langue utilisée (anglais, français…) par un appel au GUS-client. Cette information va compléter le contexte dynamique de l'utilisateur.

Après l'initialisation de la couche session, l'utilisateur va demander de travailler sur un énoncé dans un formalisme connu de l'atelier. La sous-couche gérant des formalismes anticipe l'ouverture de la session en construisant la liste des formalismes (Figure 6.4.e). L'entité distante de la couche session anticipe la demande et transmet, à la sous-couche gérant des formalismes local, la liste des formalismes disponibles pour l'utilisateur . Du côté distant cette liste est construite (①) par appel à la plate-forme GUS (GUS-serveur) qui gére l'ensemble des formalismes disponibles dans l'atelier (②). La sous-couche gérant des formalismes (③) a permis le transfert de la liste des formalismes du GUS-serveur au GUS-client (§5.10 Architecture du GUS dans un environnement distribué). Maintenant, la liste des formalismes disponibles pour l'utilisateur est accéssible du côté locale (④), la plate-forme GSV n'a joué qu'un rôle de transfert de données entre le GUS-serveur et le GUS-client.



① demande les formalismes disponibles pour l'utilisateur
② construit la liste des formalismes
③ transmission de la liste des formalismes
④ mise à jour du GUS client avec la liste des formalismes

Figure 6.4.e: Anticipation de l'ouverture de session

b. Ouverture de session

Du coté local, l'ouverture de session peut être réalisée de deux manières: soit en créant un nouvel énoncé, dans ce cas il faut choisir le formalisme de l'énoncé, soit en ouvrant un fichier énoncé existant, dans ce cas le nom du formalisme est contenu dans l'énoncé (§6.4.3.d Le langage LDE).

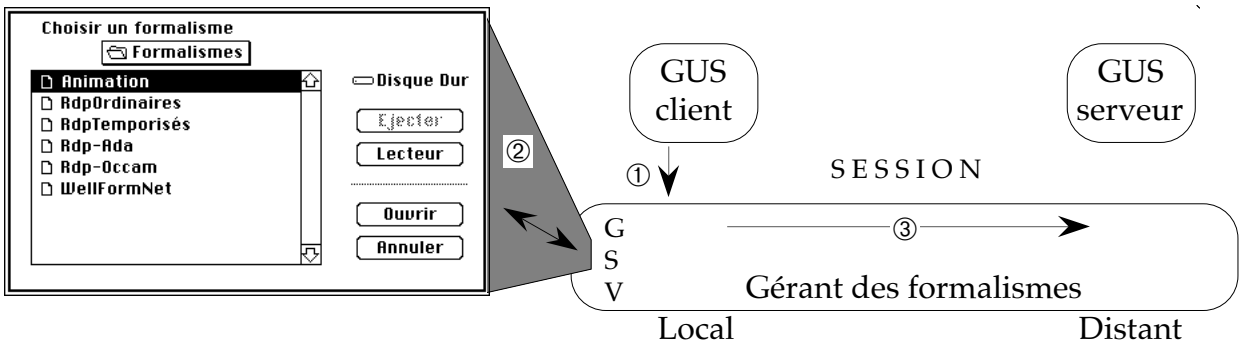
> **Définition**: fichier énoncé
>
> Un **fichier énoncé** est un fichier local qui contient à la fois le nom du formalisme, l'énoncé et les particularités esthétiques des objets de l'énoncé (§5.3.4 Modification esthétique d'un formalisme).
>
> Ce fichier est stocké dans le système de fichier local pour permettre à l'application d'interaction utilisateur de travailler de manière autonome.

Le protocole mis en œuvre par le gérant des formalismes ne transmet à l'entité distante que le message d'**ouverture de session** pour les événements de création d'un énoncé ou d'ouverture d'un fichier.

Lors de la création d'un nouvel énoncé, le gérant des formalismes propose à l'utilisateur les formalismes (①) connus du GUS client (Figure 6.4.f).

Dans le cas où la plate-forme GUS gère les modifications esthétiques des formalismes pour chaque utilisateur (§5.3.4 Modification esthétique d'un formalisme), la sous-couche gérant des formalismes proposera à l'utilisateur ses propres formalismes (①) (avec leurs modifications personnelles).

L'utilisateur ayant choisi un formalisme (②), la sous-couche gérant des formalismes transmet un message pour informer l'entité distante du choix de l'utilisateur afin de recevoir la description du formalisme (③).



① demande la liste des formalismes
② choix du formalisme par l'utilisateur
ou le formalisme est contenu dans l'énoncé ouvert.
③ transmission du choix du formalisme

Figure 6.4.f: Choix d'un formalisme

Pour permettre de travailler en autonome, des formalismes sont stockés localement (5.11); la sous-couche gérant des formalismes vérifie lors de l'ouverture d'un fichier énoncé, la validité du formalisme de l'énoncé par rapport à la liste des formalismes présent dans le GUS-client. Deux cas d'invalidité peuvent se présenter:

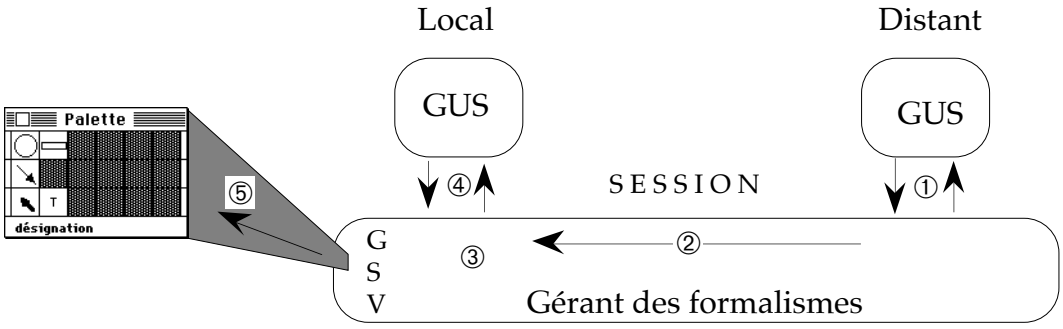
- soit le formalisme ne fait pas partie de la liste des formalismes précédemment reçue du GUS-serveur,
- soit la version du formalisme contenue dans le fichier énoncé est incorrecte. Lors de la mise au point d'un formalisme sa description évolue, il faut garantir que les utilisateurs travaillent avec la dernière version du formalisme du GUS-serveur. Ce problème est résolu par l'association d'un numéro de version à chaque formalisme (§5.3 Gestion des formalismes). Ce numéro de version garantit que l'on n'employera pas une ancienne version.

Dans les deux cas, la couche session interdit tout travail de l'utilisateur sur cet énoncé. Des services d'administrations permetterons à l'utilisateur de rectifier son énoncé pour qu'il puisse être analysé.

Afin de construire un énoncé, l'utilisateur crée des instances des **classes d'objets du formalisme**, saisit des valeurs des **champs** de chaque objet etc... Pour que l'utilisateur puisse créer graphiquement son énoncé l'entité locale de la sous-couche gérant des formalismes présente une **palette** des classes du formalisme. Pour chacune des classes, la sous-couche gérant des formalismes associe le nom de sa forme aux classes de forme graphique correspondantes. On obtient ainsi une représentation graphique de toutes ces classes.

La structure de la palette est construite à partir de la description LDF du formalisme du GUS serveur complété par héritage d'une partie graphique du GUS client (§5.10 Architecture du GUS dans un environnement distribué). L'héritage de la partie graphique consiste à associer à la forme de chaque objet définie dans le langage LDF (Figure 5.3.b: Les classes du formalisme réseaux de Petri), sa classe correspondante. La classe correspondante contiennent la méthode pour dessiner l'objet. La présentation graphique de cette palette à l'utilisateur est effectuée par l'éditeur de graphe (§8.2.1 L'éditeur de graphes). La couche session du GSV est donc capable d'analyser le langage LDF qui décrit les formalismes.

La palette est composé de plusieurs ligne. La première ligne de la palette contient les classes de nœuds et la deuxième celles des connecteurs. La dernière ligne contient des outils de base de manipulation graphique.



① recherche de la description du formalisme
② transmission de la description du formalisme
③ interprétation du formalisme
④ réception de la description locale du formalisme
⑤ présentation de la palette des classes du formalisme

Figure 6.4.g: Réception du formalisme

Après réception de la description locale du formalisme, la sous-couche gérant des formalismes locale dispose detoutes les informations nécessaires pour réaliser effectivement l'ouverture de session.

Du coté local, cette ouverture de session correspond à la création d'une fenêtre graphique. Le nom de la session (et le nom de la fenêtre) provient soit du nom du fichier énoncé, soit d'un nom attribué par défaut dans le cas d'un nouvel énoncé.

Pour assurer une cohérence entre les fichiers d'énoncés et les résultats, la couche session estampille l'énoncé par la date (§5.2 Gestion des utilisateurs). Cette date est contenue dans le fichier énoncé. Nous préciserons ultérieurement le calcul de cette estampille (§7.4.1 Le gestionnaire de graphes).

La sous-couche gérant des formalismes locale peut à présent envoyer un message d'ouverture de session à l'entité distante en lui indiquant: le nom de l'énoncé, son formalisme et la date.

Du coté distant, l'ouverture de session entraîne une enquête sur l'état des sessions en cours et du fonctionnement des serveurs de fichiers distants. Afin d'assurer une forte disponibilité, nous avons introduit des serveurs de secours sur la gestion des données des utilisateurs (§3.2.1 Système - Répartition) et un mécanisme de gestion des multiples copies des données des usagers. Les mises à jour et les consultations deviennent ainsi possibles à partir de n'importe quel site. Lors d'une ouverture de session d'un usager, l'enquête diffère selon que l'utilisateur est privilégié (accès à un serveur de secours) ou non. La figure 6.4.h présente les différents tests réalisé lors de cette enquête.

La principale difficulté provient du fait qu'un utilisateur privilégié peut avoir demandé des exécutions à un moment où le serveur principal était en panne. Si sa demande de traitement n'est pas terminée, les données sont en cours de sauvegarde sur le serveur de secours. Lorsqu'il se connecte, il faut le remettre en communication avec son serveur de secours même si le serveur principal est de nouveau en fonction.

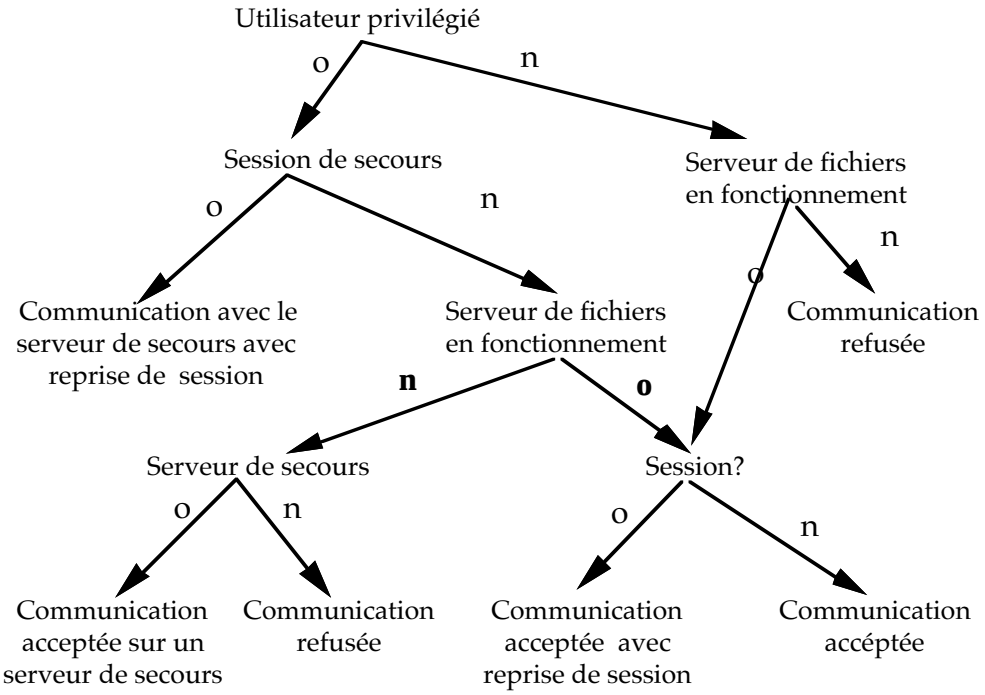


Figure 6.4.h: Enquête d'ouverture de session

Définition: Session

Une **session** correspond pour l'utilisateur à la préparation pour l'exécution d'un service. Elle correspond à l'ouverture d'un ensemble de fenêtre associées à un fichier énoncé sur poste de travail de l'utilisateur.

L'ouverture de session réalisée, l'usager va pouvoir commencer à travailler sur son énoncé. Cette ouverture de session va entraîner une ouverture de la couche présentation qui offrira à l'utilisateur l'ensemble des services accéssibles.`

Toutefois lorsque l'utilisateur travaille en autonome sur un ou plusieurs énoncés et qu'il décide de se connecter, une procédure automatique d'ouverture de session est effectuée pour chacune de ses sessions. La sous-couche gérant des formalismes autorise ainsi l'ouverture de plusieurs sessions. Nous présentons maintenant la gestion multi-sessions.

c. Multi-fenêtres et multi-sessions

La sous-couche multi-sessions est responsable du multiplexage des différentes sessions, elle permet d'activer plusieurs entités de présentation distantes. Cette sous-couche multi-sessions gère au niveau de chaque utilisateur, les contextes d'exécution qui commutent en fonction des demandes de traitement. Cette notion de multi-sessions, permet d'activer simultanément plusieurs services, ce qui induit l'exécution concurrente de plusieurs entités de protocole dans différentes couches. Cette caractéristique, si elle est observée par l'utilisateur comme une simple gestion de multi-fenêtrages, dissimule, au niveau de la couche session, des mécanismes de gestion de contextes, soumis à de multiples commutations. Elle intégre les contraintes d'un environnement réparti, et, en particulier, celles liées à la communication d'informations et au stockage distant de données.

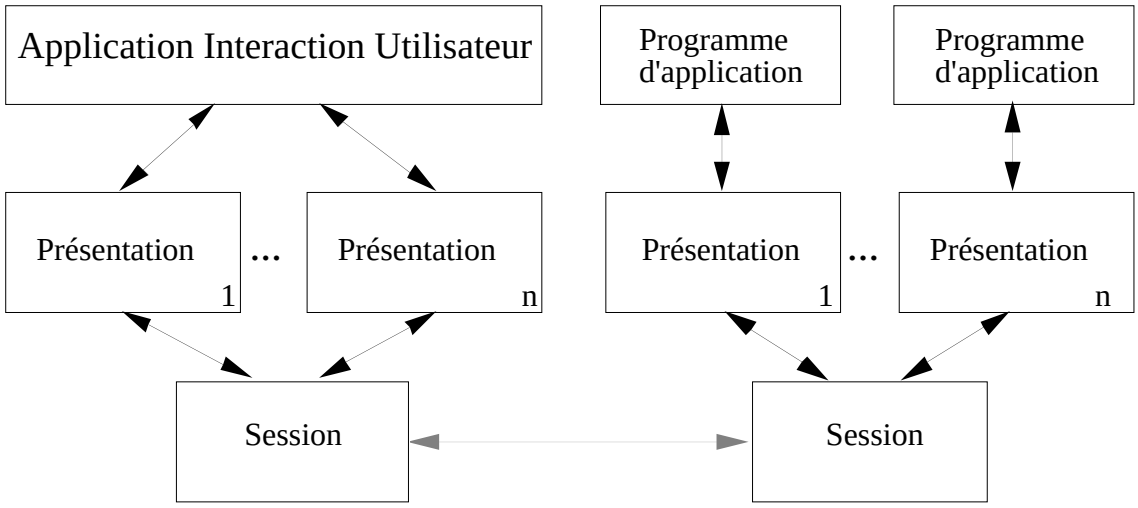


Figure 6.4.i: Les entités activées lors de la multi-sessions

Nous montrons les avantages de l'utilisation du double système de fenêtrage (§6.2.3 Répartition du systèmes de fenêtrage du GSV en systèmes local et distant) (le multi-sessions et le système de fenêtrage local). Ce système empêche les messages graphiques de transiter vers les entités distantes. Nous expliquons la gestion locale du graphisme et les influences des commutations de fenêtres sur les commutations de sessions des entités distantes.

Affichage de fenêtres associées à une session

La figure 6.4.j montre, le déplacement d'une fenêtre A sur une fenêtre B. Après le déplacement, la partie hachurée C doit être redessinée.

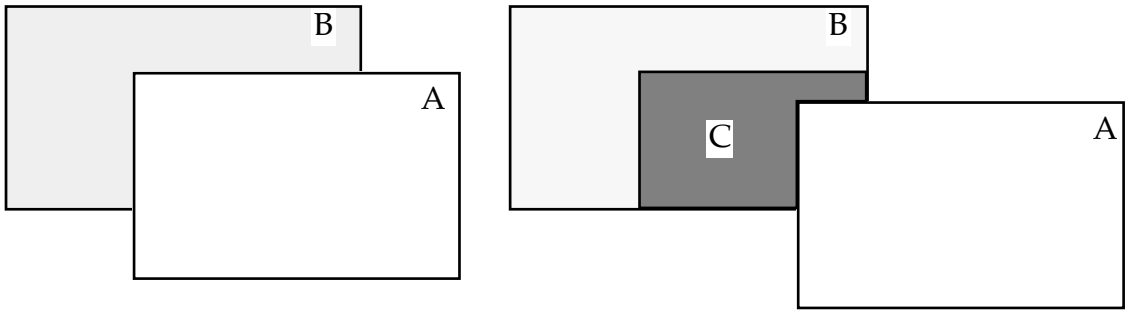


Figure 6.4.j: Déplacement de fenêtres

Le dessin est à la charge de l'application gérant la fenêtre B. Pour cela il existe deux méthodes: soit l'application connaît la description bit-map de toute la fenêtre B, soit elle est capable de la recalculer. La première méthode est très rapide mais consomme beaucoup de place mémoire: en effet il faut une information pour chaque point. Si le dessin est en noir et blanc, un bit d'information suffit mais si il s'agit d'un dessin couleur, chaque point de l'écran peut occuper 32 bits ! Ceci explique qu'il peut être plus aisé de redessiner la partie C. Les "toolbox" graphiques offrent des possibilités de dessin dans des zones de "clip" ici C: c'est à dire que le dessin ne débordera pas de la zone C.

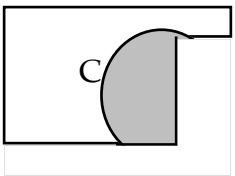


Figure 6.4.k: Zone de clip

La seconde méthode est économique en place mémoire mais demande aux applications de recalculer le dessin des objets.

Dans un système tel que X-Window, la partie à redessiner C, étant à la charge de l'application, les informations de dessin vont circuler sur le réseau entre le client et le serveur. Le temps de rafraîchissement est donc fonction de la charge du réseau et du nombre d'éléments à redessiner.

Dans notre système à base de double système de fenêtrage, le dessin est entièrement accomplit par l'application d'interaction utilisateur dans le système de fenêtrage local donc indépendamment des programmes d'application qui travaillent sur les énoncés. Ceci confère à l'utilisateur une plus grande impression d'interactivité et une indépendance des programmes d'application vis à vis de la gestion des fenêtres.

Fenêtrage et commutations de sessions

Les gestionnaires de fenêtres offrent la possibilité de travail dans une seule fenêtre à un instant donné: la **fenêtre active**. Toutes les interactions de l'utilisateur (clavier, souris) ont lieu dans la fenêtre active et le changement de fenêtre active se fait soit en pointant dans la fenêtre soit en positionnant le pointeur sur la fenêtre.

Nous étudions l'influence du changement de fenêtre active sur la commutation des sessions distantes. Une telle commutation est illustrée par la figure 6.4.l où la fenêtre B devient active, il y a donc une suspension de la session A et une reprise de la session B.

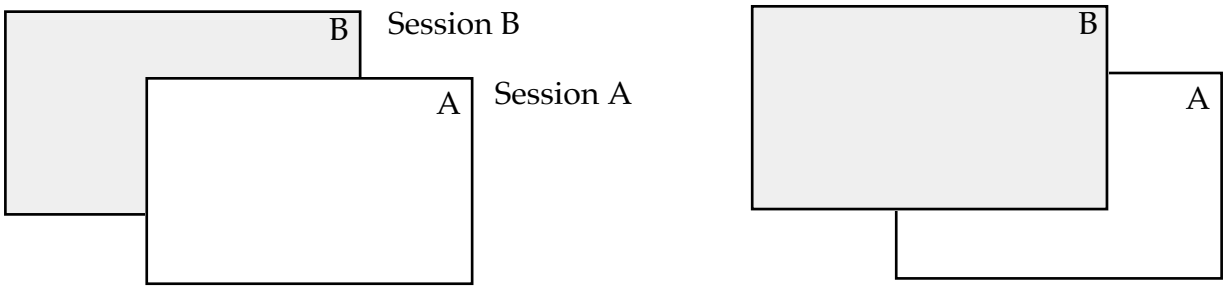


Figure 6.4.l: Commutation de session graphique

Les programmes d'application dans l'atelier ne connaissent pas la notion de fenêtre, pourtant dans le cas ci-dessus, le changement de fenêtre active va effectuer une commutation de session.

Deux solutions sont envisageables pour travailler avec plusieurs sessions: la première consiste à utiliser un multiplexage des données, la deuxième consiste en un traitement exclusif d'une seule session à un instant donné.

Le **multiplexage** consiste à permettre aux deux sessions de se dérouler en parallèle et de recevoir des messages des entités distantes (des résultats) dans chacune des fenêtres. Pour mettre en œuvre un tel mécanisme, il suffit d'inclure dans les messages, une identification de la session. Lors de l'ouverture de session, la sous-couche multi-sessions devra négocier le paramètre d'identification. L'avantage du multiplexage est lié à sa simplicité de réalisation mais son principal inconvénient est de pouvoir ralentir les traitements de premier plan. En effet, si une session en arrière plan demande d'effectuer beaucoup de dessins, l'interface utilisateur de la fenêtre active peut s'en trouver pénalisée.

Le traitement exclusif d'une seule session à la fois se concrétiser par des ordres de commutation de session (suspension et reprise de session). On obtient une meilleure interactivité à condition de résoudre deux problèmes: la resynchronisation et l'avertissement de l'usager.

La **resynchronisation** est un problème qui survient au moment où l'utilisateur demande la commutation de session en sélectionnant une fenêtre en arrière plan. Une suspension de session intervient pour la session de premier plan et une reprise de session pour l'autre fenêtre. Des messages sont envoyés à l'entité distante pour l'en informer mais il se peut qu'un ensemble de messages pour la fenêtre de premier plan soit en transit sur le réseau de communication. Le temps que l'entité distante stoppe le flux de messages de cette session et accuse réception de la suspension et de la reprise, il faut faire croire à l'utilisateur que la session a changé (en changeant la fenêtre de premier plan).

Le problème **d'avertissement** est le suivant: lorsqu'un programme d'application associé à une session en arrière plan effectue des calculs, il est important de prévenir l'utilisateur lorsque les calculs sont finis ou lorsqu'il y a besoin d'une information complémentaire pour poursuivre le traitement. Or, comme la session n'est pas au premier plan, on ne doit pas trop perturber la session de premier plan. Ces messages d'alerte peuvent être réalisés par des messages prioritaires indiquant à l'utilisateur (de manière discrète et paramétrable) qu'il se passe quelque chose dans une session en arrière plan et qu'il lui faudra aller voir lorsqu'il le voudra.

Multi-fenêtres et mono-session

Mais le changement de fenêtre ne provoque pas toujours de commutation de session comme nous allons le montrer. Dans ces cas, seul le gestionnaire de fenêtre local intervient et traite le changement de fenêtre: aucun message ne transite vers l'entité de session distante.

Lorsque l'utilisateur conçoit ou analyse son énoncé, l'application d'interaction utilisateur lui permet d'avoir plusieurs **vues** donc plusieurs fenêtres sur le même énoncé et travailler dans chacune d'elle (partie III §7.2 Mode autonome de Macao). On a donc un concept d'ensemble de fenêtres isomorphes dont l'une quelconque peut être choisie comme représentant tout l'ensemble. Ce concept de vue est entièrement traitée localement et la commutation de vue est donc une opération locale.

De même la conception **hiérarchique graphique** de modèles consiste à dessiner un modèle de haut niveau et à raffiner les objets du formalisme en créant des sous-modèles eux-même pouvant être raffinés. Chacun des sous-modèles est représenté dans une fenêtre indépendante et la commutation de fenêtres n'a aucune répercussion sur le gestionnaire de fenêtres distant étant donné que les applications distantes ne connaissent pas la notion de fenêtre. On obtient alors le concept d'ensemble de **fenêtres dépendantes**.

Lors de l'analyse de son énoncé, l'utilisateur fait appel à différents services. Certains de ces services peuvent donner les résultats textuels (§5.4.3 Représentation des résultats) affichables dans une fenêtre appropriée. Cette fenêtre lorsqu'elle est mise en premier plan n'occasionne pas non plus de commutation de session. Certains services vont rendre des ensembles de résultats et l'utilisateur peut désirer voir successivement ces résultats. Une fenêtre spécialisée, avec des boutons (suivant, prédécesseur) permet la sélection successive des différents résultats. Naturellement, l'utilisation de cette fenêtre n'entraîne pas de commutation de session. Les services de transformation (§5.4.2 Les types de traitements d'un service), créent de nouveaux énoncés dans une fenêtre résultat. Tant que l'énoncé-résultat n'est pas enregistré comme un fichier-énoncé, la commutation de fenêtres entre l'énoncé initial et l'énoncé résultat n'entraîne pas de commutation de session.

Comme nous venons de le voir, parmi les commutations de fenêtres possibles sur l'interface utilisateur, peu d'entre elles occasionnent de réelles commutations de session et la plupart sont donc traitées localement ce qui rend plus interactive l'interface utilisateur et justifie l'utilisation d'un gestionnaire de fenêtre local puissant. Nous avons montré qu'aucune donnée graphique ne transite vers les entités distantes.

Le flux de messages transmis entre les couches session et transport a été réduit ainsi au minimum: seuls les messages correspondant à des événements utilisateurs de haut niveau transitent pas le réseau de communication. Notre décomposition du GSV entre entités locales et distantes permet donc d'utiliser l'application d'interaction utilisateur au travers d'un réseau à faible débit (en particulier le réseau commuté par l'intermédiaire de modems). Toutes ces caractéristiques du double système de fenêtrage contribuent à privilégier l'interface utilisateur (Objectif 9).

d. Fermeture de la couche session

Lors de la fin de connexion, le contexte qui est associé à l'utilisateur est géré par l'entité distante de la couche session jusqu'à ce que tous les services qu'il ait demandés soient terminés. Un tel mécanisme, s'il s'avère agréable à l'utilisateur et s'il contribue à la performance du système, n'en demeure pas moins compliqué à gérer dans un environnement distribué, notamment en cas de défaillance ou si l'utilisateur se reconnecte intempestivement.

Une des fonctionnalités offertes par l'entité distante de la couche session est de permettre l'**exécution** des services **en arrière plan** (§5.2 Gestion des utilisateurs). Il est donc nécessaire de prendre en compte les résultats sans les retransmettre à l'utilisateur. Un mécanisme analogue est nécessaire lors d'un incident puisque certains services, continuant à s'exécuter, il faut éviter de répercuter les résultats à l'utilisateur qui est dans l'impossibilité de les traiter. Lors d'une prochaine connexion, si les services sont toujours actifs, l'entité distante de la couche session indiquera à l'utilisateur que des anciennes sessions sont toujours actives. L'utilisateur pourra ainsi "reprendre" des anciennes sessions.

L'utilisateur peut décider de terminer une session à tout moment. Pour cela l'application d'interaction utilisateur lui offre plusieurs possibilités: soit par un menu, soit en fermant des fenêtres. La fermeture de session, lors de la fermeture de fenêtre pose des problèmes d'options offertes aux utilisateurs.

En effet, si il n'y a qu'une seule fenêtre associée à une session, c'est au moment où l'utilisateur demande la fermeture de cette fenêtre que doit être effectuée la fermeture de session. Dans le cas où aucun service n'est en cours, il faut simplement fermer la session et fermer le fichier-énoncé en enregistrant les éventuelles modifications. Si un service est en cours, il faut demander à l'utilisateur s'il veut arrêter l'exécution du programme d'application ou si il veut que l'exécution se poursuive pour récupérer le résultat lors d'une prochaine connexion.

Dans le cas où un session est associée à plusieurs fenêtres dépendantes, une méthode consiste à prendre la première fenêtre ouverte comme fenêtre principale. La fermeture de cette fenêtre entraîne la fermeture de la session. Une autre méthode consiste à fermer la session lorsque toutes les fenêtres associées son fermées. Dans ce cas, c'est la fermeture de la dernière fenêtre qui entraîne la fermeture de session (comme dans le cas d'une seule fenêtre).

La fermeture de la couche session génère une demande de fermeture de la couche présentation en lui indiquant le mode de fermeture demandé (avec ou sans poursuite du service associé à la session).

e. Terminaison de la couche session

La terminaison de la couche session intervient en fin d'exécution de l'application d'interaction utilisateur. Cette terminaison peut être volontaire ou accidentelle. Lors d'une terminaison volontaire, l'arrêt de la couche session intervient après fermeture une à une de chacune des sessions et après accusé de réception de fermeture. Lors d'une terminaison accidentelle, tout s'arrête immédiatement et c'est l'entité distante de la couche session qui termine proprement chacune des sessions. En cas de rupture de la couche transport, la couche session distante se comporte comme s'il y avait eu terminaison accidentelle et l'application d'interaction utilisateur repasse en mode autonome (sans connexion) en fermant les sessions, en supprimant les résultats des questions en cours mais sans fermer les fenêtres. Celles-ci seront fermées par l'utilisateur quand il aura fini de les consulter.

6.4.3. Couche présentation

Dans le modèle OSI, l'objectif de la couche présentation concerne principalement l'homogénéisation des syntaxes utilisées pour véhiculer les informations de la couche application. Elle permet aux différentes applications d'échanger des données qui peuvent être codées dans des syntaxes différentes, la couche présentation effectuant les transformations (conversions) nécessaires. La couche présentation transmet de façon transparente les requêtes de l'application à la couche session. En effet, elle assure un protocole de terminal virtuel et elle est responsable de la présentation des données échangés par les applications. Elle définit complètement les manipulations des informations et connaît les fonctionnalités des applications. En résumé, la c ouche présentation s'intéresse à la syntaxe alors que la couche application se chargera de la sémantique.

La couche présentation joue un rôle important dans notre environnement hétérogène. C'est un intermédiaire indispensable pour une compréhension commune de la syntaxe des données (énoncés et résultats) qui sont transportés sur le réseau. La couche présentation du GSV procure un langage syntaxiquement commun à l'ensemble des utilisateurs connectés. La couche présentation de la plate-forme GSV est composée de trois sous-couches:

- Le **gérant des services** gère les services accessibles à l'utilisateur et permet l'exécution des applications (services).
- Le **gérant des dialogues** est responsable de l'arbre de questions associé aux applications. Il indique à l'utilisateur les fonctionnalités du programme d'application. Il traite les questions de l'utilisateur, pour les transmettre à l'application.
- L'**interface dynamique** de présentation de données, rend indépendante l'application de la représentation par l'atelier des données (Langage de Description d'Énoncés (LDE), de Résultats (LDR)).

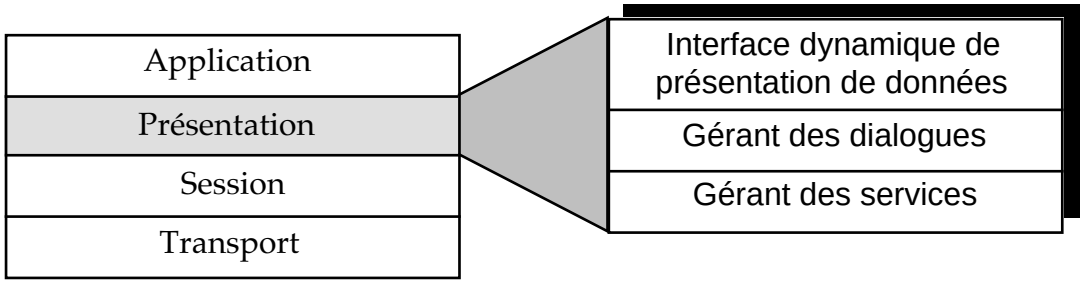


Figure 6.4.m : La couche présentation

Dans notre architecture, la couche présentation assure la **synchronisation** des actions de l'utilisateur vis à vis des programmes d'application. La synchronisation est le point le point le plus important pour la réactivité d'une interface utilisateur, c'est aussi un problème difficile à résoudre dans un environnement distribué. Un premier exemple de synchronisations apparait au moment où l'utilisateur choisit un service: le temps d'initialisation du programme d'application (lancement et exécution dans un environnement distribué) est plus long que le temps maximum autorisé pour une bonne réactivité à l'utilisateur [Shneiderman, 1984]. Un deuxième exemple de désynchronisation intervient lorsque qu'un service de transformation envoie des résultats, il faut que la fenêtre de résultats soit créée avant la réception du premier résultat. Ces deux exemples sont issus de problèmes temporels mais ne se traitent pas de la même manière. Dans le premier cas il faut améliorer la réactivité par localisation d'actions, dans le deuxième, il faut introduire une synchronisation explicite.

Nous montrons dans ce paragraphe que la couche présentation joue un rôle majeur dans **l'ouverture** de l'atelier (Objectif 16) en facilitant **l'intégration** des programmes d'application et en rendant les programmes d'application **indépendant** des représentations des données de l'atelier.

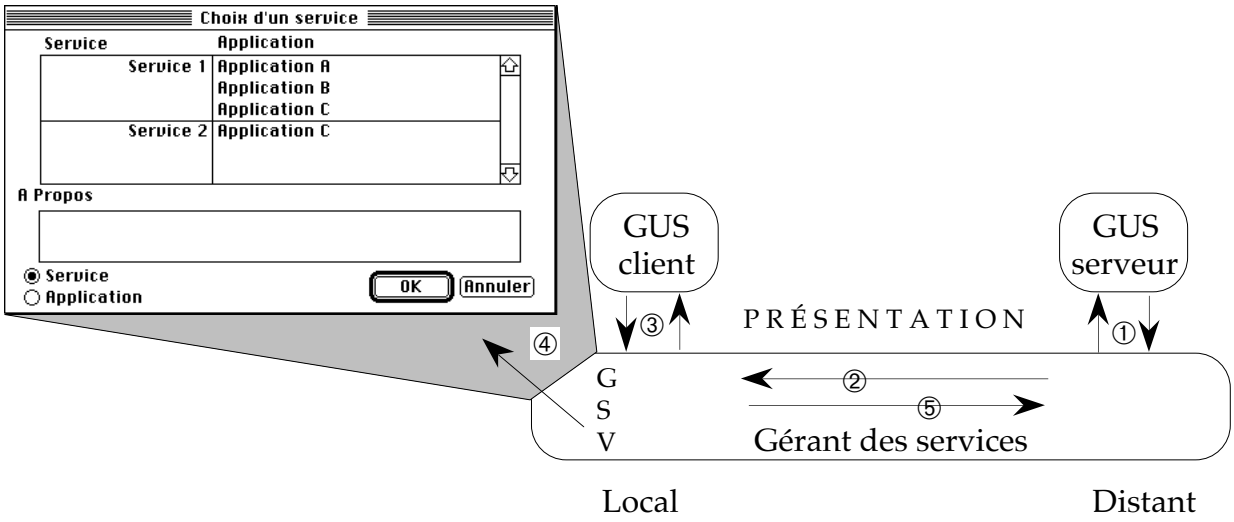
Nous présentons brièvement un scénario de dialogue de cette couche, de l'initialisation à l'exécution d'un service demandé par l'utilisateur. Nous détaillons ensuite les différentes sous-couches: gérant des services, gérant des dialogues et l'interface dynamique. Nous nous intéressons aussi à la définition du langage de description des énoncés (LDE) et le langage de description des résultats (LDR) au niveau de la couche présentation.

a. Scénario détaillé du protocole DAA

La couche présentation du coté distant, lors de son ouverture, hérite des propriétés de la couche session: le nom de l'utilisateur pour l'atelier, le nom de l'énoncé, sa date, le formalisme employé, la langue de l'utilisateur et l'adresse de transport de la couche session.

Par appel au GUS-serveur, la couche présentation (①) construit la liste des services (nom de service et options des services) accessible à l'utilisateur pour son énoncé et la transmet au GUS-client de l'entité locale (②) de l'application interaction utilisateur.

Le **gérant des services** par l'intermédiaire du GUS-client (③), propose à l'utilisateur l'ensemble des services (④). Le gérant des services stocke le nom de service choisi par l'utilisateur (⑤). Le GUS-serveur connaissant le service indique au gérant des services les programmes d'application qui réalisent ce service. Il faut rappeler qu'il y a une indépendance pour l'utilisateur entre les services et les applications qui réalisent les services. Le gérant des service choisit l'application à exécuter. Différents algorithmes de choix d'application sont mis en place à ce niveau. Deux algorithmes sont envisageables: soit il choisit la première application de la liste, soit il demande à l'utilisateur de choisir l'application par l'intermédiaire du gérant des services local.



① Construction de la liste des services
② transmission de la liste des services
③ enregistrement de la liste des services
④ présentation de la liste
⑤ transmission du choix de l'utilisateur

Figure 6.4.n: Choix d'un service

Le gérant des services a besoin d'informations sur le type du programme d'application, pour cela il fait appel de nouveau au GUS-serveur, qui lui fournit l'emplacement du fichier de renseignements sur le programme d'application. Nous avons commencé à entrevoir au chapitre §5.7.3 (Renseignements sur les applications) que les **fichiers de renseignements** sont exploitable par la plate-forme GSV, dans la section §6.4.3.b Gérant des services sur le gérant des services nous préciserons le contenu de ces fichiers.

Le gérant des services détermine si le programme d'application a déjà travailler sur cet énoncé. Pour cela, il fait appel au GUS-serveur qui stocke **les historiques des énoncé** (§5.4.3 Représentation des résultats). Le gérant des services fournit au GUS-serveur le nom, le formalisme et la date de l'énoncé, le GUS-serveur lui indique en retour si l'application a déjà travaillé sur cet énoncé.

L'entité distante du gérant des services possède une sous-couche (**lanceur**) permettant l'exécution du programme d'application. Ce lanceur grâce au contenu des fichiers de renseignements lance le programme d'application sur une machine du site supportant l'atelier. Lors de l'ouverture de la couche application, le lanceur lui transmet d'une part le nom de l'utilisateur, le nom et le formalisme de l'énoncé et le nom du service et d'autre part l'indication que l'application a déjà travaillé sur cet énoncé.

Le **gérant des dialogues** est responsable de l'ensemble des questions que l'on peut poser au programme d'application. Nous avons montré lors de l'étude de la plate-forme GUS que plusieurs services peuvent être regroupés dans un programme d'application (§5.5.1 Les programmes d'application). Les questions (services) que traite le programme d'application font partie des fichiers de renseignements (§5.8.1 Le Langage d'Arbre de Questions). Le gérant des dialogues distant, par l'intermédiaire de la plate-forme GUS, extrait l'arbre de questions. Pour satisfaire notre objectif de multi-linguismes (Objectif 8), la question est extraite dans la langue de l'utilisateur. Le gérant des dialogues interprète le langage LAQ pour le transformer en données internes.

Le gérant de dialogue distant garde l'état de toutes les questions: en cours, terminée…. Lors de l'initialisation de la couche application, il transmet au programme d'application, la question posée.

Le gérant des dialogues distant transmet l'arbre de questions et l'état des questions au gérant des dialogues local. Le gérant des dialogues local interprète l'arbre de questions et fait appel au gestionnaire des menus et des dialogues (niveau interface §7.4.2) pour le présenter graphiquement à l'utilisateur. Celui-ci sélectionne une question et le gérant des dialogues local indique la question "en cours" (le service demandé). Toutefois la question n'est réellement transmise par le gérant des dialogue local que quand il reçoit du gestionnaire des services l'indication de fin d'initialisation du programme d'application (message de synchronisation). Il est indispensable que les modifications de l'énoncé soient inhibées durant l'exécution du service (§7.4.2 Gestionnaire des menus et des dialogues).

Après cette étape de demande du service au programme d'application, l'entité distante du gérant des services dans le cas où l'application n'a pas travaillé sur l'énoncé, demande le transfert de l'énoncé à l'entité locale. **L'interface dynamique** de présentation du côté distant prend en charge le transfert, et reçoit dans le **langage LDE**, l'énoncé de l'utilisateur. Au niveau de la couche présentation, l'énoncé de l'utilisateur (en langage LDE) est décrit par les instances des classes de son formalisme. Les fichiers de renseignements permettent à l'interface dynamique de charger une table de réécriture des énoncés pour le programme d'application. L'interface dynamique transmet alors l'énoncé dans la syntaxe du programme d'application. Il est important de signaler que l'application reçoit toujours l'énoncé en entier (**mode bloc**).

A cette étape, le programme d'application exécute la question demandée sur l'énoncé. Durant toute l'exécution du service, le programme d'application envoie des informations à la couche présentation concernant le déroulement de l'opération. Toutes ces informations reçues sont des numéros de messages dont l'interface dynamique distante se sert pour accéder aux messages textuels correspondants dans les fichiers de renseignements. Le programme d'application, en fin de traitement, transmet à l'interface dynamique distante les résultats en langage LDR ou sous forme d'un nouvel énoncé ou sous forme textuel. L'interface dynamique distante, après traduction transmet les résultats à l'entité locale.

La prise en compte d'un mecanisme de reprise de présentation, sans intervention sur le niveau applicatif, est complexe à gérer dans un environnement distribué. Lors de la demande de fermeture de la couche présentation, provenant d'une demande de fermeture de la couche session, le gérant des services, suivant le mode de fermeture et le type de service, envoie un message de fermeture de l'application. Par exemple, dans le cas d'une fermeture de session avec une poursuite du service non interactif, le gérant des services conserve les messages provenant de l'application. Il les retransmettra lors de la reconnexion de l'utilisateur.

Par l'enchaînement des messages envoyés de la couche présentation, à la couche application, nous avons décrit rapidement le **protocole DAA** qui normalise l'enchaînement des traitements des programmes d'applications.

b. Gérant des services

Le gérant des services offre **différents types d'accés** aux applications, soit par un nom de service et ses options, soit par un nom de service sans option prédéfinie, soit par le nom de service puis le nom de l'application en faisant appel au gérant des dialogues pour accéder directement à tous les services d'une application particulière (cf Figure 6.4.o). Notre expérience nous a montré que cette navigation est essentielle car elle correspond à différentes utilisations des services suivant les besoins des utilisateurs. Ces différentes vues de l'atelier sont entièrement prises en charge par le gérant des services quelque soit le programme d'application.

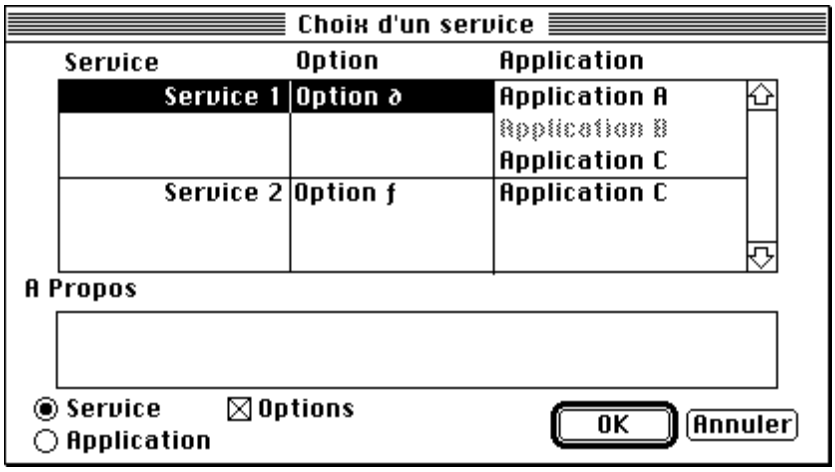


Figure 6.4.o : l'accès aux applications

Le choix d'un service est compatible avec l'appel de service pour une communication inter-applications (§5.5.3 Communication inter-applications). Dans cet appel le formalisme d'entrée est implicite (c'est le formalisme de l'énoncé), il manque le type de sortie (nouvel énoncé, LDR, texte) qui sera complété par le gestionnaire des dialogues de la couche présentation (paragraphe ci-après).

D'après la date d'estampille de l'énoncé (§5.4.3 Représentation des résultats), le gérant des services fait appel au GUS pour déterminer si l'énoncé est enregistré dans le système de fichiers distant. Ainsi, dans l'arborescence des historiques des énoncés de l'utilisateur (Figure 5.4.b Gestion d'un historique sur un énoncé), un nœud correspond au nom de l'énoncé dans le formalisme, estampillé par la bonne date. Lorsque l'utilisateur modifie l'énoncé, la date d'estampille est maintenue localement jusqu'à la prochaine question posée. Cette date est alors transmise au gérant des services distant. Pour assurer la cohérence des dates entre les gestionnaires de services, il est necessaire d'inhiber les modifications de l'énoncé durant l'exécution du service. Cette contrainte est toutefois naturelle.

Le gérant des services détermine, par appel au GUS (§5.8.3 Les résultats), si le programme d'application à déjà travaillé sur l'énoncé. Si l'application n'a pas travaillé avec l'énoncé, le gérant des services lui transmet l'énoncé. Si l'énoncé en LDE n'est pas présent dans l'historique, le gérant des services demande le transfert de l'énoncé au gérant des services local. Le transfert de l'énoncé de la couche présentation à l'entité distante de la couche application est réalisé par l'interface dynamique (§f Interface dynamique). Lorsqu'une application travaille sur un énoncé, elle doit sauvegarder sa description et les résultats calculés.

Lancement du programme d'application

Du coté distant, le gérant des services est responsable de l'exécution du programme d'application choisi par l'utilisateur.

Il a besoin du fichier de renseignements de l'application dont il connait la structure. Il en extrait les **moyens d'exécution** du programme, en particulier si le service est réalisé par un paquet de règles, par une application compilée ou interprétée (§5.7.3 Renseignements sur les applications). Dans chacun des cas, le gérant des services prépare l'environnement pour permettre l'exécution de l'application. Par exemple, pour utiliser un interpréteur, un appel au GUS fournit l'emplacement dans le système de fichiers distant de l'interpréteur et des fichiers "pré-construits" interprétant le protocole DAA. Cette étape de configuration, dépendante des moyens d'exécution de l'application, est détaillée dans la partie III §6 Réalisation sous Unix de la structure d'accueil.

Dans un environnement distribué, les programmes d'application sont susceptibles d'être utilisés sur certaines machines particulières (licence d'utilisation, environnement logiciel particulier). Ces indications de **restrictions de machine** font partie du fichier de renseignements de l'application.

Lorsqu'une application a déjà travaillé et a stocké des résultats dépendant d'une architecture de machine (§5.8.3 Les résultats), le gérant des services exécute l'application sur le même type de machine.

Le gérant des services s'appuie sur l'extension système (3.2.2.b) pour exécuter l'application, soit sur une machine spécifique, soit une des machines supportant l'atelier suivant l'algorithme interne du **gestionnaire de répartition** (répartion de charge…). L'application est exécuté dans le contexte système de l'utilisateur (droit de l'utilisateur vis à vis du système) et dans le contexte du système de fichier distant (niveau dans l'arborescence des historiques des énoncés).

Contrôle programme d'application

La couche présentation contrôle l'exécution du programme d'application en utilisant le protocole DAA. Lors de son initialisation, le programme d'application reçoit des messages du gérant des services lui indiquant le nom de l'utilisateur, le nom de l'énoncé, si l'application a déjà travaillé, la date, le nom de l'application et son emplacement dans le système de fichier distant. Nous verrons par la suite que ces informations n'atteignent pas le niveau application et sont utilisées par le niveau interface (§7.1.1 Le pilote d'une application).

Nous avons défini plusieurs **comportements d'application**. Ces comportement sont dans le fichier de renseignements de l'application.

Définition: Comportement d'application

Le comportement **cyclique** caractérise les applications dont les services peuvent être enchaînés sur un même énoncé. Le comportement **évolutif** se caractérise par le fait que d'application peut effacer son énoncé pour en recevoir une nouvelle version.

En fonction du comportement, le gérant des services utilise des particularités différentes du protocole DAA entre la fin d'exécution d'une question et le début d'exécution d'une nouvelle question.

Dans le cas cyclique, après avoir répondu à une question, l'application attend une nouvelle question de l'utilisateur. Si l'application n'est pas cyclique, elle s'arrête après avoir renvoyé les résultats. Le gérant des dialogues anticipe la prochaine question de l'utilisateur en relançant l'application et en lui indiquant qu'elle a déjà travaillé sur l'énoncé. Si l'application n'a pas un comportement évolutif et que l'utilisateur modifie l'énoncé, le gérant des services est obligé d'arrêter l'application en attente de question et de la relancer dans un nouveau contexte. Si l'application est évolutive, elle reçoit un message lui indiquant qu'elle va recevoir le nouvel énoncé. Elle devra, dans ce cas, détruire son ancien énoncé et les résultats associés.

Le mode de réception d'un énoncé fait partie du protocole DAA qui permet actuellement l'envoi en bloc.

Définition: Mode d'envoi d'un énoncé

L'envoi d'un énoncé en **mode bloc** correspond à l'envoi de l'ensemble de l'énoncé à l'application en une seule fois.

Le transfert des énoncés est réalisé dans un premier temps entre les entités locale et distante du gérant des services puis entre le gérant des services distant vers l'application.

Le gérant des services distant stocke une copie de l'énoncé dans l'historique (§5.4.3 Représentation des résultats).

Un objectif plus lointain serait de prendre en compte un mode d'envoi **immédiat** limitant le transfert de l'énoncé à ses modifications. Le mode immédiat diminue le temps de transfert. Si l'application veut recevoir l'énoncé en mode immédiat, elle doit construire et détruire dynamiquement les objets de l'énoncé ainsi que les résultats. Toutefois, il serait possible d'utiliser un mode immédiat entre les entités locale et distantes et un transfert en mode bloc vers le programme d'application.

c. Gérant des dialogues

Les fonctionnalités des services sont transmises et contrôlées par le gérant des dialogues sous la forme d'un arbre de questions. Le gérant des dialogue local après réception de l'arbre de questions, comme nous l'avons décrit dans le scénario, fait appel au gestionnaire de menus et du dialogue pour présenter graphiquement les questions sur l'interface utilisateur.

Lorsque l'utilisateur a choisi à un ensemble cohérent d'options de questions (§7.4.2 gestionnaire des menus et des dialogues), le gérant des dialogues transmet la question au coté distant en terme d'arbre de questions (sous-arborescence choisie) (§5.8.1 Le langage d'Arbre de Questions). Dans le cas où le service est interactif (§5.4.2 Les types de traitements d'un service), les feuilles de la sous-arborescence choisie contienne des données variables comme du texte ou des objets graphiques sélectionnés. Nous avons étendu le langage d'arbre de questions (LAQ) pour transmettre la sous arborescence choisie.

Lors du choix d'un service par l'utilisateur, la **première question** est **implicite**: elle correspond à la demande d'exécution du service. Cette question est posée par le gérant des dialogues.

Le gérant des dialogue distant conserve l'état des questions posées durant la session de travail de l'utilisateur sur l'énoncé. A la réception de la question de l'utilisateur, il la transmet à l'application.

> **Définition**: Etat des questions
>
> Le gérant des dialogues conserve l'**état des questions**, il intercepte les changements d'état envoyés par le programme d'application à l'application d'interaction utilisateur. Les états de question sont les suivant: inactif (la question n'a pas été posée), actif (la question est en cours de traitement), terminé (la réponse a été donnée).

Pour rendre indépendante l'application de la représentation de l'arbre de questions (LAQ), le gérant des services transforme la sous-arborescence en messages spécifiques à l'application. Nous complétons le langage LAQ par l'association d'un nom interne de question (connu de l'application) au nom externe dans la langue de l'utilisateur.

```
CreerQuestion(nom_service, nom interne)
```

Le gérant des dialogues distants transmet la sous-arborescence reçue sous forme d'un message représentant sous une forme "pseudo lisp" cette sous-arborescence dont voici un exemple: (serv1 (0 1 0)). Dans cet exemple (Figure 6.4.p) la question "APPLICATION" comporte deux services dont le premier est lui-même composée de trois options dont la deuxième est validée. L'application est capable d'interpréter le message `(serv1 (0 1 0)` car elle connait la nature et l'ordre des options.

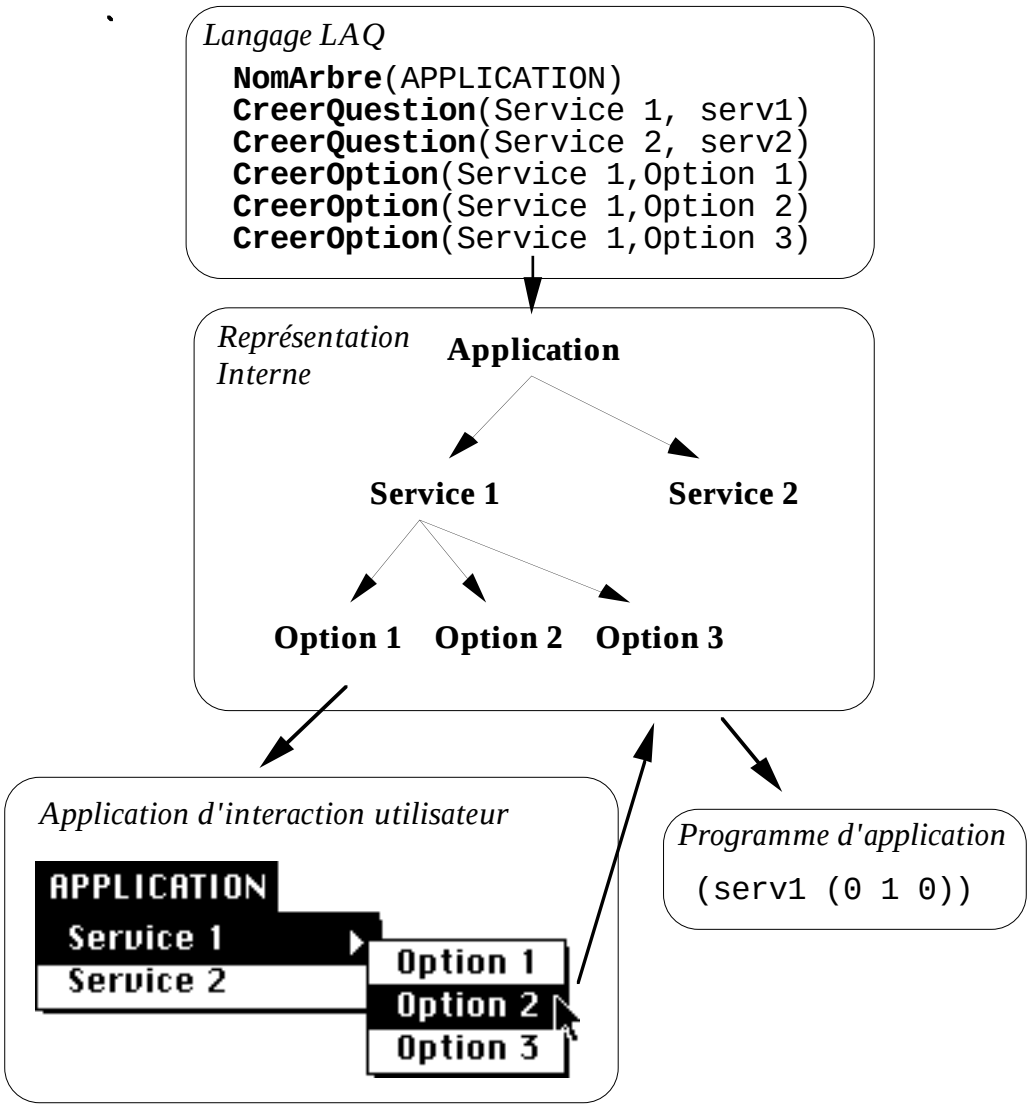


Figure 6.4.p Les différentes représentations d'un arbre de questions

Cette couche de présentation rend indépendante l'application des noms des questions et des options.

d. Le langage de description des énoncés (LDE)

Au niveau de la couche présentation, l'énoncé de l'utilisateur est décrit, en Langage de Description des Enoncés (LDE), par les instances des classes de son formalisme. Le langage LDE est interne à l'atelier, il est composé de messages. Sa définition découle de celle du Langage de Description des Formalismes (LDF).

Le langage LDE décrit les objets (Création d'un objet d'un type défini dans le formalisme) ainsi que les champs de l'objet (attribut des classes du formalisme). Tous les objets sont référencés par un **numéro interne** unique. Les attributs sont référencés par le numéro de leur objet associé. Lors de la création d'un objet par l'application d'interaction utilisateur (§8.2.1 L'éditeur de graphe), elle lui associe un numéro interne unique. De la même manière quand un service crée des objets du formalisme, il devra utiliser cette numérotation.

```
NomFormalisme(nom, version, LANGUE, nomInterne)
CreerObjet(nom classe, n°objet)
CreerConnection
     (nom classe, n°objet connecteur, n° nœud départ, n° nœud arrivée)
ValeurAttribut(n°objet, nom attribut, valeur)
```

Pour l'application d'interaction utilisateur, le LDE est complété par des informations graphiques comme la position des objets, des attributs et le chemin des connecteurs (point intermédiares sur les connecteurs). Ces informations graphiques ne sont jamais transmises du coté distant, elle sont uniquement utilisées dans le fichier énoncé stocké localement par l'application d'interaction utilisateur.

Extension graphique du langage LDE pour le fichier énoncé:

```
PositionObjet(n°objet nœud, postion)
PositionPointIntermediaire(n°objet connecteur,n°point,postion)
```

e. Interface dynamique

L'interface dynamique rend indépendante la couche application de la description des énoncés en LDE.

L'entité locale de l'interface dynamique est responsable de la constitution de l'énoncé en langage LDE par appels au gestionnaire de graphe de l'application d'interaction utilisateur. Cette entité locale rend indépendante la présentation graphique de l'énoncé à l'utilisateur du langage LDE.

L'interface dynamique distante a besoin de l'énoncé pour le transmettre à la couche application. Pour obtenir une indépendance entre les programmes d'application et le langage LDE, l'interface dynamique transcrit les messages LDE en messages pour les programmes d'application suivant des règles définies pour chaque application. Cette technique de réécriture est réalisée par des **tables de trancodage** stockées par le GUS dans les renseignement des applications(§5.7.3 Renseignements sur les applications). A l'inverse, pour rendre l'application indépendante du Langage de Description des Resultats (LDR), nous avons mis en place un **lexique** permettant de traduire message de résultats d'application en message LDR.

La table de transcodage

La table de transcodage fournit un moyen de transcoder le langage LDE en un langage connu de l'application (différent pour chaque application). Cette indépendance de l'application avec les représentations des énoncés de l'atelier amène une flexibilité et une ouverture de la structure d'accueil de l'atelier (Objectifs 16, 23). Il sera d'autant plus facile d'enrichir le langage LDE qu'il ne porte pas préjudice aux applications, et d'ajouter (récupérer) de nouvelles applications puisque la couche présentation adapte son dialogue pour chaque applications.

Il faut maintenant définir un moyen de traduire un énoncé en LDE en un autre langage. La simplicité de la description du formalisme en LDF et de l'énoncé en LDE, permet de réaliser cette opération de traduction par l'utilisation d'une table de trancodage.

L'interface dynamique connait la sémantique du langage LDF, elle fait appel à la plate-forme GUS pour obtenir la description d'un formalisme. La description en langage LDF d'un formalisme permet de connaitre l'ensemble des messages LDE possible. Le langage LDE étant composé de messages, la table de transcodage permet de réécrire chaqu'un des messages LDE en un autre message pouvant utiliser les paramètres du message LDE.

Par exemple, dans le formalisme des réseaux de Petri ordinaires, la création d'un objet de type *place* de numéro interne *12* génère le message LDE suivant: *Creer_Objet("place",12)* . Ce message peut être réécrit "{PL 12}" pour une application donnée. Un exemple complet de table de transcodage est donné dans la partie III (§6.3Plate-forme GSV).

La construction de la table de transcodage fonctionne pour un formalisme dans toutes les langues avec une description unique des règles de réécriture pour chaque application.

Le Langage de description des résultats (LDR)

Nous venons de voir comment l'énoncé de l'utilisateur est transmis à l'application. Nous allons voir maintenant les différents types de résultats que peut rendre une application et les évolutions à apporter à la couche présentation.

Tout d'abord, le gérant des dialogues, connaissant l'état des questions, encapsule la réponse de l'application par deux commandes du Langage de Description des Résultats (LDR):

```
DebutReponse(nom_service, [nom_option_service])
FinReponse()
```

La première forme de résultat est la plus simple: c'est le résultat textuel. Deux formes pourtant existent: l'envoie de message "en dur" c'est à dire dans le code de l'application ou l'envoie de messages traduits. Dans le deuxième cas, l'interface dynamique offre une possibilité de fabriquer des messages dans lesquels des parties de texte variables peuvent être insérées. On obtient la partie du Langage LDR suivant:

ReponseTextuel("texte" [,valeurs]***)**

La deuxième forme de résultat est la mise en évidence d'ensembles d'objets associés à des textes. Cette forme de résultats permet par exemple d'indiquer des **erreurs** associées à des objets ou de montrer des ensembles d'objets étant résultats du calcul. Ces ensembles d'objets sont encapsulés entre deux messages de début et de fin d'ensemble.

DebutEnsemble(["nom de l'ensemble"]**)**
FinEnsemble()

Pour former les messages, l'application doit désigner les objets. Elle le fait grace aux numéro internes des objets.

MiseEnEvidence(n°objet**)**
ReponseTextuel("texte" [,valeurs]***)**

La troisième forme consiste à modifier des ensembles d'attributs textuels. Pour modifier les attributs textuels, elle désigne l'objet par son numéro et l'attribut par son nom. Nous avons rendu indépendant l'application de la description du formalisme par un **lexique** associant les noms des attributs connus de l'application aux noms connus du formalisme (dictionnaire du formalisme (§5.4 Gestion des services)).

ChangerAttribut(n°objet,nom attribut,valeur**)**

La quatrième forme de résultat consiste à supprimer des objets de l'énoncé. Pour cela les objets à supprimer son désignés par leur numéro internes.

SupprimerObjet(n°objet**)**

La cinquième consiste en la création d'objets. Pour cela l'application doit générer des numéro d'objets différents de ceux déjà employés. De plus les noms des classes des objets seront traduit par le lexique.

CreerObjet(nom classe, n°objet**)**
CreerConnection
(nom classe, n°objet connecteur, n° nœud départ, n° nœud arrivée**)**

Enfin, le résultat d'un service peut produire un nouvel énoncé en langage LDE. Pour cela un lexique du formalisme du résultat pour l'application traduit le nom de classe et les noms des attributs.

6.4.4. Couche application

La couche application est la couche la plus haute du modèle OSI. Elle contient toutes les fonctions impliquant des communications entre systèmes, en particulier si elles ne sont pas réalisées par les couches inférieures. Elle s'occupe essentiellement de la sémantique, contrairement à la couche présentation, qui prend en charge la syntaxe. Dans notre architecture hiérarchique de l'atelier, elle correspond aux niveaux interface et applications (paragraphes §7 Niveau interface et §8 Niveau applications ci-dessous).

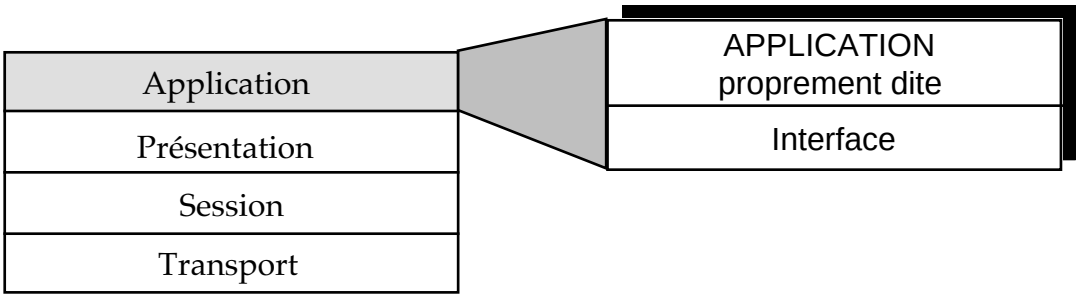


Figure 6.4.q: La couche application

7. Niveau interface

Nous présentons les modules du niveau interface (Figure 7). Cette couche interface programmes d'application dans le modèle OSI se situe au niveau application. Nous avons délibérément découpé cette couche en deux sous-couches: le niveau interface et le niveau application. Le niveau interface a pour rôle de rendre indépendant les programmes du niveau plate-forme. Le niveau interface facilite ainsi l'intégration de nouvelles applications.

Dans ce paragraphe, nous détaillons successivement les fonctionnalités des modules: interface programme d'application, l'application système expert, l'interface de communication entre application et l'interface d'application d'interaction utilisateur.

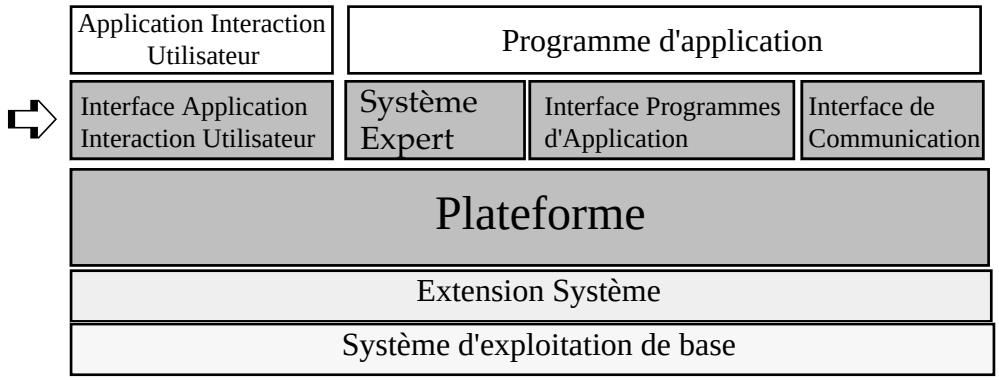


Figure 7: Le niveau interface

7.1. Interface programmes d'application

L'intégration d'une application dans l'atelier passe par une phase d'enrobage par un ensemble d'interfaces standardisées. Ces interfaces permettent de contrôler le déroulement de l'application et de centraliser les demandes d'entrée/sortie pour maintenir l'intégrité des résultats.

Dans une approche d'ouverture (Objectif 16), nous avons exploité la structure en couches du GSV et introduit la notion de pilote pour faciliter l'intégration d'application.

Le comportement de l'application vis à vis de la plate-forme GSV de l'atelier est défini par le protocole **D**ialogue **A**pplication **A**telier (§5.5.1 Les programmes d'application). L'interface programmes d'application est responsable de la gestion du protocole, et du "déroulement" de l'application; elle **pilote** l'application vis à vis de l'atelier.

7.1.1. Le pilote d'une application

Le pilote contient un ensemble de fonctions (**interface - pilote**) prédéfinies accessibles à l'application. Tous les accès au Gestionnaire des Utilisateurs et des Services (GUS) et au Gestionnaire de Station de travail Virtuel (GSV) passent par ces fonctions. Certaines de ces fonctions, lors des résultats des application, génèrent des messages pour la couche présentation du GSV. La couche présentation interprète ces messages et génère des résultats en Langage de Description de Résultats (LDR) ou Langage de Description d'Énoncés (LDE). Ces résultats sont interprétés par l'interface d'application utilisateur et visualisés graphiquement par l'application d'interaction utilisateur.

Exemple
Dans le formalisme réseaux de Petri, le résultat d'un programme d'application consiste en la création d'une place. L'application désigne cet objet par un nom interne "PL", elle appelle une fonction du pilote "CreerObjet" en lui indiquant le type "PL", sont numéro x. Le pilote envoie le message correspondant. La couche présentation transforme ce message en LDE: "Création d'un objet de type "place", formalisme réseau de Petri ordinaire, d'objet numéro x". L'interface d'application utilisateur ajoute une instance de place ayant le numéro x dans la fenêtre de l'énoncé résultat. L'application d'interaction utilisateur détermine une position adéquate pour la représentation graphique de la place.

Cet exemple montre l'indépendance du programme d'application vis à vis de la représentation graphique des résultats. De même, l'utilisateur interagit sur le déroulement de l'application par des actions sur l'interface utilisateur qui sont transmises au pilote de l'application.

Ainsi, le pilote **gère la communication** avec l'application mais c'est aussi un **gérant de données**. Le pilote garde trace de tout ce qui s'est passé, les questions qui ont été posées à l'application, les résultats et **l'historique des événements** s'étant produits avec l'application. Le pilote est exécuté par la couche présentation qui lui impose un **contexte utilisateur** (environnement multi-utilisateurs). Lors d'une re-exécution du même pilote sur le même énoncé, la couche présentation le remet dans le même contexte vis à vis de ses données. Le pilote est ainsi capable de connaître ce qui a déjà été réalisé par l'application.

Un pilote est en interaction avec quatre entités : la plate-forme GSV, la plate-forme GUS, l'application et parfois le système d'exploitation. Les liaisons entre, d'une part le pilote et les plate-formes GUS et GSV et d'autre part, le pilote et une application sont fondamentales. Elles supportent un transfert de données lorsqu'un utilisateur demande la réalisation d'une des fonctionnalités de l'application. Bien que, les applications doivent être le plus indépendante possible de l'environnement [Purtillo, 1985] une liaison entre le pilote et le système d'exploitation facilite l'écriture de l'application.

Le pilote comporte un **gérant des Entrées/Sorties** qui centralise l'accès au système de fichier distant. La couche présentation exécute le pilote de l'application en lui fournissant un contexte d'exécution particulier (contexte statique et dynamique de l'utilisateur et informations sur l'application (§6.4.3.b Gérant des services). A l'initialisation, le pilote exécute une fonction prédéfinie qui permet de récupérer les messages d'initialisation de la couche présentation (nom de l'utilisateur, le nom du programme d'application,...). Ces informations sont conservées par le module interface - pilote et sont nécessaires pour effectuer des **appels au GUS**. Par exemple, si le pilote a besoin d'un fichier d'initialisation particulier pour l'application, il fait un appel à une fonction de l'interface - pilote qui construit un appel au GUS en lui fournissant le nom de l'application. Le GUS renvoie l'emplacement de ce fichier dans le système de fichier distant.

L'utilisation d'un gérant **des Entrées/Sorties** et l'appel au GUS rend l'application indépendante de son environnement d'exécution. Ainsi, la couche présentation est capable d'exécuter une application sur n'importe quelle machine du site. Les algorithmes utilisés, pour exécuter les applications de manière distribuée ont été détaillés dans la couche présentation du GSV (§6.4.3.b Gérant des services). Bien entendu, il ne s'agit pas de distribuer les algorithmes internes de l'application mais de choisir un site d'exécution pour une application. Nous obtenons ainsi une liaison indirecte entre le pilote et le système, ce qui nous permet de contrôler les accès du pilote au système.

7.1.2. Liaisons enter le pilote et la plate-forme GSV

Les règles d'échange des données entre la plate-forme GSV et le pilote définissent le protocole DAA.

L'utilisateur ayant demandé un service, la plate-forme GSV exécute une application. La question de l'utilisateur, posée par l'intermédiaire de l'application d'interaction utilisateur, transite par le pilote qui reçoit l'énoncé. Le pilote confie la question et l'énoncé à l'application. Celle-ci exécute alors la requête de l'utilisateur. Ensuite, le pilote récupère les données résultantes, en les affinant si nécessaire pour enfin, les faire parvenir à la plate-forme GSV qui les transmet à son tour à l'utilisateur. Comme on peut aisément le constater, le flux de données forme un cycle empruntant chacune des liaisons, dans un sens puis dans l'autre.

La plate-forme GSV et le pilote communiquent en utilisant un protocole d'échange de messages (§3.2.1 Système - Répartition). La réalisation du protocole DAA par le pilote permet d'une part le contrôle de l'application et d'autre part la transmission de l'énoncé et des questions d'interrogation d'une application.

Le pilote est l'initiateur de la communication avec le GSV. Dès la fin de son initialisation et de celle de l'application, il en informe la couche présentation du GSV.

Le pilote connaît d'une part, la syntaxe de l'énoncé qu'il va recevoir et d'autre part, celle de l'application. Dans la plupart des cas, une application garde trace des résultats d'un traitement dans une structure de données propre. Si nécessaire, elle demande au pilote de les transmettre à la plate-forme GSV soit dans le langage LDE (si c'est un nouvel énoncé) soit dans le langage LDR. Signalons que la plate-forme GSV ne garde pas trace des résultats qu'elle reçoit du pilote. C'est donc à l'application de conserver les résultats dans ses structures de données.

7.1.3. Liaisons entre le pilote et l'application

Notre expérience nous a amené à distinguer deux typologies de pilotes. En effet, le pilote peut, soit faire corps avec l'application, soit en être complètement dissocié. Nous parlerons dans le premier cas, d'un **pilote interne** et dans le second, d'un **pilote externe**.

Bien évidement la liaison pilote-application est dépendante de la typologie de pilote. Un pilote interne (Figure 7.1.a) est une unité de compilation liée à l'ensemble des unités de compilation composant l'application. Ainsi, il partage le même espace adressable avec l'application. On peut donc envisager le partage de structures de données mais aussi de procédures.

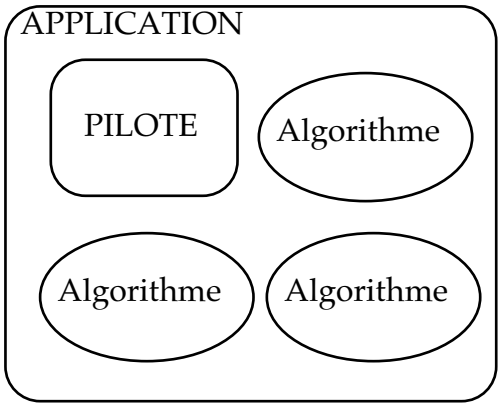


Figure 7.1.a: Pilote interne

Un pilote externe (Figure 7.1.b) est une entité complètement dissociée de l'application. C'est un processus à lui tout seul. Les espaces adressables sont disjoints. C'est donc au moyen de fichiers, pipes etc... que communiquent le pilote et l'application.

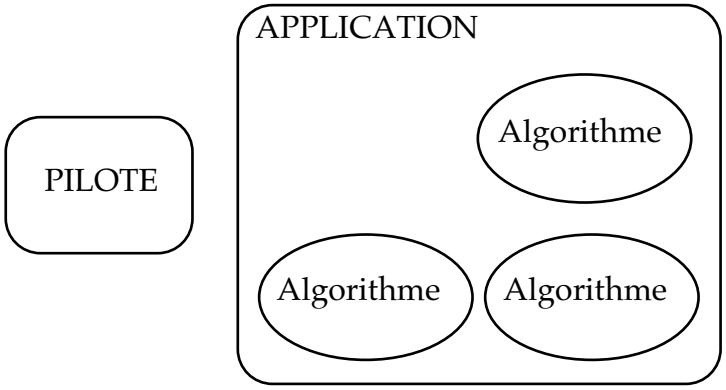


Figure 7.1.b: Pilote externe

Si le pilote est interne, la constitution de l'énoncé pour l'application est réalisée directement par accès par le pilote aux structures de données de l'application. Si le pilote est externe, il transmet par un moyen de communication l'énoncé dans la forme voulue par l'application.

Lorsque la liaison pilote-application est mise à contribution, le pilote dispose de l'énoncé et de la question. Le pilote a alors, une mission de traducteur quand les formats des données de l'application ne peuvent être rendus compatibles par l'atelier (§6.4.3.e Interface dynamique). Cette phase de traitement se caractérise par une une totale dépendance vis-à-vis de l'application.

7.2. Application - Système Expert

L'atelier de spécification possède un programme d'application spécifique: un système expert. Ce module application-système expert est composé de deux couches un système expert et une interface du système expert avec le niveau plate-forme.

L'interface est un pilote particulier puisqu'il est écrit en actions du sysème expert. De plus, nous avons montré dans la partie §5.6 Gestion des applications paquets de règles, que le système expert gère la notion de paquet de règle. Rappelons qu'un paquet de règles (niveau programmes d'application) est vu comme la réalisation d'un service. Ainsi, l'atelier (plate-forme GUS) gère les bases de connaissances du système expert pour chaque formalisme. L'interface intégre donc des actions spécifiques pour charger des paquets de règles.

Le système expert tient spécialement compte des problèmes spécifiques posés par notre domaine de recherche [Beldiceanu, 1988a]. Il possède un langage de règles proche des définitions mathématiques, des quantificateurs multiples, des règles indépendantes de la représentation des connaissances, . Il permet l'expression de contraintes globales entres variables, un choix de représentations adaptées aux problèmes, l'accès implicites à des structures complexes (relations, ensembles, matrices creuses) en tant que types abstraits, la coopération de règles et d'algorithmes. Les énoncés sont ainsi représentables sous forme de faits indépendamment des algorithmes. L'utilisation d'actions et de contraintes pré-programmées, ainsi que la gestion de représentations multiples de connaissances permettent une écriture synthétique des règles.

7.3. Interface de communication

Nous avons exposé (§5.5.3 Communication inter-applications) le principe de la communication inter-applications. Le module interface de communication, permet de réaliser cette communication. Nous présentons maintenant la technique utilisée pour rendre transparent ce dialogue.

Lorsque différentes applications travaillent sur des données communes (issues ou non d'un même formalisme), on est confronté aux problèmes classiques de partage d'informations. Les problèmes à résoudre portent sur l'existence d'une représentation externe unifiée et sur la gestion du transfert, à un niveau élémentaire, de résultats entre différents services (Objectif 14).

L'architecture de l'atelier de spécification permet la communication entre l'application d'interaction utilisateur et les programmes d'application. Nous avons montré que les programmes d'application ne sont pas liés à l'application d'interaction utilisateur (§7.1 Interface programmes d'application). La structure d'accueil, telle que nous l'avons décrite, réalise déjà une communication entre applications (Figure 7.3).

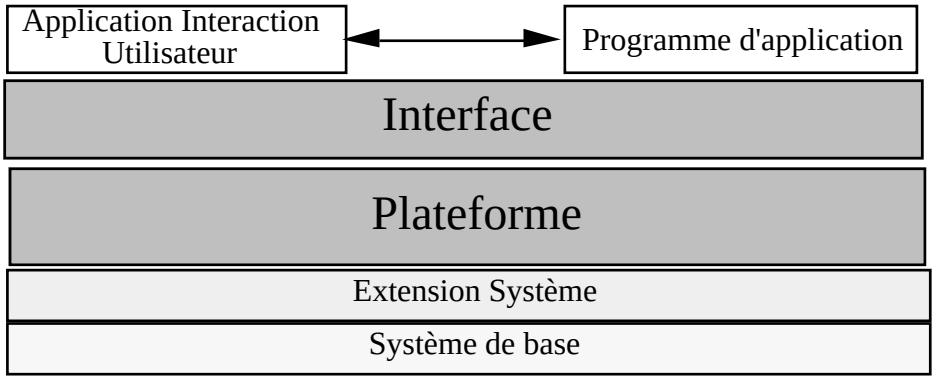


Figure 7.3: Communication entre applications

Pour réaliser une communication entre les programmes d'application, il suffit que le programme demandeur se comporte comme une application d'interaction utilisateur: c'est le rôle de l'interface de communication. Illustrons cette communication par un exemple:

Exemple:

Le programme A demande un service. Cette demande correspond à une demande de communication vers un programme B (§5.5.3 Communication inter-applications). Le module interface de communication se faire passer pour l'application d'interaction utilisateur pour le programme B. Le module interface de communication traduit la demande de service du programme A en une question pour le programme B (§6.4.3 Couche présentation). Le module interface de communication fournit un moyen de transférer l'énoncé de A vers B et de recevoir les résultats.

7.4. Interface application d'interaction utilisateur

L'interface Application d'Interaction Utilisateur est composée de deux gestionnaires de haut niveau comme le **gestionnaire de graphe** et le **gestionnaire des menus et des dialogues** décomposé en deux entités: les menus et dialogues de l'éditeur de graphe et ceux des services (programmes d'application).

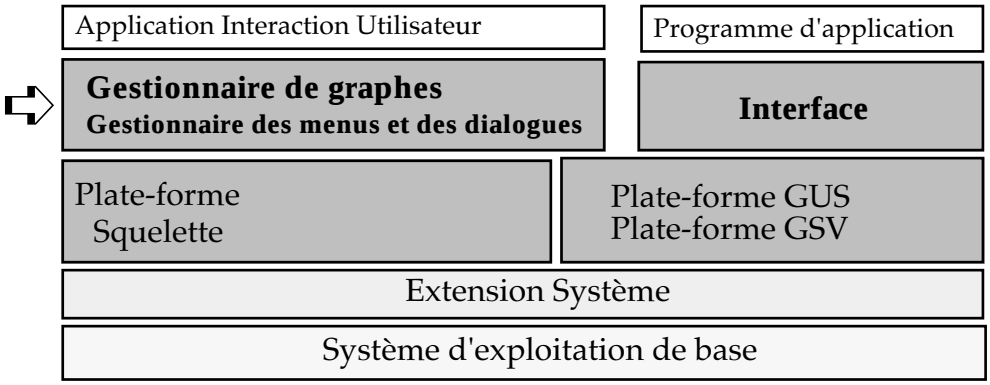


Figure 7.4: Le niveau interface d'application d'interaction utilisateur

7.4.1. Le gestionnaire de graphes

Le gestionnaire de graphe gère les **objets du graphe** (nœuds et connecteurs) ainsi que les textes composants le graphe. Il assure les fonctions de bases utilisées dans l'éditeur de graphes (changement de position, dessin). Il fait le lien entre les objets du graphe et leur fenêtres d'appartenance. Il contrôle en particulier les objets visibles dans chacune des fenêtres.

7.4.2. Le gestionnaire des menus et des dialogues

Nous pouvons distinguer deux sortes de dialogue dans cette application. En effet, l'interface utilisateur gère le dialogue associé à l'éditeur de graphes et le dialogue des programmes d'application intégrés dans l'atelier.

a. Gestionnaire des menus de l'éditeur de graphes

Le gestionnaire de menus et du dialogue est responsable de l'enchaînement et de la disponibilités des fonctionnalités de l'éditeur de graphe. Il s'appuie sur la squelette menu et dialogue (§4.5 Squelette des menus et 4.6 Squelette des dialogues) de façon à assurer ses fonctionnalités.

Le gestionnaire du dialogue est responsable de l'enchaînement des différentes étapes du dialogue par la présentation des différentes boîtes de dialogue. Il s'agit de la syntaxe de haut niveau du dialogue.

Dans le cas de menus de l'éditeur de graphe, il fait appel l'éditeur de graphe (§8.2.1 L'éditeur de graphes) pour calculer la validité de certains éléments des menus.

b. Gestionnaire des menus des services

L'interface utilisateur des services représente l'interface que voit l'utilisateur lorsqu'il accède aux services de l'atelier. C'est cette partie qui sera chargée d'interpréter les arbres de questions (§6.4.3 Couche présentation) pour les présenter sous formes de menus ou de fenêtres de dialogues en s'appuyant sur le squelette des menus et des dialogues (§4.5 Squelette des menus et 4.6 Squelette des dialogues).

Le gestionnaire des menus regroupe l'ensemble des interactions avec l'utilisateur nécessaires pour formuler une question. Une fois tous les paramètres de la question enregistrés, elle sera transmise, à la demande de l'utilisateur, au programme d'application par le gestionnaire de station virtuelle.

Lorsque l'utilisateur a posé une question, le gestionnaire des menus transmet l'information à l'éditeur de graphe pour qu'il inhibe les actions de l'utilisateur visant à modifier l'énoncé initial. L'énoncé est de cette manière protégé durant l'exécution d'un service car toute modification syntaxique peut introduire une incohérence sur le résultat.

Lorsqu'une exécution de service demande un temps de calcul assez long, l'utilisateur est prévenu immédiatement que l'action a bien été prise en compte ("feed back" immédiat et informatif) et est constamment informé de l'évolution de sa commande. On obtient ainsi une simplicité d'interaction par normalisation (Objectif 9). Pendant l'exécution du service, l'interface utilisateur des services est chargée de présenter à l'utilisateur les différents messages en provenance des applications de calcul.

Une description de réalisation est faite dans la partie III, §7.5 Utilisation de l'atelier: Application.

8. Niveau applications

La figure 8 présente la circulation des messages entre l'application d'interaction utilisateur (AIU) et le programme d'application. Elle détaille l'entité distante de la couche présentation.

L'utilisateur a saisi un énoncé dans une fenêtre graphique (①) et demandé l'exécution d'un service. La couche présentation distante a donc exécuté l'application correspondant au service. Le gérant des dialogues a transmis l'arbre de questions correspondant au gérant des dialogues de l'AIU qui le présente sous forme d'un menu (②). Lorsque l'utilisateur choisit un article dans le menu du programme d'application, la question en langage LAQ est transmise au gérant des dialogues distant (③) qui la traduit et envoie un message au pilote de l'application (④). Le gérant des services demande le transfert de l'énoncé à la couche présentation locale. L'énoncé est transmis à l'interface dynamique en langage LDE (⑤). L'interface dynamique trancode le langage LDE en messages spécifiques pour l'application (⑥). Le programme d'application effectue le traitement demandé et renvoie les résultats par l'intermédiaire du pilote (⑦). La couche présentation reçoit ces résultats (⑧) qu'elle transmet en langage LDR (⑨) à l'application d'interaction utilisateur qui les visualise graphiquement.

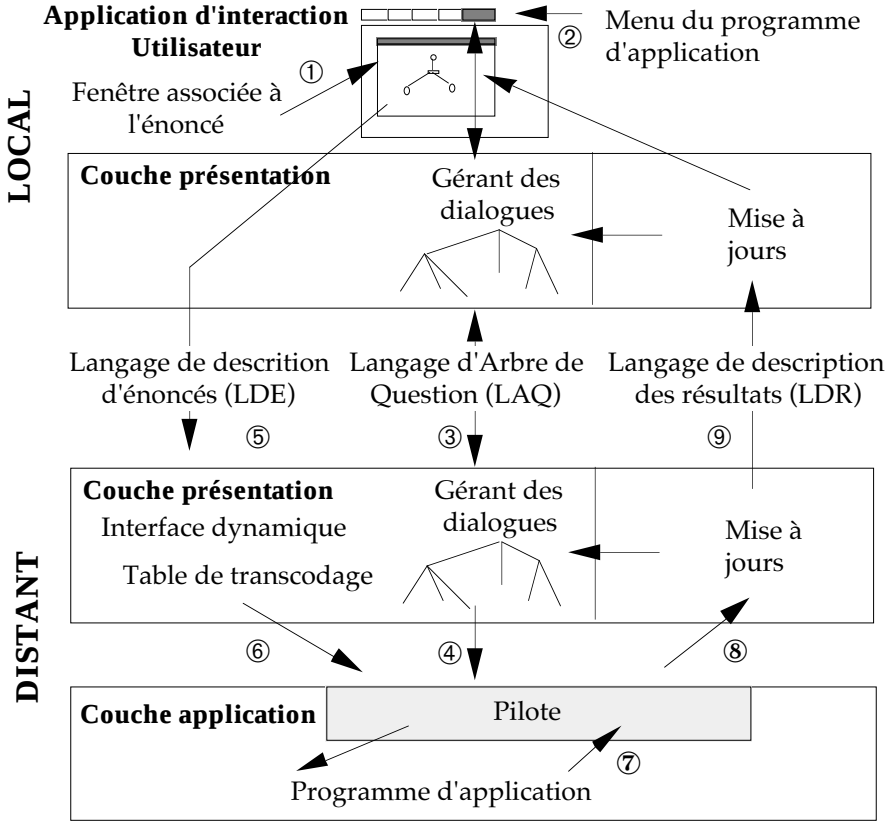


Figure 8 : Circulation des messages entre l'interface utilisateur et une application

Nous présentons d'une part les programmes d'application et d'autre part l'application d'interaction utilisateur.

8.1. Programmes d'application

8.1.1. Les deux catégories d'application

Nous avons défini deux catégories de services (§5.4.1 Les deux catégories de services): services utilisateurs travaillant sur les données de l'utilisateur et services d'administration exploitant les données internes de l'atelier. Des services d'un même formalisme peuvent être regroupés au sein d'une application (§5.5.1 Les programmes d'application). Il existe ainsi deux catégories d'applications:

Définition: Les deux Catégories d'application

Il existe deux catégories d'applications: les **applications utilisateur** et les **applications d'administration**.

La catégorie d'application fait partie des **renseignements** sur les applications.

8.1.2. Application - Programme - Règles

La couche application est la couche la plus externe du modèle. Les services connaissent la sémantique de l'énoncé. Le coté utilisateur forme l'interface utilisateur et correspond à des applications d'introduction et de visualisation de données.

Les Services permettent d'une part l'exécution des traitements sur les énoncés et d'autre part l'administration de tout l'atelier (gestion des utilisateurs, des services et des domaines).

Le concepteur d'un service peut utiliser le système expert et écrire sous forme de paquets de règles la réalisation de son service. Dans ce cas le programme d'application définit les paquets de règles associés au service.

Nous avons donc réussi à homogénéiser la notion de programme d'application (code exécutable) et celle de paquets de règles (interprété). Nous avons ainsi défini une notion de **machine virtuelle**.

8.1.3. L'intégration d'application

L'intégration d'une application dans un système consiste à remodeler cette application pour que les fonctions qu'elle réalise soient utilisables dans un cadre uniforme. Une application intégrée participe à la réalisation des services de l'atelier. Ses fonctions entrent dans la composition de services au même titre que n'importe celles des autres applications de l'atelier.

Avant d'être intégrée dans un atelier logiciel, une application fonctionne de façon autonome. Elle comporte en général des composants logiciels spécialisés dans la communication avec l'extérieur. Il s'agit principalement d'une interface utilisateur, et d'une interface avec le système d'exploitation composée d'utilitaires permettant l'échange de fichiers avec d'autres applications.

Une telle utilisation autonome d'une application demande à tout utilisateur novice de se familiariser avec une nouvelle interface utilisateur. Malgré la qualité de celle-ci, cet apprentissage est une perte de temps considérable, car il devient caduque dés que l'on change d'application. Certes les concepteurs d'applications tentent de lutter contre cet inconvénient, en utilisant la même interface pour plusieurs applications. Toutefois, cette solution ne fait que diminuer la fréquence d'apparition du problème : dés que l'on sort d'une "famille d'applications", il faut apprendre une nouvelle interface.

Une fois intégrée dans un atelier logiciel, une application perd son autonomie pour faire partie intégrante d'un groupe d'applications. L'application est ainsi utilisable de manière plus homogène et plus pertinente. Chaque application intégrée est utilisée dans la réalisation d'un ou plusieurs services accessibles à travers une interface utilisateur unifiée, cachant complètement aux utilisateurs le caractère hétérogène (regroupement d'applications diverses) de l'atelier. Chaque utilisateur de l'atelier ne connait donc que l'interface utilisateur de l'atelier et non les détails des applications.

Par ailleurs, la communication inter-applications est assurée par la structure d'accueil de l'atelier chargée de réaliser une interaction maximale entre les applications (§7.3 Interface de communication).

On assiste de surcroît, aux développement parallèles, dans des laboratoires différents, d'applications réalisant des fonctions similaires voire identiques. La fonction d'intégration tend à limiter ce phénomène en favorisant la réutilisation, donc la diffusion des applications parmi les équipes de recherche, et conduit à une certaine concertation quant aux choix des orientations guidant le développement de composants logiciels dans les laboratoires.

De plus, la perspective de bénéficier des services d'un atelier logiciel (du service d'interaction avec l'utilisateur notamment) arrive à stimuler les équipes de recherche pour développer des maquettes de logiciels destinés à être intégrés et mettre en avant des résultats théoriques récents. L'intérêt de telles maquettes réside essentiellement dans l'accélération de leur mise au point et leur concision. Les concepteurs se limitent dès lors strictement à la programmation de leurs résultats théoriques. Leur but est la diffusion de leurs outils à la communauté des utilisateurs sur site ou à distance.

La fonction d'intégration ne se limite donc pas à la réutilisation d'applications existantes (**intégration à posteriori**), mais concerne aussi la réalisation de nouvelles applications réutilisables, destinées à être intégrées (**intégration à priori**).

8.2. Applications d'interaction utilisateur

8.2.1. L'éditeur de graphes

L'éditeur de graphes (Objectif 10) est une partie centrale de notre travail. Bien que programmé de manière "traditionnelle" c'est à dire dans un langage de haut niveau, il n'en demeure pas moins intrinsèquement lié à la notion d'**interface utilisateur privilégiée** (Objectif 9). En effet, à tous moments, l'utilisateur doit pouvoir agir. Ceci implique de ne pas entrer dans de longues périodes de calcul et de pouvoir quasi instantanément changer de nature de travail. Il faut par exemple permettre la suppression d'objets en pleine réorganisation graphique de ces mêmes objets.

Dans le dessin de graphes, on distingue les parties esthétique, syntaxique et sémantique.

- La partie **esthétique** concerne la forme graphique des objets comme la taille d'un nœud, le type de trait, les couleurs et les positions. La couleur ou les niveaux de gris peuvent être utilisés pour mettre en valeur certains objets afin de visualiser des résultats.
- La partie **syntaxique** caractérise les différents types d'objets. Elle reflète le formalisme. Par exemple dans le formalisme de réseaux de Petri, un nœud de type place, la définition d'attribut de type marquage pour les objets de type place.
- La partie **sémantique** vérifiée au niveau de l'éditeur de graphes, concerne en particulier les autorisations de connexion des arcs sur les nœuds. Elle garantit aussi les valeurs par défaut de certains attributs.

La partie esthétique n'interesse pas les programmes d'application; par conséquent, qu'il sera possible de modifier l'esthétique d'un énoncé en cours de calcul ou l'esthétique d'un résultat avant que les calculs soient terminés.

L'éditeur de graphes est composé d'une juxtaposition de fonctionnalités définies par les besoins des utilisateurs et par les habitudes de travail avec des logiciels similaires comme les éditeurs graphiques (Composante 2). Ces fonctionnalités font en général appel à un gestionnaire de graphes sous-jacent pour connaître les objets du graphe de l'utilisateur. Les fonctionnalités sont dans la majorité de cas très indépendantes les unes des autres. Par exemple une fonctionnalité d'alignement est totalement indépendante d'une fonctionnalité de modification d'attributs. L'ajout de fonctionnalités de haut niveau et performantes contribuent à la qualité de l'éditeur de graphes.

L'éditeur de graphes est basé sur le **gestionnaire de formalismes** (§5.3 Gestion des formalismes) pour connaître les différents type d'objets à présenter à l'utilisateur. Par contre, cet éditeur ne contient **pas de fonctionnalités spécifiques** à un formalisme particulier. Pour ajouter une fonctionnalité spécifique, il est en général souhaitable d'en trouver une généralité réutilisable pour d'autres formalismes. Par exemple, si on se rend compte que l'on veut présenter un l'évolution d'un histogramme lors d'une simulation, il ne s'agit, ni plus ni moins, que d'une nouvelle façon de présenter un attribut numérique d'un objet. A la demande de l'utilisateur, un tel attribut numérique ne sera plus représenté sous la forme textuelle habituelle mais sous forme graphique. Par extension, il devient aussi souhaitable de saisir ce type d'attribut de manière graphique. Nous obtenons ainsi une interface utilisateur homogène (Objectif 9).

La conception hiérarchique graphique de modèles consisterait à dessiner un modèle de haut niveau et à raffiner les objets du formalisme en créant des sous-modèles eux-même pouvant être raffinés. Cette conception hiérarchique pourra être entièrement cachée aux programmes d'application si l'éditeur de graphes leur fournit une description planaire de l'ensemble du modèle.

L'éditeur de graphes prend en compte le **degré de compétence** de l'utilisateur. Cette prise en compte dans l'utilisation de l'interface graphique est traitée de manière décentralisée au niveau de chaque application d'interaction utilisateur. En effet par le jeu de fichiers de préférence, il est possible de configurer une partie de l'interface utilisateur (Objectif 8).

Une description plus complète des fonctionnalités de l'éditeur de graphes est faite dans la partie III, §5 Réalisation de l'application d'interaction utilisateur et §7 Utilisation de l'atelier. En particulier le suivit des connecteurs lors du déplacement d'objets, l'organisation des points de d'arrivée des connecteurs sur les nœuds, les problèmes de couper/coller…

8.2.2. L'interface utilisateur d'administration

L'interface utilisateur d'administration, comme l'éditeur de graphes, doit respecter l'objectif d'interface utilisateur privilégiée. Pour cela elle s'appuie sur le squelette d'interface utilisateur (§4 Niveau plate-forme - Squelettes d'interface utilisateur) pour la gestion des événements, des menus et du dialogue.

La description des services d'administration n'est pas l'objet de cette thèse mais l'atelier MARS est conçu pour accueillir ces services ayant une interface utilisateur spécifique. Ces programmes utilisent le gestionnaire des formalismes et accèdent au Gestionnaire de Station Virtuelle. Ils sont les seuls à permettre l'exécution des outils d'administration pour la mise à jour les donnée du GUS.

Nous avons introduit des formalismes d'administration adaptés aux problèmes de gestion de configurations Par exemple, dans le cas de gestion des machines supportant l'atelier, une partie de l'éditeur de graphes peut être utilisé car il est possible de dessiner les différentes machines, les réseaux les reliant et les différentes caractéristiques des machines. Par contre nous avons constaté, en l'expérimentant, que la gestion des menus des programmes d'applications ne devait pas s'effectuer au moyen de l'éditeur de graphes mais par un programme spécialisé. Il faut donc construire des interfaces utilisateur adaptées aux données à gérer (utilisateurs, services, formalismes…).

Nous avons vu dans la partie I que l'atelier de spécification comporte un système expert (Objectif 21). Comme tout système expert, des paquets de règles d'inférence [El Fallah, 1989] doivent être manipulés. Pour permettre un intégration de ce système expert dans l'atelier de spécification distribué, nous devons mettre un éditeur de paquets de règles à la disposition des concepteur d'outils. Cet éditeur doit être lié au GUS pour permettre une utilisation intégrée et distribuée.

9. Conclusion

L'ensemble des composantes que nous avions dégagées dans la partie I sont maintenant intégrées dans l'architecture fonctionnelle de la figure 9. L'analyse que nous avons effectuée dans la partie II a organisé ces composantes en couches et les a décomposées en modules en vue de leur programmation.

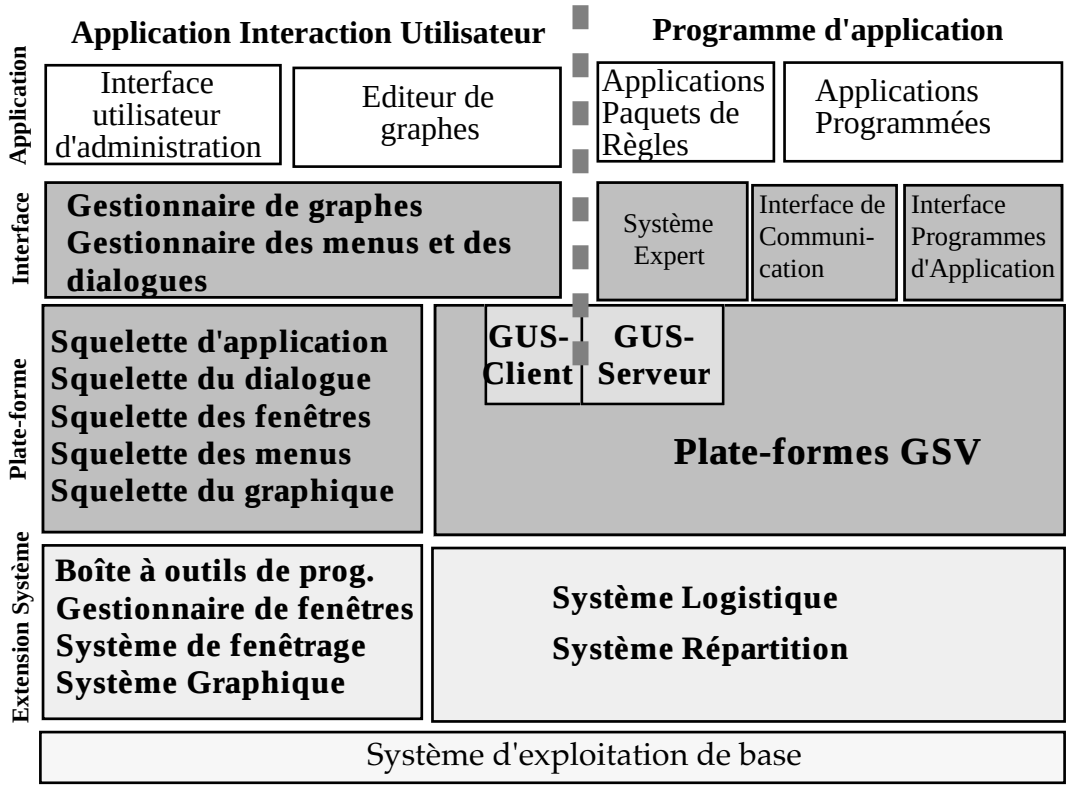


Figure 9: Architecture fonctionnelle complète

Cette architecture est destinée à satisfaire l'ensemble des objectifs de la partie I. Nous pouvons maintenant recapituler les objectifs résultant du cahier des charges de la partie I afin de préciser quels sont les ensembles de modules qui concourent pour les satisfaire.

Formalismes

- Les plates-formes GUS et GSV gérent le multi-formalismes de haut niveau. Elles assurent une gestion globale des données des utilisateurs et intégrent la notion d'environnement multi-utilisateurs.
- Les plates-formes GUS, GSV et les squelettes assurent à l'atelier une ouverture en facilitant l'intégration de nouveaux outils et de formalismes.

Utilisateurs

- L'application d'interaction utilisateur permet l'édition des énoncés et la représentation des résultats des services.
- La double gestion locale et distant du système de fenêtrage de la plate-forme GSV privilégie l'interface utilisateur.
- L'interface utilisateur de l'atelier est homogène et indépendante des programmes d'applications. Elle facilite l'utilisation de l'atelier.

Structure d'accueil

- La plate-forme GSV assure le parallélisme des services par l'utilisation du principe de multi-sessions. La plate-forme GSV rend transparents aux l'utilisateurs les réseaux de machines, elle intégre des mécanismes de distribution des applications sur l'ensemble des machines du site.
- Les services de l'atelier assurent une analyse immédiate des énoncés.
- L'architecture de l'atelier permet de réaliser des services de prototypage. Le niveau plate-forme assure les correspondances entres les énoncés et leurs résultats.

Outils logiciels

- La plate-forme GSV peut exécuter des applications compilées ou interprétées ou appliquer les règles d'un système expert.
- L'architecture intègre un mécanisme de communication entre les applications.