

# X11 Motif modélisé avec UML

## www.thierrycros.net

---

UML modélise les applications que l'on construit. UML visualise aussi des systèmes tiers ou patrimoines.

### Points étudiés

L'environnement graphique X11 modélisé avec UML

## 4.1 Ce qu'est X-Window

X-Window (ou X11) est un environnement graphique multi-fenêtré, basé sur un protocole réseau client / serveur. Cet environnement est indépendant :

Du matériel : d'un écran monochrome bas de gamme à un écran couleur très haute résolution, X11 accepte une vaste gamme de matériels, quel que soit le constructeur

Du système d'exploitation : X11 est l'interface graphique d'Unix, fonctionne également sous VMS de DEC mais n'impose pas ces systèmes

Du réseau : tout comme pour le matériel et le système d'exploitation, le protocole réseau X11 ne contient pas d'éléments spécifiques à TCP/IP ou Decnet.

Pour mieux comprendre ce qu'est X11, nous le modélisons avec UML.

### 4.1.1 Le modèle Client / Serveur

Le premier point important est le modèle Client / Serveur. Contrairement à d'autres environnements graphiques dans lesquels ce modèle d'architecture est mis en œuvre de façon limitée, sur un seul nœud, X11 repose sur un véritable protocole réseau : plusieurs nœuds interviennent dans sa réalisation. Ainsi, une machine peut être spécialisée dans un service qui est l'interface graphique utilisateur. Tout l'intérêt de ce modèle d'architecture informatique réside dans la spécialisation d'une machine, ce qui décharge d'autant les nœuds utilisateurs.

Le modèle Client / Serveur est basé sur un nœud qui a pour rôle la gestion d'une certaine ressource. Généralement, la ressource est une base de données ou un ensemble de

fichiers. Le principe d'une interface graphique basée sur un réseau synchrone est apparu début des années 80.

**N** Le terme « serveur » est souvent employé abusivement. Les expressions « serveur Unix » ou « serveur NT »  
**O** ne veulent pas dire grand chose tant que la ressource servie n'est pas précisée. Un système d'exploitation  
**T** possède ou ne possède pas les sous-systèmes qui lui permettent de jouer le rôle de serveur sur un réseau. Par  
**E** exemple, les drivers réseau sont nécessaires. Le fait de posséder ces ressources ne signifie pas qu'elles sont  
nécessairement mises en œuvre.

Une solution pour ne pas se laisser abuser par ces termes est de se poser la question : quelle est la ressource gérée ? Le serveur est alors le gestionnaire de cette ressource : il y accède physiquement. Enfin, les clients sont par définition les logiciels qui envoient des requêtes au serveur pour utiliser la ressource sans y accéder physiquement.

Au passage, la relation client / serveur offre souvent une abstraction supplémentaire par rapport à la ressource. Ainsi, un serveur de base de données manipule des fichiers (\*.data et \*.index par exemple) ou du moins des données sur disque dans le cas de serveurs optimisés qui court-circuitent le système. Par contre, les clients émettent des requêtes basées sur l'abstraction « table ».

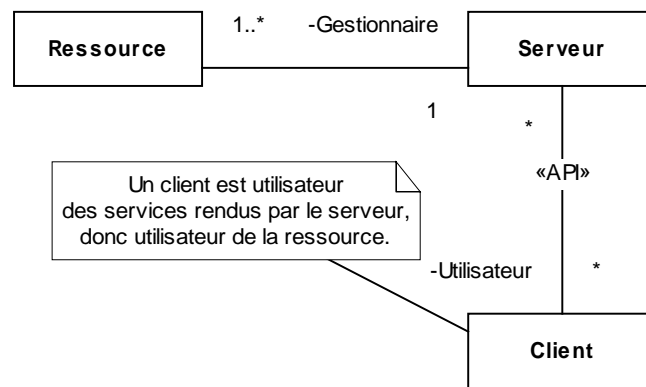


Figure 4-1 : Le modèle Client / Serveur version UML

Ce diagramme de classes indique les relations entre ressources, serveurs et clients. Un logiciel client n'est pas lié aux ressources mais aux serveurs. Généralement, plusieurs clients accèdent à un serveur qui lui-même gère plusieurs ressources. Une interface de programmation d'application (API) relie les clients aux serveurs. Standard Query Language (SQL) est une API utilisée dans le cas de serveur de base de données. Le protocole réseau X11 joue exactement le rôle de SQL, mais dans le cadre de l'interface graphique.

```

// Exemples de requêtes des protocoles SQL et X11 réseau
CREATE TABLE X // création d'un objet de base
CREATE WINDOW // idem
...
INSERT TUPLE INTO X // insertion dans un objet
DRAW LINE X1,Y1,X2,Y2 // idem
  
```

Dans le cas de X11, la ressource est l'ensemble clavier, écran(s), souris. Contrairement à d'autres réalisations client / serveur, l'utilisateur est physiquement *proche* du serveur. Cette caractéristique (qui n'est pas moins client / serveur qu'une autre) rend la localisation du serveur souvent délicate pour un novice. L'utilisateur humain s'apparente au client

informatique qui joue le rôle d'utilisateur (lui aussi...) de la ressource. Or, l'utilisateur humain n'est pas modélisé dans le client / serveur.

#### 4.2.2 Architecture matérielle logicielle de X11

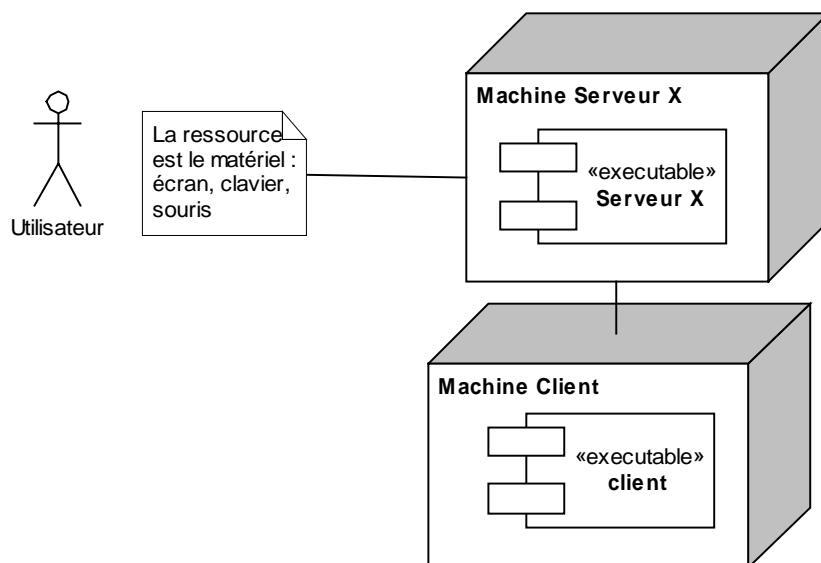


Figure 4-2 : Architecture matérielle – logicielle X-Window

Dans ce diagramme de déploiement, la communication entre les classes de nœuds « Machine Serveur X » et « Machine Client » représente tout réseau ou protocole réseau sur lequel X11 s'appuie : TCP/IP, Decnet, etc.

#### 4.2.3 Plusieurs types de serveurs

Plusieurs machines supportent des serveurs X.

Stations de travail Unix : Sun, HP, GNU/Linux, FreeBSD, etc ont la particularité d'héberger le serveur X et des clients, tous se cotoient en tant que process du système.

Terminaux X : ce sont des machines spécialement conçues pour X-Window qui ne possèdent pas de système d'exploitation mais exécutent un flot qui est le serveur X.

Applications des environnements propriétaires Macintosh ou Microsoft. Ces logiciels offrent une passerelle entre Unix et d'autres systèmes.

Chaque solution offre des avantages mais vient avec son lot d'inconvénients. Une station est autonome et le logiciel libre XFree86 est un bon moyen de s'initier à cet environnement. Il permet d'ailleurs la construction et le déploiement d'applications professionnelles. Un terminal X est dépendant d'une ou plusieurs machines Unix (ou VMS) sur le réseau. Toutefois son coût n'est pas comparable à celui d'une station Unix commerciale. Par ailleurs, ces terminaux sont dépendants de la mémoire vive installée, contrairement à une station qui bénéficie des performances du système Unix et de son mécanisme de swapping. Enfin, une solution de type Wintel offre l'intégration de deux mondes, bien qu'elle se traduise souvent par des caractéristiques graphiques très moyennes, comparées aux standards des stations Unix.

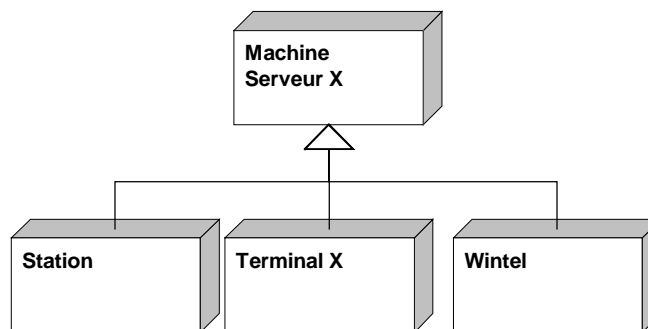


Figure 4-3 : Différents serveurs X

#### 4.2.4 Motif et X11

Les solutions offertes par Unix sont ouvertes. Ainsi, X11 est la couche graphique de base, basée sur le protocole réseau qui sépare les traitements (le client) de l'interface utilisateur (le serveur). Mais X11 n'impose pas le « look & feel » (aspect général et gestion des fenêtres) de l'interface. Au contraire, certains environnements Unix (dont les logiciels libres GNU/Linux par exemple) proposent à l'utilisateur de nombreuses interfaces telles que Motif, Open Look, AfterStep, KDE (clone de Common Desktop Environment) ou Gnome. L'utilisateur retrouve ainsi des aspects et comportements proches de ce qu'il peut rencontrer sur d'autres plates-formes ou bien bénéficie des caractéristiques de gestionnaires de fenêtres particulièrement sophistiqués. Par ailleurs, toutes les réalisations ne nécessitent pas de gestion de fenêtres telles que *resizing* ou simplement déplacement sur l'écran.

Motif de l'Open Software Foundation (OSF) est le gestionnaire de fenêtre (window manager) standard sous Unix. Common Desktop Environment (CDE) est l'environnement bureautique standard de fait, reconnaissable par son bandeau en bas d'écran qui offre l'accès aux écrans virtuels. Il offre un gestionnaire de fichiers, une aide hypertexte, un éditeur graphique, de nombreux utilitaires.

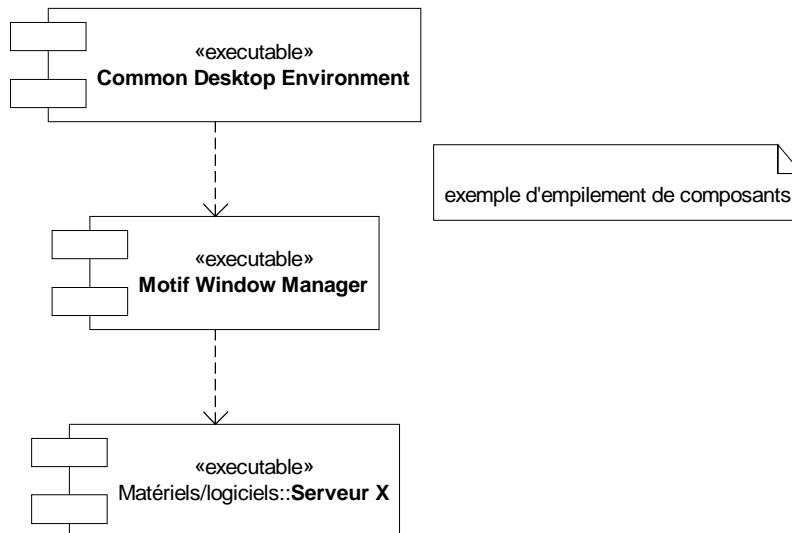


Figure 4-4 : Liaisons entre CDE et Motif

## 4.2 Développement d'application

### 4.2.1 Les bibliothèques

Motif est un gestionnaire de fenêtres. C'est aussi une bibliothèque de développement d'applications graphiques sous Unix. L'élément de base est alors la *widget*, objet graphique *du client* à aspect et comportement prédéfini. Un `PushButton` est ainsi un bouton poussoir Motif. La Toolkit est une bibliothèque qui gère les widgets (création, relation avec les fenêtres dans le serveur...) alors que Motif est une bibliothèque de quelques dizaines de widgets (de types d'objets graphiques).

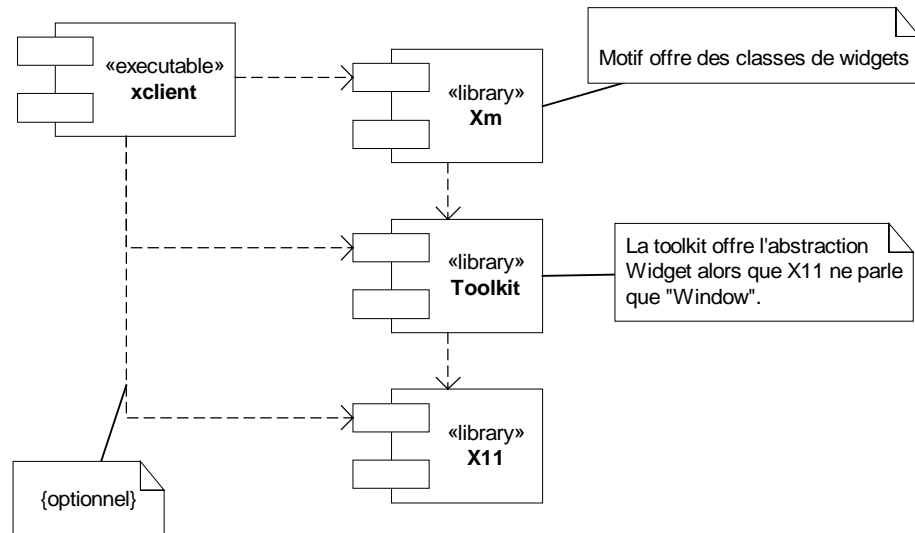


Figure 4-5 : Dépendances entre exécutable et bibliothèques

Un client X dépend de la bibliothèque Xm (Motif) car il utilise les types de widgets de cette bibliothèque. Elle est aussi la plus élevée sémantiquement. De ce fait, une opération *sera a priori* plus simple à programmer à ce niveau.

Un client X dépend de la bibliothèque Xt (Toolkit). Comme son nom l'indique, il s'agit d'une véritable boîte à outils destinée au programmeur d'application graphique. La connexion avec le serveur, la gestion des événements qu'il envoie sont autant de fonctionnalités de cette bibliothèque.

Un client X dépend de la bibliothèque X11 (X-Window). Cette dépendance est optionnelle car tous les clients ne font pas directement appel aux services de cette bibliothèque. La gestion du graphisme, au sens tracé, est un exemple d'utilisation.

#### 4.2.2 Aspect dynamique

Un client se connecte à un serveur, crée les objets graphiques nécessaires (ses widgets), demande à la Toolkit de créer les fenêtres correspondantes dans le serveur et termine dans une boucle infinie de gestion d'événements provenant du serveur. Ces événements sont typiquement issus de la souris ou du clavier. Pour simplifier, nous confondrons dans la suite les bibliothèques Motif, Toolkit et X11 car l'objectif est de cerner les interactions entre la partie qui nous incombe (la partie applicative) et les bibliothèques utilisées.

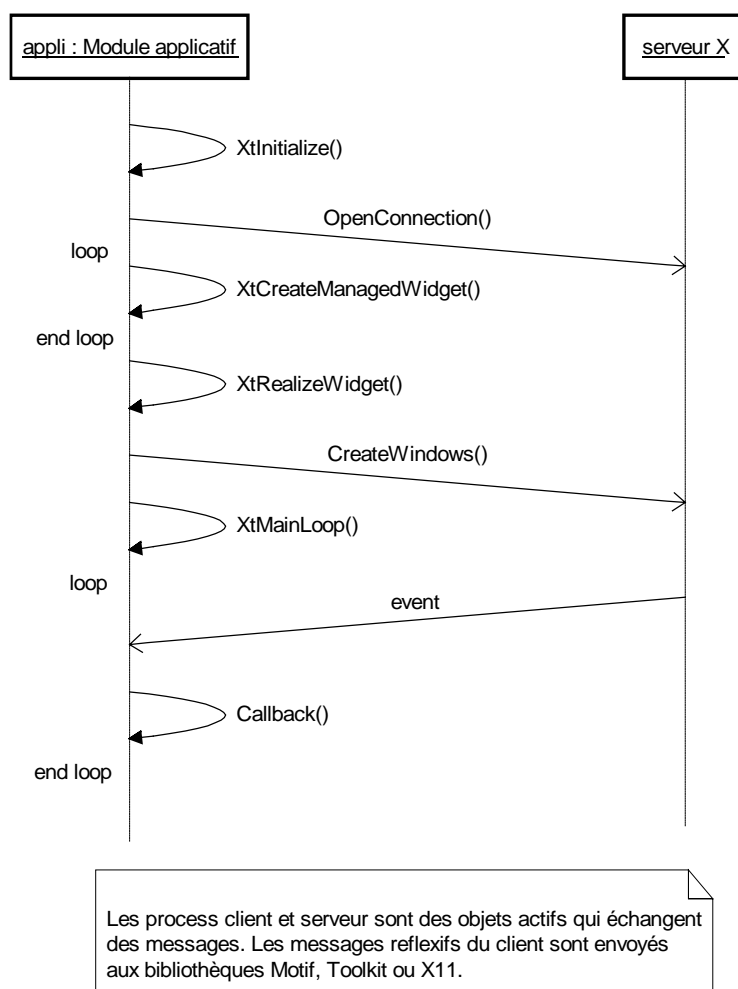


Figure 4-6 : Diagramme de principe : trame d'un client Motif

Dans ce diagramme, les flèches obliques visualisent le délai de propagation sur le réseau.

### 4.2.3 Programmation événementielle

La conception et la programmation sont constituées de deux phases distinctes : création des objets graphiques, les widgets qui forment les menus, les grilles et développement des fonctions de réaction aux événements qui surviennent dans les fenêtres, ce sont les callbacks ou fonctions réflexes. Ce type de fonction est automatiquement déclenché (par la Toolkit) lorsque l'événement correspondant survient dans le serveur. Dans un premier temps, le programmeur enregistre cette fonction de traitement à l'aide de `XtAddCallback()`. Dans les diagrammes qui suivent, les interactions sont échangées entre composants.

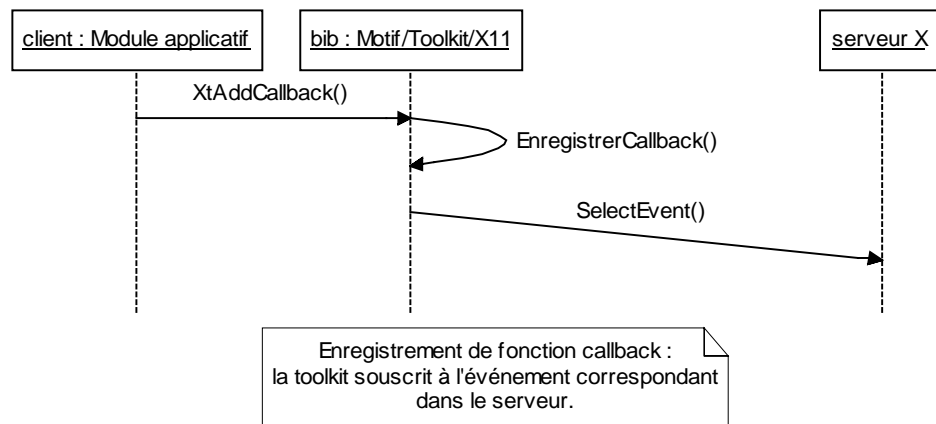


Figure 4-7 : Enregistrement de callback

Lorsque l'événement survient dans le serveur, plus précisément dans la fenêtre associée à la widget pour laquelle le callback est enregistré, un événement est envoyé au client. Cet événement est reçu par la Toolkit, puis réparti vers la widget concernée sous la forme de callback déclenché. L'objet graphique widget mémorise les callbacks (fonctions de traitement en réponse à un événement) à déclencher dans une table associée au type d'événement géré.

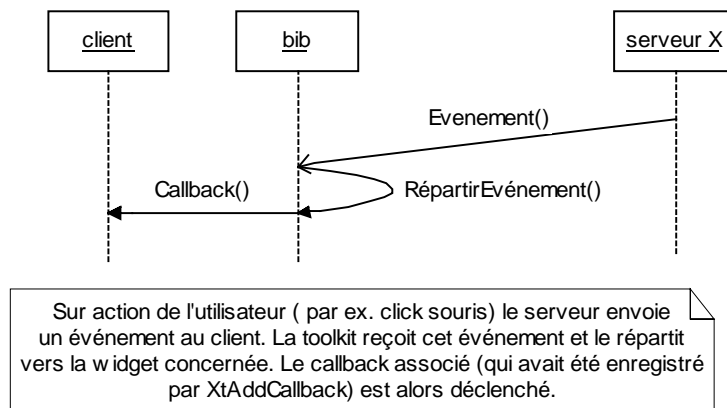


Figure 4-8 : Déclenchement de callback

Parmi les callbacks déclenchés, certains sont prédéfinis d'autres sont enregistrés par l'application. Une modification de l'affichage d'une widget sur réception d'un événement correspond au déclenchement de callback prédéfini.

#### 4.2.4 Arborescence de widgets

Les widgets d'un client forment une arborescence depuis une racine généralement nommée « toplevel » (l'équivalent de « / » dans une arborescence Unix).

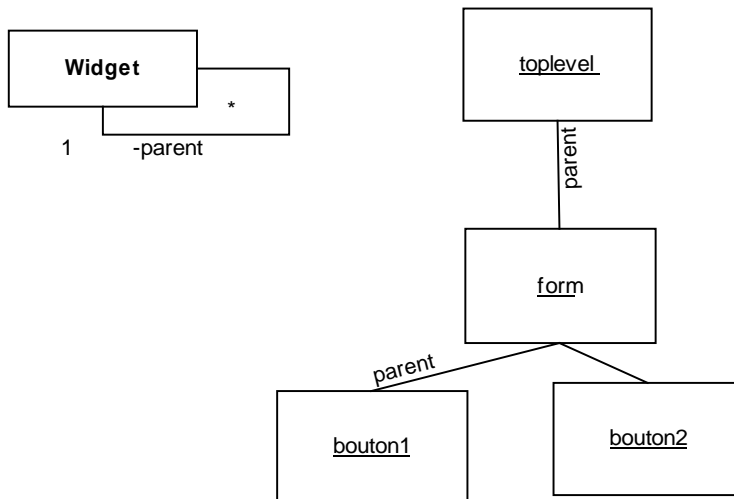
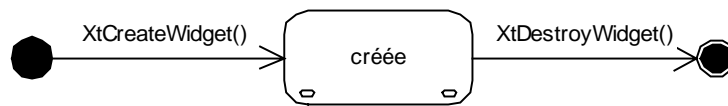


Figure 4-9 : Arborescence de widgets

L'arborescence de widgets est maintenue au fur et à mesure de la création des widgets. La widget toplevel joue le rôle d'interface avec l'environnement du client dans le serveur. Sa widget fille correspond à la fenêtre telle que l'utilisateur la perçoit. Enfin, les boutons correspondent aux éléments « utiles » de l'interface.

### 4.3 État d'une widget

Un objet graphique du client X doit être « réalisé » : la fenêtre correspondante dans le serveur est alors créée. Par ailleurs, une widget est « managée » autrement dit gérée par sa widget parente, ce qui correspond au calcul de son aspect, sa position, etc.



L'état créé est constitué de plusieurs sous-états parallèles.

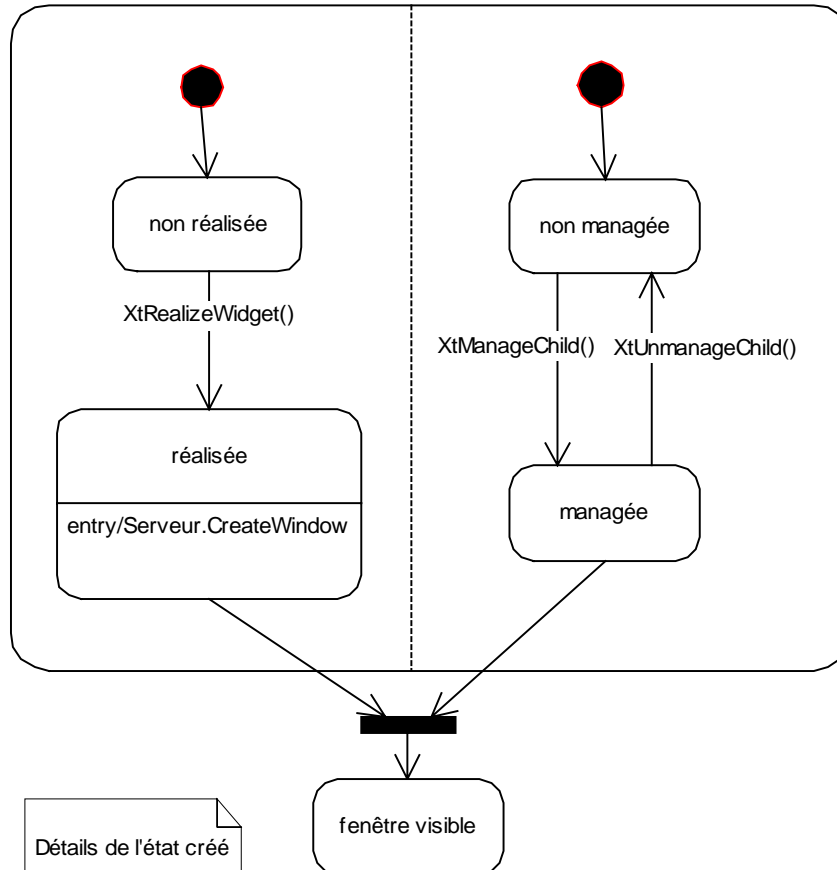


Figure 4-10 : État / transition simplifié d'une widget

Etant donné qu'une widget doit être « managée » pour être visible, il est possible de jouer sur la visibilité à partir de son management. Ainsi, les fonctions `XtManageChild()` et `XtUnmanageChild()` permettent de rendre visible ou invisible un objet graphique. Le nom de ces fonctions suggère que l'action de gestion (management) d'une widget est à la charge de la widget parente. Ainsi, `XtmanageChild(bouton1)` correspond à une demande de

management de la widget `bouton1` mais destinée en fait à la widget parente, par exemple `form` dans le diagramme 4-9.

<b>X11 MOTIF MODÉLISÉ AVEC UML .....</b>	<b>1</b>
<b>4.1 Ce qu'est X-Window .....</b>	<b>1</b>
4.1.1 Le modèle Client / Serveur .....	1
4.2.2 Architecture matérielle logicielle de X11 .....	3
4.2.3 Plusieurs types de serveurs .....	3
4.2.4 Motif et X11 .....	4
<b>4.2 Développement d'application.....</b>	<b>5</b>
4.2.1 Les bibliothèques .....	5
4.2.2 Aspect dynamique.....	6
4.2.3 Programmation événementielle.....	7
4.2.4 Arborescence de widgets .....	9
<b>4.3 état d'une widget .....</b>	<b>9</b>



