DÉVELOPPEMENT D'UN SYSTÈME DE GESTION DE WORKFLOWS DISTRIBUÉ

par

Lotfi ben Othman

mémoire présenté au Département de mathématiques
et d'informatique en vue de l'obtention du grade de maître ès sciences (M.Sc.)

FACULTÉ DES SCIENCES
UNIVERSITÉ DE SHERBROOKE

Sherbrooke, Québec, Canada, août 2000

National Library of Canada
Acquisitions and Bibliographic Services
395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque nationale du Canada
Acquisitions et services bibliographiques
395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

Our file Notre référence

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-67689-7

Canada

Le 21/8/2000 , le jury suivant a accepté ce mémoire dans sa version finale.
date

Président-rapporteur:	M. Roger N'Kambou Département de mathématiques et d'informatique
Membre:	M. Richard Egli Département de mathématiques et d'informatique
Membre:	M. Richard St-Denis Département de mathématiques et d'informatique
Membre:	M. Naoufel Kraiem Université Libre de Tunis

Sommaire

Les nouvelles générations de systèmes d'information intègrent les systèmes d'applications aux systèmes de gestion de workflows. La modélisation de procédures d'affaires ou de workflows est un domaine récent. Les chercheurs dans ce domaine visent à développer des méta-modèles de workflows, à intégrer différentes implémentations de workflows et à augmenter la flexibilité des systèmes de gestion de workflows.

Le travail de recherche présenté dans ce mémoire porte sur le développement d'un système de gestion de workflows distribué. Dans ce travail, nous appliquons une méthode orientée objets (OMT) pour l'analyse et la conception d'un tel système. Puis, nous construisons le système suivant une architecture client/serveur à base d'objets distribués. En plus, nous réalisons un client workflows et un serveur workflows. Ces deux modules sont développés avec le langage de programmation Java. La communication entre les objets distants est gérée par le bus RMI. La base de données est construite à l'aide du SGBD d'Oracle et les accès aux données sont réalisés en utilisant JDBC.

Remerciements

Je tiens à exprimer mes vifs remerciements à mon directeur de recherche, le professeur Richard St-Denis, pour m'avoir accueilli dans son équipe. Ses conseils avisés et ses remarques pertinentes m'ont facilité la réalisation de ce travail, dont l'aboutissement est pour moi l'occasion de lui exprimer toute ma gratitude.

Je remercie Monsieur Naoufel Kraiem, professeur à l'Université Libre de Tunis et mon co-directeur, pour sa patience, ses directives et ses nombreuses séances d'assistance.

Je remercie aussi ma famille et mes amis pour leur soutien moral et financier.

Table des matières

Liste des abréviations

API	Application Programming Interface
CORBA	Common Object Request Broker Architecture
HTTP	Hyper Text Transport Protocol
IDL	Interface Definition Language
IP	Internet Protocol
JDBC	Java Database Connectivity
OMG	Object Modeling Group
OMT	Object Modeling Technique
RMI	Remote Method Invocation
RPC	Remote Procedure Call
SGBD	système de gestion de bases de données
SI	système d'information
SQL	Structured Query Language
SWAP	Simple Workflow Access Protocol
TCP	Transport Control Protocol
WAPI	Workflow Application Programming Interface
Wf-XML	Workflow eXtended Markup Language
WfMC	Workflow Management Coalition
WfMS	Workflow Management System
XML	eXtended Markup Language

Liste des tableaux

Liste des figures

Introduction

Dans toute organisation, les individus se partagent le travail et coopèrent afin d'atteindre un certain nombre d'objectifs. Dans le domaine du traitement de l'information, une organisation possède un système d'information. Selon Rivard et Talbot [20], un tel système

> « n'est qu'un ensemble d'activités qui saisissent, stockent, transforment et diffusent des données sous un ensemble de contraintes appelées l'environnement du système. Ces activités sont concrétisées par des personnes, des procédures, des équipements et des données. »

Les systèmes d'information peuvent être formels ou informels. Dans les systèmes d'information formels, il existe des règles et des méthodes de travail documentées (e.g., fiche client, procédure de vente). Dans les systèmes informels (e.g., les notes de service, les réunions), il n'existe pas de règles précises. Avec cette division des systèmes d'information, les méthodologies traditionnelles d'analyse et de conception des systèmes d'information ne s'intéressent qu'aux informations formelles [20]. Ces méthodologies projettent les informations sur trois axes qui sont les entrées du système, les sorties du système et les processus qui représentent le traitement de l'information.

Un système d'information peut être divisé en un ensemble de sous-systèmes d'applications et de sous-systèmes de workflows [11]. Les sous-systèmes d'applications représentent des tâches à effectuer. Les applications automatisent des tâches et manipulent des données (stockage, traitement et transformation). Les sous-systèmes de workflows représentent la

définition des données à manipuler dans les procédures de travail, la circulation des données dans les organisations et l'automatisation des procédures ainsi définies.

Les workflows peuvent être gérés par un système de gestion de workflows. McCarty et Blustein définissent un système de gestion de workflows comme étant [9] :

> « a proactive system which manages the flow of work among participants according to a defined procedure consisting of a number of tasks. It coordinates user and system participants, together with the appropriate data resources which may be accessible directly by the system or off-line, to achieve defined objectives by set deadlines. »

En effet, un workflow est, d'une part, une collection d'activités ou de tâches regroupées au moyen d'interactions et, d'autre part, une collection d'agents (appelés aussi des acteurs) qui sont des personnes ou des machines qui exécutent des tâches [13].

Dans ce mémoire, un workflow est une représentation formelle d'une procédure de travail. Dans une organisation, il peut y avoir autant de workflows que de procédures de travail.

Un système de gestion de workflows distribué est un système informatique exploitable dans un environnement homogène ou hétérogène (types de matériel différents, types de système de communication différents). Un tel système est composé de ressources physiques (matériel) et logiques (processus) géographiquement dispersées. Ces ressources sont reliées par un système de communication qui leur permet d'échanger des données en vue de coopérer à une tâche commune. Ces ressources doivent avoir un haut degré de cohésion et de transparence.

Les systèmes de gestion de workflows distribués qui gèrent des workflows offrent trois possibilités pour les représenter. Ces possibilités se résument comme suit :

- représentation d'un workflow en utilisant un langage déclaratif qui permet au concepteur de décrire l'organisation de l'entreprise, les données à manipuler et les procédures à utiliser [1] ;

- représentation d'un workflow avec un système de gestion de workflows, puis génération d'un programme exécutable [8] ;
- représentation d'un workflow avec un système de gestion de workflows, qui reste la base du fonctionnement, et enregistrement du workflow sous la forme de propriétés dans une base de données.

Dans le cadre de ce projet, nous abordons le développement d'un système de gestion de workflows distribué selon la troisième possibilité. Ce système doit garantir la cohérence des données. Il doit aussi fonctionner sur différentes plates-formes.

Problème

L'objectif du présent mémoire est l'analyse, la conception et l'implémentation d'un système de gestion de workflows distribué.

Le premier problème abordé dans ce mémoire consiste en la proposition d'un méta-modèle workflow. Ce méta-modèle doit permettre la structuration et l'organisation des données manipulées par les workflows. Ce méta-modèle doit avoir les caractéristiques suivantes :

- la réutilisation des composants,
- la modularité de la solution,
- la facilité de la maintenance du système.

Selon Klamma et al. [10], les données d'un workflow peuvent être organisées en cinq types (voir la figure 1). Le premier type est appelé *agent*. Il représente un rôle organisationnel. Un agent est appelé à réaliser une tâche manuellement ou automatiquement en utilisant un outil. Le deuxième type est appelé *outil*. Les données de ce type décrivent un outil utilisé par un agent pour accomplir une tâche. Le troisième type est appelé *procédure*. Une procédure est un ensemble d'activités que l'agent est appelé à réaliser selon des circuits de données et des règles de traitement de données. Le quatrième type est

appelé *objet*. Un objet représente les données manipulées par une application du système d'information. Le cinquième type est appelé *activité*. Une activité est une division de la procédure. En effet, chaque procédure est composée d'un ensemble d'activités. Le rôle de chaque activité est la transformation d'un objet par un agent.

Le deuxième problème abordé dans ce mémoire consiste en la spécification d'un système de gestion de workflows distribué flexible [28]. Pour être flexible, le système de gestion de workflows distribué doit permettre la modification de la conception d'un workflow et le changement du fonctionnement d'un workflow. Par exemple, l'administrateur peut résoudre le problème de cycle infini lors de l'exécution d'un workflow [17].

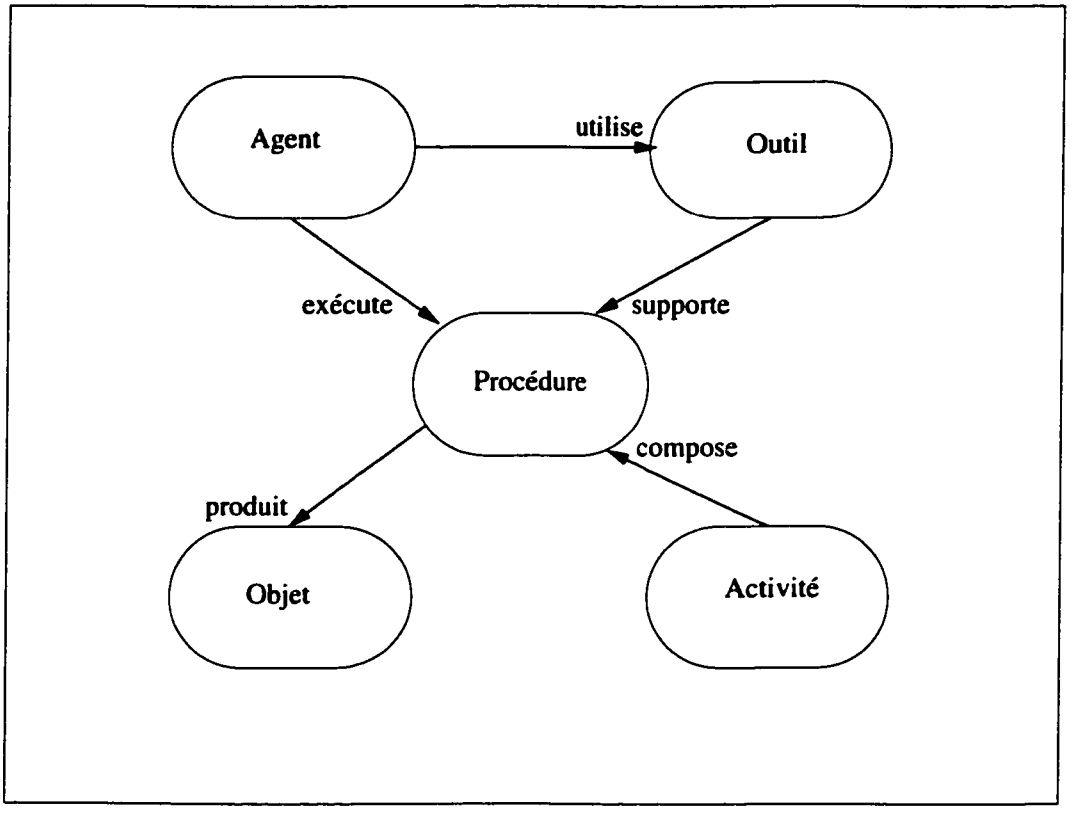

FIG. 1 – *Types de données manipulées par un système de gestion de workflows distribué*

Le troisième problème abordé dans ce mémoire consiste en la décomposition du

système en un ensemble de composants, puis l'implémentation de ces composants selon une architecture client/serveur dans un environnement distribué. Ces composants peuvent fonctionner avec un bus de communication dont l'organisation leur est inconnue. L'abstraction de la communication permet au système proposé d'utiliser des moyens de communication hétérogènes. Cette communication doit être organisée selon une architecture client/serveur.

Le quatrième problème abordé dans ce projet est l'implémentation d'un système de gestion de workflows distribué qui respecte les caractéristiques suivantes: la portabilité des composants, l'indépendance du système vis-à-vis du bus de communication et la persistance des données. En effet, étant donné la diversité du matériel et des systèmes d'exploitation, ainsi que le besoin d'intégrer des ressources (applications de gestion, bases de données) dans un système de gestion de workflows distribué, les composants doivent être indépendants des spécificités des ressources.

Le système de gestion de workflows distribué doit utiliser un mécanisme de stockage et de manipulation des données workflow. Ce mécanisme peut être un système de gestion de fichiers partagés, un mécanisme de gestion d'objets persistants ou un système de gestion de bases de données. Dans l'implémentation de notre système de gestion de workflows distribué, nous devons choisir un mécanisme de gestion de données workflow et nous devons organiser les données workflow en conséquence.

L'objectif principal de ce mémoire est donc de proposer, de spécifier, de décomposer en un ensemble de composants et d'implémenter un système de gestion de workflows distribué.

Méthodologie

Pour mettre en place un méta-modèle d'un système de gestion de workflows distribué qui respecte les caractéristiques introduites dans la section précédente, nous avons choisi

d'utiliser une méthode orientée objets.

Le sujet principal des méthodes orientées objets est la programmation. À l'inverse, la méthode OMT [21] couvre l'ensemble du processus de développement informatique. La méthode OMT se base sur la construction des objets du monde réel et elle combine à la fois des structures de données et des comportements des objets. La technique de modélisation par objets (OMT) intègre trois vues de modélisation des systèmes: le modèle objets, le modèle dynamique et le modèle fonctionnel. Le modèle objet représente les aspects statiques et structurels ainsi que les données du système. Le modèle dynamique représente les aspects comportementaux et de contrôle d'un système. Le modèle fonctionnel représente les aspects des fonctions et de transformation. Ce dernier est éliminé car son principal rôle est la vérification des deux autres modèles.

Dans ce mémoire nous avons retenu la méthode OMT. Cette méthode permet l'analyse, la conception et l'implémentation d'un système selon une approche intégrée. En particulier, le modèle dynamique facilite l'esquisse des événements et des actions d'un système.

Dans la phase d'analyse, nous développons un modèle objet et un modèle dynamique du système de gestion de workflows distribué. Le modèle objet représente l'aspect statique du système, alors que le modèle dynamique représente son aspect comportemental. Dans cette modélisation, nous devons prendre en considération le besoin d'avoir un système de gestion de workflows distribué flexible. Pour ceci, nous devons séparer le mode exécution et le mode conception d'un workflow [28].

Dans la phase de conception, nous décomposons le système en cinq composants: le bus de communication, le gestionnaire de workflows, l'outil de définition d'un workflow, les données workflow et les clients du système de gestion de workflows distribué. Dans l'étude des fonctionnalités des composants, nous étudions certaines alternatives. Dans cette phase, nous sommes aussi intéressés à la conception d'une architecture du système de gestion de workflows distribué. La figure 2 illustre une architecture simplifiée. Les

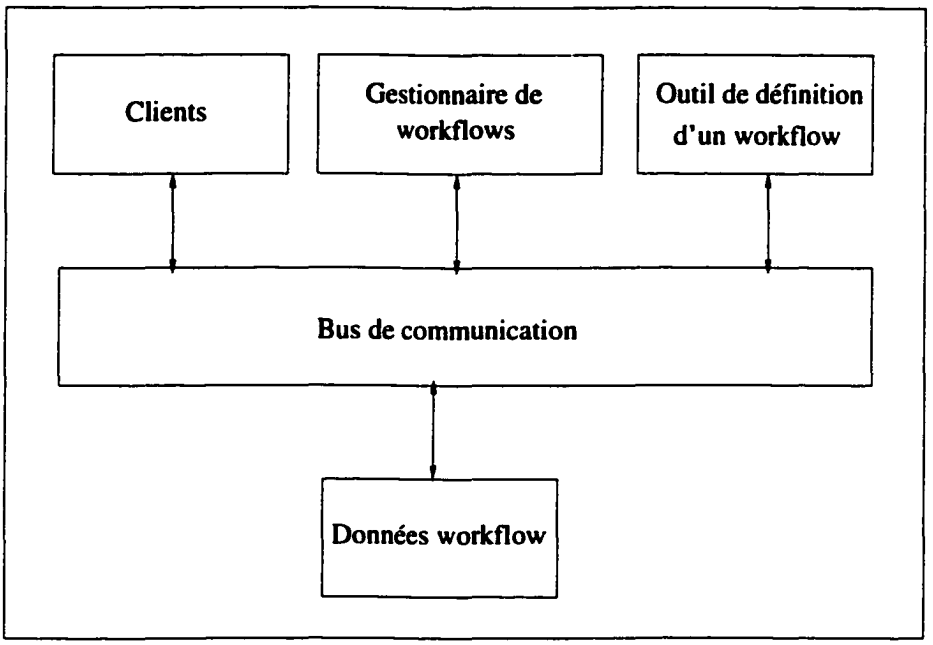

FIG. 2 – *Architecture du système de gestion de workflows distribué*

composants de ce système communiquent entre eux via un bus de communication. Ce bus peut être un protocole de communication, un bus de type CORBA [4, 23] ou un bus RMI [25].

Dans la phase d'implémentation, nous devons faire certains choix qui concernent :

- le langage de programmation qui permet la portabilité de notre système,
- le bus de communication entre les objets,
- le mécanisme de stockage et de manipulation des données.

Résultat

Le principal résultat de ce travail est un système de gestion de workflows distribué fonctionnel développé selon une approche orientée objets. Il a été programmé dans le langage de programmation Java. Les objets distants communiquent entre eux via le bus RMI et les données sont gérées par le SGBD d'Oracle. Les accès à la base de données

sont réalisés en utilisant JDBC. Ce système, bien qu'il supporte les principaux aspects présentés dans ce mémoire, doit être avant tout vu comme un prototype exploitant plusieurs facilités offertes par le langage de programmation Java.

Organisation du mémoire

Le reste de ce mémoire est composé de quatre chapitres. Dans le premier chapitre, nous exposons des travaux sur la modélisation des systèmes de gestion de workflows. Dans le deuxième chapitre, nous abordons l'analyse d'un système de gestion de workflows distribué. Nous construisons le modèle objet et le modèle dynamique de notre système. Le troisième chapitre est consacré à la conception du système de gestion de workflows distribué. Nous décomposons le système en un ensemble de composants et nous étudions les alternatives de fonctionnement de ces composants. Nous concevons aussi une architecture du système de gestion de workflows distribué. Dans le quatrième chapitre, nous décrivons une implémentation du système de gestion de workflows distribué. Nous présentons d'abord les moyens logiciels utilisés. Nous nous intéressons ensuite à la structure des données manipulées par le système et à la structure des modules à mettre en place. À la fin de ce mémoire, nous proposons trois annexes contenant respectivement un glossaire, la description des tables et une étude de cas.

Chapitre 1

Aperçu du domaine

L'automatisation des procédures de travail est le centre d'intérêt de plusieurs chercheurs et d'acteurs de l'industrie informatique à cause de son apport économique et organisationnel ainsi que de la complexité de l'information manipulée.

Dans ce qui suit nous allons présenter quatre travaux qui s'intéressent à la spécification des composants d'un système de gestion de workflows. Dans la première section, nous présentons des travaux relatifs au développement d'un protocole propre aux workflows. Dans la deuxième section, nous présentons les travaux de l'organisme WfMC pour la spécification d'un système de gestion de workflows. Dans la troisième section, nous présentons les travaux de l'organisme OMG pour la spécification d'interfaces qui permettent la communication entre des systèmes de gestion de workflows via le bus CORBA. Dans la dernière section, nous présentons un travail pour la mise en oeuvre d'une communication workflow de qualité.

1.1 Le protocole SWAP

Swenson [27] a défini un système de gestion de workflows comme étant un système qui définit et modélise les procédures de travail, puis synchronise l'exécution des activités

propres à un workflow. Leur but est la spécification d'un protocole appelé SWAP (Simple Workflow Access Protocol). Ce protocole exécute, contrôle et administre les opérations workflow [19]. Ces opérations se résument en la création d'un bon de travail, au lancement et à l'arrêt de l'exécution d'un bon de travail, en la récupération des résultats d'exécution, de l'état ou de l'historique d'un bon de travail. Ce protocole permet aussi l'interprétation d'un workflow, le contrôle de l'instanciation des procédures, le séquencement des activités, l'ajout des bons de travail à l'utilisateur. SWAP offre un mécanisme d'échange de données workflow entre les différents systèmes de gestion de workflows.

Afin de structurer les données manipulées par SWAP, ces auteurs ont proposé une spécification des ressources gérées par un système de gestion de workflows sous forme de classes. Ces classes concernent la définition, l'instanciation et le contrôle d'une procédure de travail, la définition et le contrôle des activités qui forment une procédure de travail, puis les corbeilles des participants. Deux types de données sont associés à ces classes. Le premier type traite la manipulation des données telle que la recherche et le contrôle des procédures. Le deuxième type concerne les états des activités et les ordres pour le changement des états [19].

SWAP a été spécifié comme un protocole asynchrone qui utilise le protocole HTTP comme couche inférieure, ce qui lui permet d'être intégré à Internet. Ce protocole intègre une en-tête composée d'un ensemble de champs qui permettent l'échange de données entre les composants d'un workflow. Pour la représentation des données, SWAP utilise la norme XML (eXtended Markup Language). Dans SWAP, les auteurs ont spécifié l'enregistrement de l'historique ce qui permet la récupération de données workflow après panne.

1.2 La Workflow Management Coalition

En janvier 1995, le Workflow Management Coalition (WfMC) a publié son premier document qui couvre les concepts, les terminologies et une structure générale d'un système de gestion de workflows [6]. Dans ce document, cet organisme a spécifié les principaux composants d'un système de gestion de workflows et les fonctionalités de ces composants.

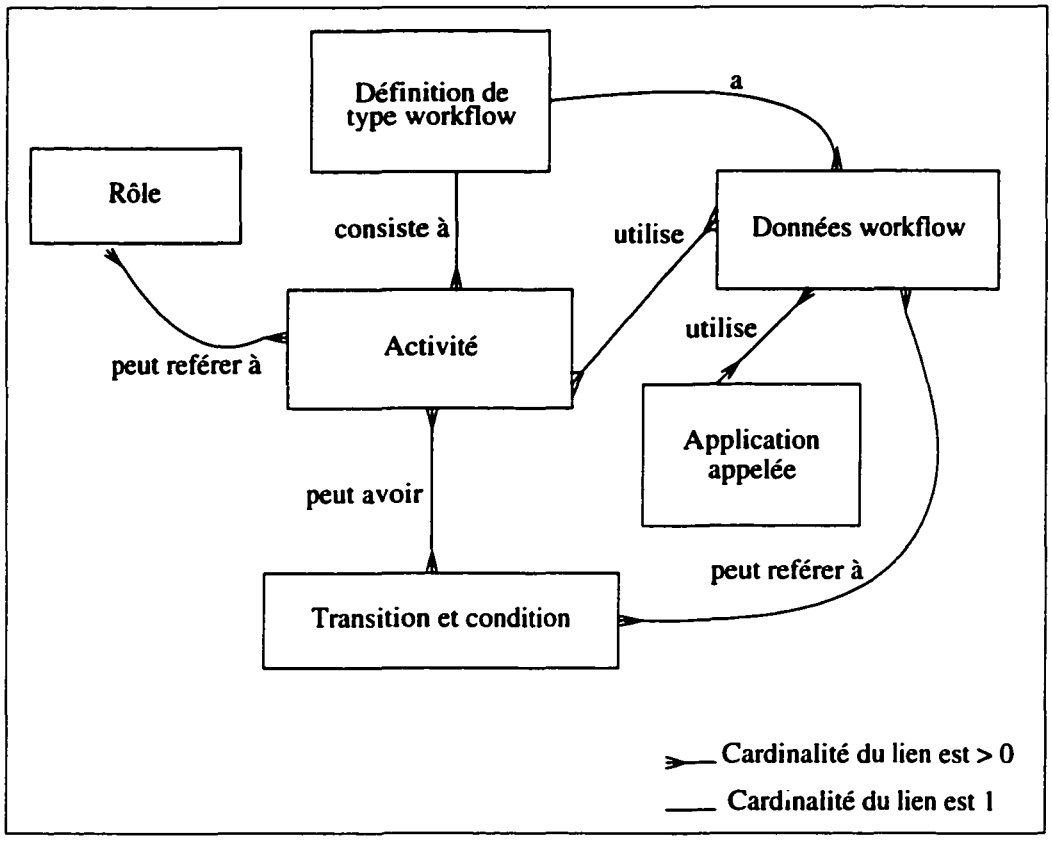

FIG. 3 – *Méta-modèle de base de la WfMC tiré de [6]*

Afin de spécifier les interfaces des composants d'un système de gestion de workflows, le WfMC a construit un méta-modèle qui modélise les données traitées par un tel système. Tel qu'illustré à la figure 3, ce méta-modèle est composé principalement de l'entité *rôle* qui modélise les participants d'un système de gestion de workflows, les rôles organisationnels et les responsabilités des participants, de l'entité *activité* qui modélise les bons de travail

à réaliser dans une procédure de travail et de l'entité *transition et condition* qui indique les règles de séquencement du flux de travail [6]. Cet organisme a proposé un ensemble de spécifications qui permettent la communication entre différents systèmes de gestion de workflows. Cette spécification englobe la définition de structures de données, des fonctions et des procédures utilisées par un système de gestion de workflows distribué.

En juillet 1999, le WfMC a proposé l'intégration de XML (eXtended Markup Language) comme un moyen d'échange de données entre les différents systèmes de gestion de workflows [24]. Puis en janvier 2000, l'organisme WfMC a spécifié un langage de description de données workflow qui se base sur XML [30]. Ce langage est indépendant des particularités des mécanismes d'implémentation. Ce langage est appelé Wf-XML et il fonctionne avec le protocole HTTP.

Lawrence [12] a détaillé davantage les spécifications et les *Workflow Application Programming Interfaces* (WAPI) construites par le WfMC. Dans son travail, il a décrit des applications possibles d'un workflow. Puis, il a présenté une esquisse des besoins d'un système de gestion de workflows en termes d'algorithme de transactions. Ces algorithmes de transactions doivent prendre en considération la complexité des structures de données workflows.

1.3 L'Object Management Group

En juillet 1998, l'Object Management Group (OMG) [16] a mis en place une spécification d'un gestionnaire de workflows pour un environnement distribué. L'objet de cette spécification est la réponse aux besoins d'avoir une interopérabilité entre les différentes implémentations d'objets de workflows qui fonctionnent dans un environnement CORBA. Selon cet organisme, les interfaces spécifiées sont suffisamment complètes pour supporter différents domaines (gestion, scientifique, médical).

L'OMG a utilisé le modèle de référence d'un système workflow établi par le WfMC.

Il a construit ensuite un modèle conceptuel d'un gestionnaire de workflows. Selon la figure 4 ce modèle est composé d'un ensemble de classes qui modélisent les événements, les activités, les procédures de travail et les ressources d'un workflow. Ce modèle a été par la suite raffiné pour inclure d'autres classes. Puis l'OMG a porté l'attention sur la spécification des interfaces des objets d'un système de gestion de workflows distribué avec le langage IDL. Cette spécification contient les attributs, les méthodes, les états et les relations entre les objets. Les données spécifiées dans ces interfaces sont les données qui peuvent être échangées entre différents workflows.

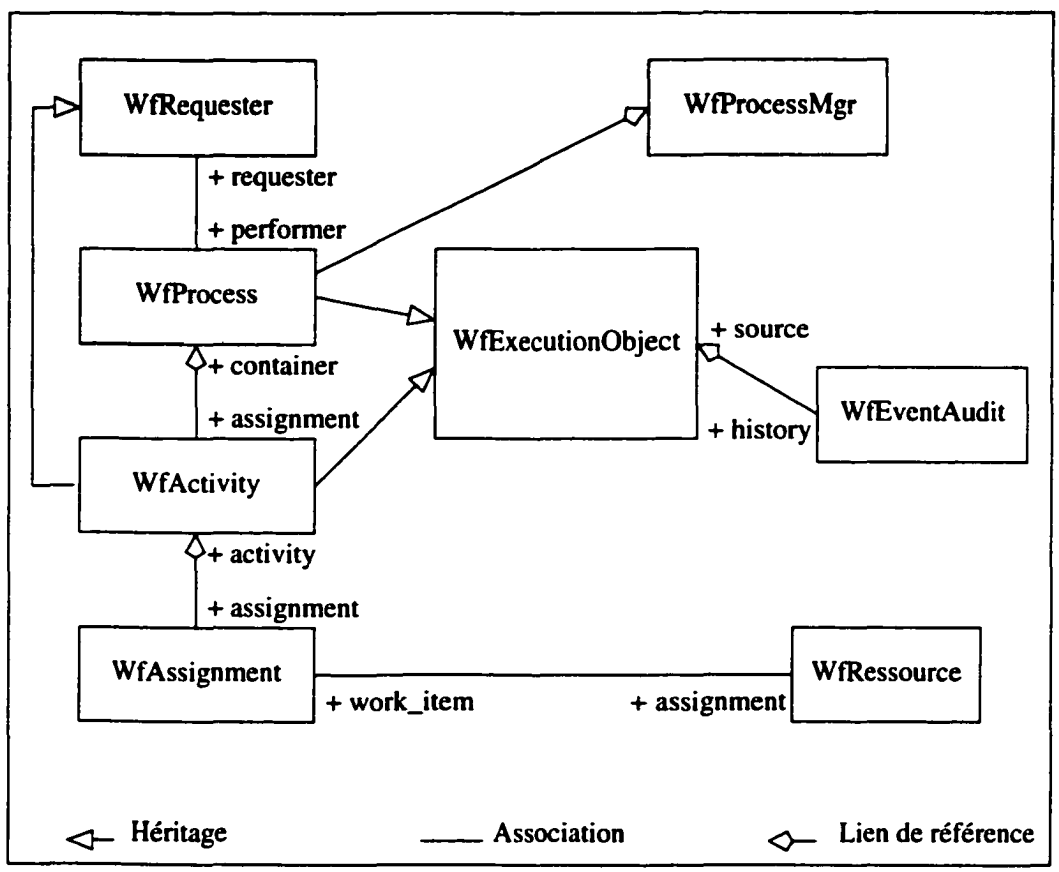

FIG. 4 - *Modèle du système de gestion de workflows de l'OMG tiré de [16]*

L'OMG a approfondi le travail sur le service de communication entre des workflows, mais il n'a pas résolu les autres problèmes d'un workflow distribué tels que le service de

transactions ou le service de récupération après panne [15]. L'OMG doit donc prendre en considération la spécification d'algorithmes pour supporter les transactions workflow. Il doit aussi spécifier des algorithmes pour la récupération après panne.

1.4 Le système de gestion de workflows *mobile*

Le système de gestion de workflows distribué regroupe un ensemble de serveurs et d'applications distribués qui fonctionnent selon une architecture client/serveur. La qualité de communication dans ce système se mesure par le respect de certaines propriétés qui sont : la durabilité de la transaction, l'unicité de l'exécution d'une transaction workflow et le type de la communication (synchrone ou asynchrone). Le système de gestion de workflows distribué *mobile* [22] utilise une technique qui offre une qualité dans la communication entre les objets. Cette technique repose sur l'utilisation de copies des objets manipulés par une transaction. Les auteurs ont spécifié un mécanisme de récupération de workflows après une panne. Avec ce mécanisme le client ou le serveur précise s'il veut faire une récupération des données workflow avant une panne (utiliser la technique des points de reprise) au après une panne (ignorer les requêtes qui ont échouées). Le type de récupération dépend d'un ensemble de paramètres tels que le type de communication ou l'attente de résultat. L'objet du service est la synchronisation des opérations de récupération d'une manière transparente au client ou au serveur. Les auteurs de cette architecture l'ont comparée avec d'autres techniques de communication objet comme CORBA, RPC et les transactions distribuées. Ils estiment que leur technique demande moins d'effort de programmation dans la mise en place d'une communication client/serveur dans un système de gestion de workflows.

Chapitre 2

Analyse

Dans l'introduction, nous avons défini un système de gestion de workflows distribué comme étant un système qui permet la modélisation et l'exécution de workflows. La solution que nous proposons repose sur la modélisation d'un tel système avec la méthode OMT [21]. Dans cette solution, nous proposons un modèle basé principalement sur cinq concepts à savoir : les procédures, les activités, les objets, les outils et les agents.

L'objectif de ce chapitre, qui porte sur l'analyse et qui constitue la première phase de la méthode OMT, est la description la plus précise possible d'un système de gestion de workflows distribué. Cette phase est initialement basée sur la définition du problème et la proposition d'une vue d'ensemble des concepts du système. Elle produit deux modèles, à savoir le modèle objets et le modèle dynamique.

2.1 Description du système

Le système de gestion de workflows distribué que nous développons doit permettre la formalisation et l'exécution de workflows dans un système distribué.

2.1.1 La formalisation d'un workflow

La formalisation d'un workflow est une description des flux de travail et des règles qui organisent ces flux. En effet, cette formalisation repose sur l'identification des activités qui forment une procédure de travail , des données et de leurs transformations et, enfin, des participants à cette procédure.

Les données workflow décrivent les flux de travail et les règles qui organisent ces flux. Ces données peuvent être organisées en cinq types (voir la figure 1).

Pour modéliser les règles qui organisent ces flux, nous choisissons la notion de transition. Chaque transition contient une condition pour son activation, un événement qui déclenche la transition et des données liées à la transition.

Dans la modélisation, nous utilisons le terme participant comme synonyme d'agent, ressource comme synonyme d'outil. Les procédures et les activités conservent leur nomination.

2.1.2 L'exécution d'un workflow

Considérons le workflow *circuit de vente* présenté en annexe C. L'organisme qui applique ce workflow peut l'utiliser plusieurs fois pendant une période. La trace de chaque utilisation de ce workflow doit exister pour toutes les ventes. En mode exécution d'un workflow, nous ajoutons trois classes à savoir : instance de procédure, instance d'activité et instance de donnée. Ces classes permettent la création d'une instance de procédure, d'activité ou de donnée, le lancement d'une instance de procédure, le lancement d'une instance d'activité et la valorisation d'une instance de donnée.

L'exécution d'un workflow est l'instanciation des propriétés du workflow, la gestion des objets qui modélisent le workflow et la détermination du séquencement de ses activités.

Dans notre travail, nous nous intéressons particulièrement à l'exécution d'un workflow. Nous fournissons aussi un ensemble de méthodes qui permettent la formalisation

d'un workflow. Le scénario type de l'exécution d'un workflow se résume en la séquence suivante :

- récupérer une session sur un gestionnaire de workflows
- récupérer une liste des travaux,
- choisir un bon de travail,
- récupérer les données sur une activité à exécuter,
- créer une instance d'activité à exécuter,
- demander l'allocation des ressources,
- exécuter une instance d'activité,
- informer le système de la fin d'une activité,
- libérer les ressources,
- récupérer les données d'une transition,
- évaluer une condition,
- choisir la prochaine activité,
- créer le bon de travail de la prochaine activité.

À ce scénario nous pouvons ajouter quelques requêtes, à savoir :

- créer une instance d'une procédure,
- arrêter l'exécution d'une instance de procédure,
- arrêter l'exécution d'une instance d'activité.

2.2 Modèle objets

Selon OMT [21], un modèle objets est une description de la structure statique des classes et de leurs relations. Il est représenté par un graphe d'objets. Dans ce modèle, les noeuds sont des classes et les arcs sont des associations entre les classes. Pour construire un modèle objets, nous avons d'abord réalisé un dictionnaire de données, puis nous avons

identifié les classes et les attributs, ensuite nous avons ajouté les associations entre les classes, enfin, nous avons organisé et simplifié les classes en utilisant l'héritage.

La représentation des données de passage entre les activités d'une procédure peut se faire avec un diagramme de transitions ou un diagramme d'états. Un diagramme d'états est un formalisme graphique qui permet la modélisation des comportements de systèmes complexes [5]. Cette méthode de représentation est utilisée par les approches de conception objets [3]. Dans un tel formalisme, un noeud représente un état d'un objet et une flèche représente une transition qui permet le passage d'un état à un autre. Une transition englobe l'événement déclencheur de la transition, la condition qui permet l'activation de la transition lors de l'occurrence de l'événement et les données internes à la transition. Un diagramme d'états se distingue d'un diagramme de transitions par le fait qu'il est une extension de ce dernier. En effet, un diagramme d'états est un diagramme de transitions auquel on ajoute la notion de niveau de profondeur, d'orthogonalité et de communication [3]. Le niveau de profondeur est le détail d'un comportement d'un objet. L'orthogonalité est le passage à deux états différents en parallèle. Dans ce mémoire, nous avons choisi les diagrammes de transitions pour faciliter le travail.

Dans notre modèle, nous définissons deux types d'énumérations : **state** et **category**. Le **state** représente l'état que peut avoir une procédure ou une activité. L'énumération du type **state** est : *Started*, *Suspended*, *Completed*, *Aborted*, *Activated* et *Terminated*. Le type **category** représente la catégorie d'une donnée manipulée. L'énumération du type **category** est : *Data* et *Program*.

Notre modèle objets est composé de treize classes. Ces classes représentent quatre types de données :

- les données sur un workflow et les activités qui le composent,
- les données sur le passage entre des activités,
- les données sur les ressources utilisées,
- les données sur les participants.

La figure 5 illustre le modèle objets de notre système de gestion de workflows. Il comporte les classes suivantes : **WfProcessMgr**, **WfParticipant**, **WfProcess**, **WfTransition**, **WfProcessInstance**, **WfActivity**, **WfActivityInstance**, **WfEvent**, **WfData**, **WfCondition**,**WfResource**, **WfDataInstance** et **WfWorkItem**.

Dans notre modèle, nous décrivons l'aspect statique d'un système de gestion de workflows distribué, en distinguant deux modes : le mode conception et le mode exécution [28]. Le mode conception est celui dans lequel le concepteur formalise un workflow. Le mode exécution est celui dans lequel le participant instancie un workflow. Dans le mode conception, le développeur conçoit les propriétés d'un workflow. Dans le mode exécution le participant ou le système de gestion de workflows instancie les propriétés d'un workflow. Cette séparation est concrétisée par la création des classes **WfProcess**, **WfActivity** et **WfDataInstance** pour le mode conception et l'utilisation des classes **WfProcessInstance**, **WfActivityInstance** et **WfDataInstance** pour le mode exécution. Les classes **WfActivityInstance** et **WfDataInstance** sont en relation avec la classe **WfProcessInstance**. L'utilisation de deux modes de travail au niveau du système de gestion de workflows permet au système de créer plusieurs instances d'un même workflow et d'avoir leur trace d'exécution.

Dans le modèle objets, un objet de la classe **WfProcessMgr** gère plusieurs workflows et chaque workflow est composé d'une seule procédure. Une procédure est composée d'un ensemble d'activités et d'un ensemble de transitions. Chaque transition est composée d'un événement et d'une condition. Elle utilise deux activités, une activité source et une activité destination. Une activité utilise une ressource et elle est appliquée à un participant. Chaque condition est composée de deux données. Un bon de travail réfère à un participant. Il utilise une ressource et une instance d'activité.

Dans le reste de cette section, nous présentons les classes qui forment le modèle, ainsi que leurs attributs et leurs méthodes. Dans cette description, nous excluons les constructeurs des classes puisque chaque constructeur accepte essentiellement comme

arguments tous les attributs de la classe correspondante. Les nominations utilisées sont en langue anglaise.

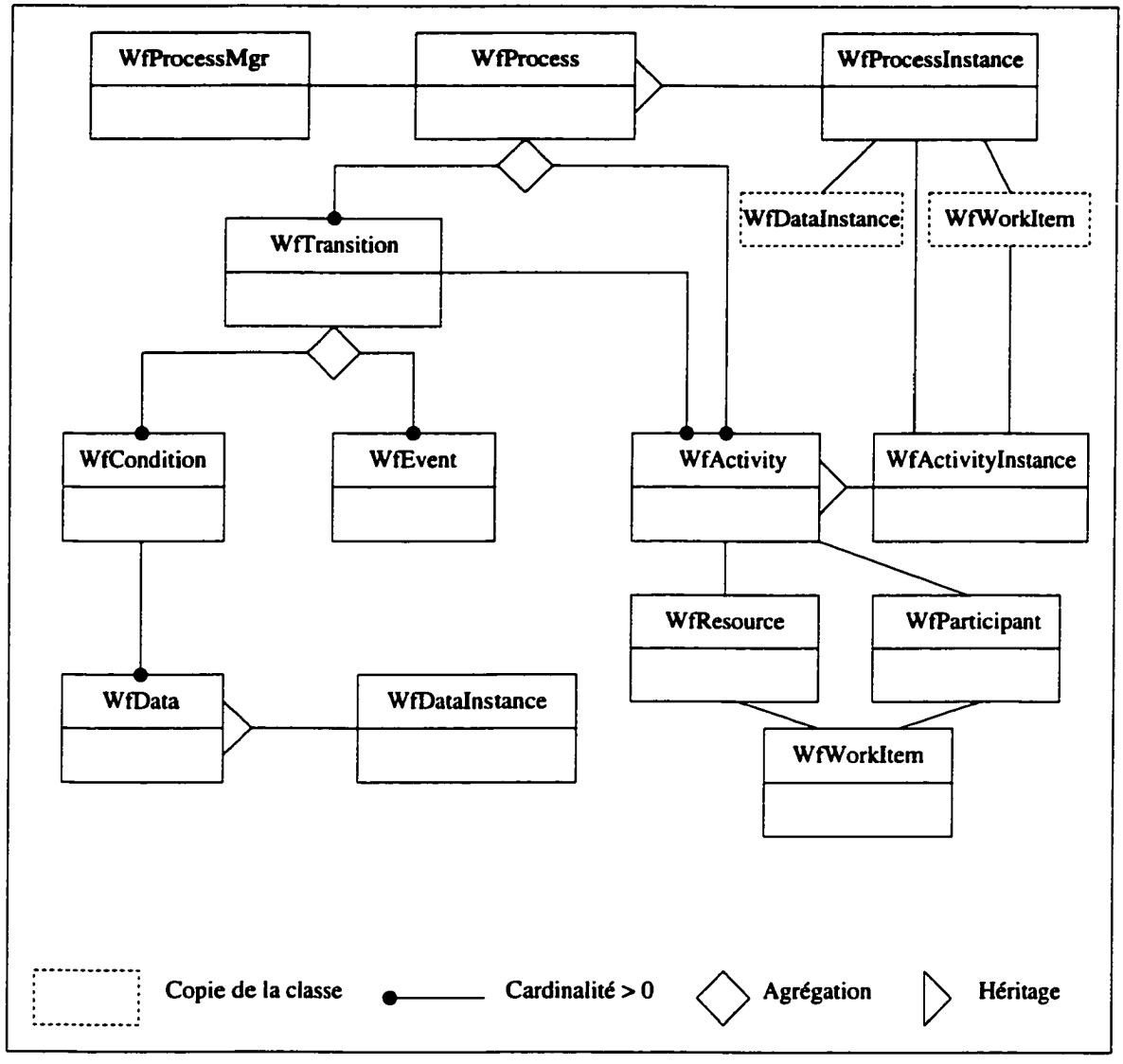

FIG. 5 – *Modèle objets du système de gestion de workflows*

2.2.1 La classe WfActivity

La classe **WfActivity** modélise les activités d'une procédure en mode conception. Une activité est une étape dans une procédure. Chaque activité a besoin de ressources pour supporter l'exécution de la procédure.

TAB. 1 – *Attributs de la classe* **WfActivity**

Nom	Type	Description
ActivityId	`integer`	identification de l'activité
ActivityName	`string`	nom de l'activité
ActivityDescription	`string`	description de l'activité

Les attributs de la classe **WfActivity** sont décrits dans le tableau 1. Cette classe comporte quatre méthodes.

La méthode `SetActivity()` permet l'insertion d'une activité dans la base de données. Cette méthode n'a pas de paramètres puisque les données à enregistrer ont été fournies lors de sa création, c'est-à-dire lors de l'instanciation d'un objet de la classe.

La méthode `GetActivity(string` *name*`)` permet la récupération des données d'une activité, celle ayant comme nom *name*. Les données récupérées sont les valeurs des attributs de l'activité, c'est-à-dire *ActivityId* et *ActivityDescription*.

La méthode `GetActivity(integer` *id*`)` permet la récupération des données d'une activité, celle ayant comme identifiant *id*. Les données récupérées sont les valeurs des attributs de l'activité, c'est-à-dire *ActivityName* et *ActivityDescription*.

La méthode `Instantiate(integer` *process_instance_id*`)` crée une instance d'activité en prenant en considération que cette instance est en relation avec une instance de procédure, celle ayant comme identifiant *process_instance_id*. Après, elle insère cette instance dans la base de données. La méthode `Instanciate` permet d'avoir plusieurs instances d'une même activité.

2.2.2 La classe WfActivityInstance

Cette classe regroupe les données d'une instance d'activité. Une activité est instanciée avec la méthode `Instanciate` de la classe **WfActivity**.

Un objet de la classe **WfActivityInstance** peut avoir un état parmi les états suivantes: *Started*, *Suspended*, *Activated*, *Completed*, ou *Terminated*.

TAB. 2 – *Attributs de la classe* **WfActivityInstance**

Nom	Type	Description
ActivityInstanceId	`integer`	identité de l'instance d'activité
ActivityState	`integer`	état de l'instance d'activité

Les attributs de la classe **WfActivityInstance** sont décrits dans le tableau 2. Cette classe comporte sept méthodes.

La méthode `Start()` change l'état de l'objet en *Started* qui est l'état de départ, puis elle enregistre le changement d'état dans la base de données.

La méthode `Suspend()` change l'état de l'objet en *Suspended*. Ce changement d'état ne peut s'effectuer que si l'état courant est *Activated*. Après, le changement d'état la méthode enregistre le nouvel état dans la base des données.

La méthode `Notify()` change l'état de l'objet en *Activated*. Ce changement d'état ne peut s'effectuer que si l'état courant est *Started* ou *Suspended*. Après, le changement d'état, la méthode enregistre le nouvel état dans la base de données.

La méthode `Abort()` change l'état de l'objet en *Terminated* parce qu'une ressource est perdue. Ce changement d'état ne peut s'effectuer que si l'état courant est *Activated*. Après le changement d'état, la méthode enregistre le nouvel état dans la base de données.

La méthode `Release()` change l'état de l'objet en *Terminated*. Ce changement d'état ne peut s'effectuer que si l'état courant est *Completed*. Après le changement d'état, la méthode enregistre le nouvel état dans la base de données.

La méthode `Allocate()` change l'état de l'objet en *Completed*. Ce changement d'état ne peut s'effectuer que si l'état courant est *Activated*. Après le changement d'état, la méthode enregistre le nouvel état dans la base de données.

La méthode `SetActivityInstance()` permet l'insertion d'une instance d'activité dans la base de données. Cette méthode n'a pas de paramètres puisque les données à enregistrer ont été fournies lors de sa création.

2.2.3 La classe WfCondition

Cette classe modélise une expression logique et permet l'évaluation de cette expression. Le valeur retournée par l'évaluation permet de prendre une décision sur le bon de travail suivant à construire.

TAB. 3 – *Attributs de la classe* **WfCondition**

Nom	Type	Description
ConditionName	`string`	nom de la condition
ConditionId	`integer`	identifiant de la condition
Criterion	`string`	critère de comparaison (<, >, =)

Les attributs de la classe **WfCondition** sont décrits dans le tableau 3. Cette classe comporte trois méthodes.

La méthode `SetCondition()` permet l'insertion d'une condition dans la base de données. Cette méthode n'a pas de paramètres puisque les données à enregistrer ont été fournies lors de sa création.

La méthode `GetCondition(string` *name*`)` permet la récupération des données d'une condition, celle ayant comme nom *name*. Les données récupérées sont les valeurs des attributs de la condition, c'est-à-dire *ConditionId* et *Criterion*.

La méthode `GetCondition(integer` *id*`)` permet la récupération des données d'une condition, celle ayant comme identifiant *id*. Les données récupérées sont les valeurs des

attributs de la condition, c'est-à-dire *ConditionName* et *Criterion.*

2.2.4 La classe WfData

Cette classe modélise les données utilisées dans une transition. Cette classe n'est qu'une nomination d'un objet. Par exemple, dans le workflow *circuit de vente*, présenté en annexe C, le *Quota* constitue une donnée.

Les attributs de la classe **WfData** sont décrits dans le tableau 4. Cette classe comporte trois méthodes.

La méthode `SetData()` permet l'insertion d'une donnée workflow dans la base de données. Cette méthode n'a pas de paramètres puisque les données à enregistrer ont été fournies lors de sa création.

La méthode `GetData(string` *name*`)` permet la récupération des données d'une donnée workflow, celle ayant comme nom *name*. Les données récupérées sont les valeurs des attributs de la donnée workflow, c'est-à-dire *DataId*, *DataDescription*, *DataType* et *DataValue.*

La méthode `GetData(integer` *id*`)` permet la récupération des données d'une donnée workflow, celle ayant comme identifiant *id.* Les données récupérées sont les valeurs des attributs de la donnée workflow, c'est-à-dire *DataName*, *DataDescription*, *DataType* et *DataValue.*

TAB. 4 – *Attributs de la classe* **WfData**

Nom	Type	Description
DataName	`string`	nom de la donnée
DataId	`integer`	identifiant de la donnée
DataDescription	`string`	description textuelle de la donnée
DataType	`integer`	type de la donnée, à savoir entier (0), texte (1), booléen (2), enregistrement (3)
DataValue	`string`	valeur générique ou par défaut de la donnée

2.2.5 La classe WfDataInstance

Cette classe modélise des instances de données utilisées dans les activités. Cette classe est une sous-classe de **WfData**. Elle est en relation avec la classe **WfProcessInstance**.

TAB. 5 – *Attributs de la classe* **WfDataInstance**

Nom	Type	Description
DataInstanceId	`integer`	identifiant de l'instance de donnée
DataValue	`string`	valeur de l'instance de donnée

Les attributs de la classe **WfDataInstance** sont décrits dans le tableau 5. Cette classe comporte trois méthodes.

La méthode `SetDataInstance()` permet l'insertion d'une instance de donnée dans la base de données. Cette méthode n'a pas de paramètres puisque les données à enregistrer ont été fournies lors de sa création.

La méthode `GetDataInstance(integer` *id*`)` récupère une donnée d'une instance de donnée, celle ayant comme identifiant *id*. La donnée récupérée est l'attribut de l'instance de la donnée, c'est-à-dire *DataValue*.

La méthode `GetDataInstance(integer` *data*`, integer` *process_instance*`)` permet la récupération des données d'une instance de donnée, celle référant à la donnée *data*, mais qui est propre à l'instance de procédure *process_instance*. Les données récupérées sont les attributs de l'instance de donnée, c'est-à-dire *DataInstanceId* et *DataValue*.

2.2.6 La classe WfEvent

Cette classe modélise les événements d'un workflow. Par exemple, dans le workflow *circuit de vente*, présenté en annexe C, l'*Acceptation du règlement* par le client constitue un événement.

TAB. 6 – *Attributs de la classe* **WfEvent**

Nom	Type	Description
EventName	`string`	nom de l'événement
EventId	`integer`	identifiant de l'événement
EventType	`integer`	type de l'événement (interne/externe)

Les attributs de la classe **WfEvent** sont décrits dans le tableau 6. Cette classe comporte quatre méthodes.

La méthode `SetEvent()` permet l'insertion d'un événement dans la base de données. Cette méthode n'a pas de paramètres puisque les données à enregistrer ont été fournies lors de sa création.

La méthode `GetEvent(string` *name*`)` permet la récupération des données d'un événement, celui ayant comme nom *name*. Les données récupérées sont les valeurs des attributs de l'événement, c'est-à-dire *EventId* et *EventType*.

La méthode `GetEvent(integer` *id*`)` permet la récupération des données d'un événement, celui ayant comme identifiant *id*. Les données récupérées sont les valeurs des attributs de l'événement, c'est-à-dire *EventName* et *EventType*.

La méthode `SignalEvent(integer` *id*`, integer` *process_instance*`)` force la construction d'un bon de travail. L'événement à forcer est celui ayant comme identifiant *id* et l'instance de procédure à laquelle nous appliquons l'événement est celle ayant comme identifiant *process_instance*.

2.2.7 La classe WfParticipant

La classe **WfParticipant** représente les acteurs impliqués dans des activités. Elle représente un client du système de gestion de workflows distribué.

Les attributs de la classe **WfParticipant** sont décrits dans le tableau 7. Cette classe comporte cinq méthodes.

TAB. 7 – *Attributs de la classe* **WfParticipant**

Nom	Type	Description
ParticipantId	`integer`	identifiant du participant
ParticipantName	`string`	nom du participant
ParticipantDescription	`string`	description du participant
ParticipantRole	`string`	définition du rôle du participant

La méthode `SetParticipant()` permet l'insertion d'un participant dans la base de données. Cette méthode n'a pas de paramètres puisque les données à enregistrer ont été fournies lors de sa création.

La méthode `GetParticipant(string` *name*`)` permet la récupération des données d'un participant, celui ayant comme nom *name*. Les données récupérées sont les valeurs des attributs du participant, c'est-à-dire *ParticipantId*, *Description* et *Role*.

La méthode `GetParticipant(integer` *id*`)` permet la récupération des données d'un participant, celui ayant comme identifiant *id*. Les données récupérées sont les valeurs des attributs du participant, c'est-à-dire *Name*, *Description* et *Role*.

La méthode `StartSession()` crée une session de travail. D'abord, elle demande à l'unique objet de la classe **WfProcessMgr** de créer une session de travail sur le serveur. Si la réponse est positive, cette méthode récupère les bons de travail du participant.

La méthode `WfActivityAssociated(integer` *id*`)` permet l'exécution d'une instance d'activité identifiée par *id*. Avec cette méthode, le participant donne l'ordre au serveur d'exécuter une instance d'activité.

2.2.8 La classe WfProcess

Cette classe formalise une procédure. Elle est représentée comme étant un ensemble de coordinations d'activités (en parallèle et en série). Ces activités sont connectées dans le but d'atteindre les objectifs d'une procédure.

Les attributs de la classe **WfProcess** sont décrits dans le tableau 8. Cette classe

TAB. 8 – *Attributs de la classe* **WfProcess**

Nom	Type	Description
ProcessId	`integer`	identifiant de la procédure
ProcessName	`string`	nom de la procédure
ProcessDescription	`string`	description de la procédure
ProcessVersion	`integer`	version de la procédure

comporte quatre méthodes.

La méthode `SetProcess()` permet l'ajout d'une procédure dans la base de données. Cette méthode n'a pas de paramètres puisque les données à enregistrer ont été fournies lors de sa création.

La méthode `GetProcess(string` *name*`)` permet la récupération des données d'une procédure, celle ayant comme nom *name*. Les données récupérées sont les valeurs des attributs de la procédure, c'est-à-dire *ProcessId*, *ProcessDescription* et *ProcessVersion*.

La méthode `GetProcess(integer` *id*`)` permet la récupération des données d'une procédure, celle ayant comme identifiant *id*. Les données récupérées sont les valeurs des attributs de la procédure, c'est-à-dire *ProcessName*, *ProcessDescription* et *ProcessVersion*.

La méthode `Instanciate(integer` *id*`)` permet la création d'une nouvelle instance de procédure identifiée par *id*. Cette méthode crée un objet **WfProcessInstance** dont l'identifiant est celui attribué par un séquenceur. Puis, elle enregistre cette instance dans la base de données.

2.2.9 La classe WfProcessInstance

Cette classe représente une instance de procédure. Elle représente la classe **WfProcess** en mode exécution. En effet, le système de gestion de workflows distribué peut lancer plusieurs instances d'une procédure. Par exemple, nous pouvons avoir plusieurs instances de la procédure *circuit de vente* et chacune traite la vente de certains produits

à un client.

Un objet de la classe **WfProcessInstance** peut avoir un état parmi trois états possibles qui sont *Enabled*, *Disabled* et *Terminated*. Au lancement, une instance de procédure prend l'état *Enabled*, puis change d'état selon le diagramme d'états de la figure 8 (voir la section 2.3.2).

TAB. 9 – *Attributs de la classe* **WfProcessInstance**

Nom	Type	Description
ProcessInstanceId	`integer`	identification de l'instance de procédure
ProcessState	`integer`	état de l'instance de procédure

Les attributs de la classe **WfProcessInstance** sont décrits dans le tableau 9. Cette classe comporte sept méthodes.

La méthode `Start()` change l'état d'une instance de procédure en *Enabled*, puis enregistre le changement d'état dans la base de données.

La méthode `Disable()` change l'état d'une instance de procédure en *Disabled*. Ce changement ne peut être réalisé que si l'état courant est *Enabled*. Puis, la méthode enregistre le changement d'état dans la base de données.

La méthode `Enable()` change l'état d'une instance de procédure en *Enabled*. Ce changement ne peut être réalisé que si l'état courant est *Disabled*. Puis, la méthode enregistre le changement d'état dans la base de données.

La méthode `Terminate()` change l'état d'une instance de procédure en *Terminated*. Puis, elle enregistre le changement d'état dans la base de données.

La méthode `SetProcessInstance()` permet l'insertion d'une instance de procédure dans la base de données. Cette méthode n'a pas de paramètres puisque les données à enregistrer ont été fournies lors de sa création.

La méthode `GetProcessInstance(integer` *id*`)` permet la récupération de la donnée d'une instance de procédure, celle ayant comme identifiant *id*. La donnée récupérée est la valeur de l'attribut de l'instance de procédure, c'est-à-dire *ProcessState*.

La méthode **GetNextWorkItem(integer** *activity*, **integer** *process_instance*) permet la création d'un bon de travail. Ce bon de travail utilise une activité, celle ayant comme identifiant *activity*. Le bon de travail à créer appartient à une instance de procédure, celle ayant comme idetifiant *process_instance*.

2.2.10 La classe WfProcessMgr

Cette classe modélise des gestionnaires de procédures relatives à des workflows. Elle est responsable de l'allocation des sessions. Chaque gestionnaire est identifiable par son nom.

TAB. 10 – *Attributs de la classe* **WfProcessMgr**

Nom	Type	Description
ProcessMgrId	**integer**	identification du gestionnaire
ProcessMgrName	**string**	nom du gestionnaire
ProcessMgrDescription	**string**	description du gestionnaire
ProcessMgrVersion	**integer**	version du gestionnaire

Les attributs de la classe **WfProcessMgr** sont décrits dans le tableau 10. Cette classe contient une seule méthode qui est **GetSessionId(string** *participant_name*). L'objet de cette méthode est la vérification de la validité du participant dont le nom est *participant_name*. Si le participant est valide, elle retourne un numéro de session. Si le participant n'est pas valide, elle retourne comme numéro de session la valeur « 0 ». Le numéro de session est un numéro séquentiel généré par un séquenceur interne à la classe.

2.2.11 La classe WfResource

Cette classe modélise les ressources utilisées par des activités. Une ressource peut être une donnée ou une application. Une ressource doit avoir un état: *ResourceAllocated* et *ResourceNotAllocated*.

TAB. 11 - *Attributs de la classe* **WfResource**

Nom	Type	Description
ResourceName	`string`	nom de la ressource
ResourceId	`integer`	identification de la ressource
ResourceCategory	`integer`	catégorie de la ressource (donnée ou application)
ResourceLocation	`integer`	localité de la ressource (par exemple l'adresse de l'ordinateur où elle se trouve)
ResourceAllocated	`boolean`	état de la ressource (utilisée ou non)

Les attributs de la classe **WfResource** sont décrits dans le tableau 11. Cette classe comporte cinq méthodes.

La méthode `SetResource()` permet l'insertion d'une ressource dans la base de données. Cette méthode n'a pas de paramètres puisque les données à enregistrer ont été fournies lors de sa création.

La méthode `GetResource(string` *name*`)` permet la récupération des données d'une ressource, celle ayant comme nom *name*. Les données récupérées sont les attributs de la ressource, c'est-à-dire *ResourceId*, *ResourceCategory*, *ResourceLocation* et *ResourceAllocated*.

La méthode `GetResource(integer` *id*`)` permet la récupération des données d'une ressource, celle ayant comme identifiant *id*. Les données récupérées sont les attributs de la ressource, c'est-à-dire *ResourceName*, *ResourceCategory*, *ResourceLocation* et *ResourceAllocated*.

La méthode `DisallocateResource()` change l'état d'une ressource en *ResourceNotAllocated*. En effet, cette méthode permet de libérer une ressource.

La méthode `AllocateResource()` change l'état d'une ressource en *ResourceAllocated*.

2.2.12 La classe WfTransition

Cette classe modélise les mouvements dans une procédure. En effet, une transition permet le passage d'une activité à une autre dans une procédure. Une transition peut être une réponse à un événement externe ou un appel de procédure par un participant. Chaque transition est composée d'une condition, d'un événement et de deux données. Elle est en relation avec une activité source et une activité destination.

TAB. 12 – *Attributs de la classe* **WfTransition**

Nom	Type	Description
TransitionId	**integer**	identifiant de la transition
TransitionName	**string**	nom de la transition
Branch	**branch**	type de branchement utilisé

Les attributs de la classe **WfTransition** sont décrits dans le tableau 12. Dans ce tableau, nous utilisons un type **branch** dont l'énumération est la suivante : *And_Split*, *And_Join*, *Or_Split*, *Or_Join* et *Iteration*. Cette classe comporte sept méthodes.

La méthode `SetTransition()` permet l'insertion d'une transition dans la base de données. Cette méthode n'a pas de paramètres puisque les données à enregistrer ont été fournies lors de sa création.

La méthode `GetTransition(string` *name*`)` permet la récupération des données de la transition ayant comme nom *name*. Les données récupérées sont les valeurs des attributs de la transition, c'est-à-dire *TransitionId* et *Branch*.

La méthode `GetTransition(integer` *id*`)` permet la récupération des données de la transition ayant comme identifiant *id*. Les données récupérées sont les valeurs des attributs de la transition, c'est-à-dire *TransitionName* et *Branch*.

La méthode `CountSucesseurs(integer` *activity*`)` retourne le nombre d'activités qui sont les successeurs de l'activité *activity*.

La méthode `GetFirstTransition(integer` *process*`, integer` *source*`)` identifie les

transitions ayant comme activité source celle passée en paramètre, mais qui sont propres à la procédure ayant comme identifiant *process*. En plus, elle permet d'obtenir une copie de la première transition identifiée ; l'objet auquel elle s'applique étant une transition. Si l'ensemble des transitions identifiées est vide, cette méthode retourne la valeur *faux*.

La méthode **GetNextTransition(integer** *process*, **integer** *source*) retourne la valeur *faux* si l'ensemble des transitions, obtenu antérieurement par un appel à la méthode **GetFirstTransition** avec les mêmes paramètres, a été entièrement parcouru. Autrement, elle permet d'obtenir une copie de la prochaine transition ayant comme activité source celle passée en paramètre, mais propre à la procédure ayant comme identifiant *process*.

La méthode **Evaluate(integer** *process_instance*) permet l'évaluation d'une condition, celle associée à une transition, celle associée à l'instance d'une procédure dont l'identifiant est *process_instance*. Elle récupère les deux données à comparer. Puis, elle les compare selon la valeur de l'attribut *Criterion* de la condition. La valeur retournée par la méthode est *true* si les données respectent la condition, sinon elle retourne la valeur *faux*.

2.2.13 La classe WfWorkItem

Cette classe modélise les bons de travail des participants. Elle permet l'enregistrement des bons de travail des participants et la récupération de la liste des bons de travail à effectuer par un participant.

TAB. 13 – *Attributs de la classe* **WfWorkItem**

Nom	Type	Description
WorkName	**string**	nom d'un bon de travail
WorkId	**integer**	identifiant d'un bon de travail

Les attributs de la classe **WfWorkItem** sont décrits dans le tableau 13. Cette classe

comporte quatre méthodes.

La méthode `GetWorkList(integer` *id*) permet la récupération de la liste des bons de travail du participant, celui ayant comme identifiant *id*.

La méthode `SetWorkItem()` permet l'insertion d'un bon de travail dans la base de données. Cette méthode n'a pas de paramètre puisque les données à enregistrer ont été fournies lors de la création.

La méthode `AddWorkItem(string` *name*, `integer` *id*, `integer` *participant*, `integer` *data_instance*, `integer` *activity_instance*, `integer` *resource*) met à jour les attributs de l'objet. Le nom du bon de travail est *name*, son identifiant est *id*. Ce bon de travail participe à l'instance d'activité *activity_instance*. Elle utilise la donnée identifiée par *data_instance* et la ressource identifié par *resource*. Ce bon de travail serai réalisé par un participant, celui ayant comme identifiant est *participant*

La méthode `RemoveWorkItem(integer` *id*) permet la suppression d'un bon de travail, celui ayant comme identifiant *id*.

2.3 Modèle dynamique

Le modèle dynamique s'intéresse au comportement des différentes classes qui forment le système de gestion de workflows distribué. Pour construire le modèle dynamique, nous procédons en cinq étapes. D'abord, il faut créer des scénarios pour les séquences d'interactions typiques. Puis, il faut identifier les événements entre les objets. Après, il faut préparer un diagramme de suivi des événements. Ensuite, il faut préparer le diagramme de flots des événements du système. Enfin, il faut développer un diagramme d'états pour chaque classe qui a un comportement dynamique important.

Dans notre étude, nous nous intéressons aux diagrammes d'états, puisque les autres étapes permettent l'esquisse des données servant de base à la construction des dits diagrammes. En d'autres termes, les étapes de modélisation du système par les scénarios et

l'esquisse des événements permettent de préparer les diagrammes d'états du système qui est l'objectif final du modèle dynamique.

Dans notre modèle dynamique, nous distinguons deux types de diagramme d'états. Le premier représente le diagramme d'états de tout le système. Le deuxième type de diagramme représente le comportement d'un objet à la réception d'un événement.

Dans ce qui suit nous décrivons le comportement général du système, puis nous décrivons le comportement des principales classes du système: **WfActivityInstance** et **WfProcessInstance**.

2.3.1 Modèle dynamique du système

La figure 6 illustre le comportement du système de gestion de workflows distribué.

À sa création un objet **WfParticipant** envoie l'événement `GetSessionId` à un objet **WfProcessMgr** pour créer une session. Si le système a pu créer une session, l'objet **WfParticipant** envoie l'événement `GetWorkList` à l'objet **WfWorkItem** pour récupérer la corbeille du participant.

Le participant peut choisir un bon de travail de sa corbeille et envoyer l'événement `Notify` à un objet **WfActivityInstance** pour démarrer l'exécution d'une activité. L'objet **WfActivityInstance** correspondant envoie l'événement `AllocateResource` à un objet **WfResource**. Si la ressource est allouée avec succès, il retourne l'événement `Allocate` à l'objet **WfActivityInstance**. À la fin de l'exécution de la ressource, l'objet **WfResource** envoie l'événement `Release` à l'objet **WfActivityInstance**. Sinon, l'objet **WfResource** envoie l'événement `Suspend` à l'objet **WfActivityInstance**. À la réception de l'événement de la fin de l'exécution du bon de travail, l'objet **WfActivityInstance** envoie l'événement `WorkItemTerminate` à l'objet **WfParticipant**. Ce dernier crée une nouvelle instance de donnée de l'activité et envoie l'événement `SetDataInstance` à un objet **WfDataInstance**. Enfin, l'objet **WfParticipant** envoie l'événement `RemoveWorkItem` à

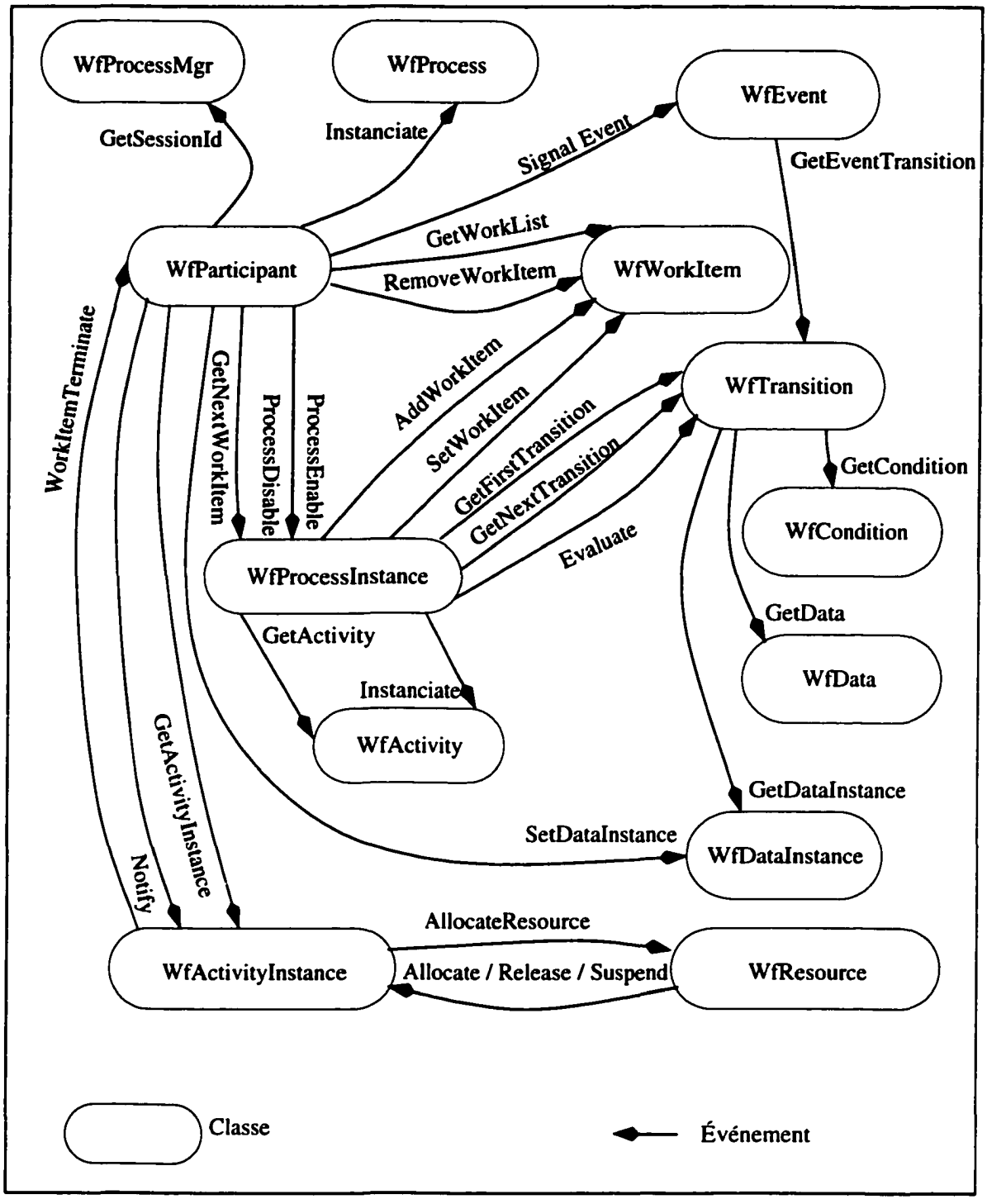

FIG. 6 - *Diagramme du modèle dynamique du système de gestion de workflows*

l'objet **WfWorkItem** pour détruire le bon de travail suivi de l'événement `GetNextWorkItem` pour demander à un objet **WfProcessInstance** la construction du prochain bon de travail.

À la réception de l'événement `GetNextWorkItem`, un objet **WfProcessInstance** envoie l'événement `GetFirstTransition` à un objet **WfTransition** avec comme paramètre l'activité courante. Cet objet cherche la transition dont l'activité source est l'activité fournie comme paramètre et envoie l'événement `GetCondition` à un objet **WfCondition** pour récupérer la condition. Puis, l'objet **WfProcessInstance** envoie l'événement `Evaluate` à l'objet **WfTransition** en question. Ce dernier, envoie l'événement `GetData` à un objet **WfData** pour récupérer la donnée utilisée par la partie gauche de la condition, suivi de l'événement `GetDataInstance` à un objet **WfDataInstance** pour récupérer l'instance de la donnée. Ce scénario de récupération d'une donnée est répété pour la partie droite de la condition. Enfin, il évalue la condition. Si la condition est vérifiée, l'objet **WfProcessInstance** envoie l'événement `GetActivity` suivi de l'événement `Instanciate` à l'objet **WfActivity** pour créer une nouvelle instance de l'activité suivante. Après, ce dernier construit le bon de travail suivant et envoie l'événement `AddWorkItem` à l'objet **WfWorkItem** pour ajouter le bon de travail suivi de l'événement `SetWorkItem` pour enregistrer le bon de travail. Si l'évaluation de la condition a retourné la valeur « faux », alors l'objet **WfTransition** cherche s'il existe une autre transition valable. Si l'objet **WfProcessInstance** a détecté que l'activité courante est la dernière, il conclut que c'est la fin de la procédure.

Le participant peut envoyer l'événement `SignalEvent` pour forcer un chemin dans la procédure. L'objet **WfEvent** envoie alors l'événement `GetEventTransition` pour récupérer la transition appropriée à l'événement en question. À la réception de cette événement, l'objet **WfTransition** récupère la condition et les données de la transition. Puis, il évalue la condition. Si la condition est vérifiée, il envoie l'événement `GetWorkItem` à la classe **WfProcessInstance** pour demander la création d'un bon de travail.

En plus des comportements que nous avons décrit, chaque classe dispose de trois méthodes qui permettent :

- l'insertion d'un nouvel objet dans la base de données,
- la récupération des données d'un objet dont la clé de recherche est l'identifiant de l'objet,
- la récupération des données d'un objet dont la clé de recherche est la désignation de l'objet.

2.3.2 Diagrammes d'états

Dans cette section, nous allons décrire les diagrammes d'états des classes qui ont un comportement jugé complexe.

Diagramme d'états de la classe WfActivityInstance

La figure 7 illustre le comportement d'un objet **WfActivityInstance**. Lors de la création d'un objet **WfActivityInstance**, son état prend la valeur *Started* grâce à l'événement `Start`. Puis, l'objet reçoit l'événement `Notify`, ce qui lui permet de passer à l'état *Activated*. En état *Activated* l'objet peut recevoir trois événements. S'il reçoit `Allocate`, il passe à l'état *Completed*. En état *Completed*, si l'objet reçoit `Release`, il passe à l'état *Terminated*. En état *Activated*, l'objet peut aussi recevoir l'événement `Suspend` et il passe à l'état *Suspended*. À cet état l'objet reste en attente de l'événement `Notify` pour revenir à *Activated*. Enfin, en état *Activated*, l'objet peut reçevoir l'événement `Abort` et il passe donc à *Terminated*.

Diagramme d'états de la classe WfProcessInstance

La figure 8 illustre le comportement de la classe **WfProcessInstance**. Après la création d'un objet de cette classe, il reçoit l'événement `Start` pour prendre son état

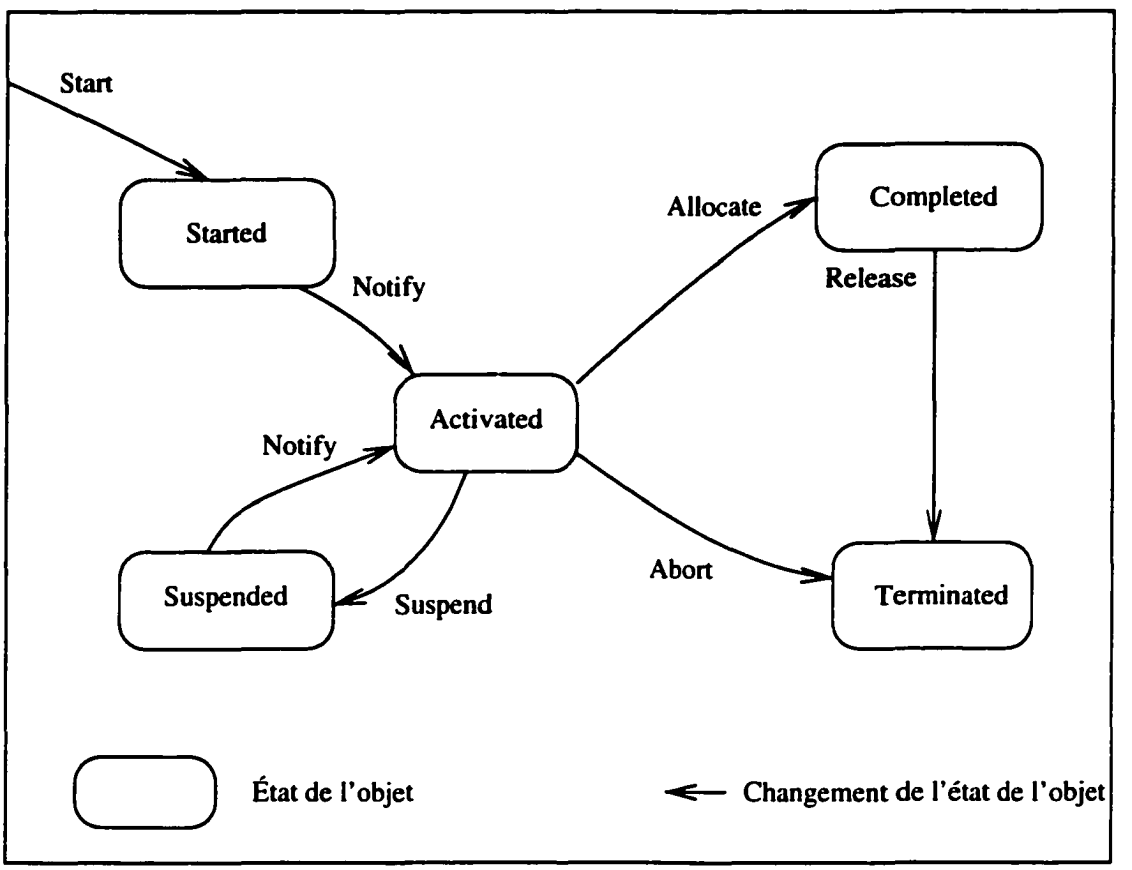

FIG. 7 – *Diagramme d'états de la classe* **WfActivityInstance**

initial *Enabled.* Dans cet état l'objet peut recevoir l'événement **`Terminate`** et change en état *Terminated.* Il peut aussi recevoir l'événement **`Disable`** et changer en état *Disabled.* Lorsque l'objet est en état *Disabled* et reçoit l'événement **`Enable`**, il revient à l'état *Enabled.* En état *Disabled*, il peut aussi reçevoir l'événement **`Terminate`** et changer en état *Terminated.*

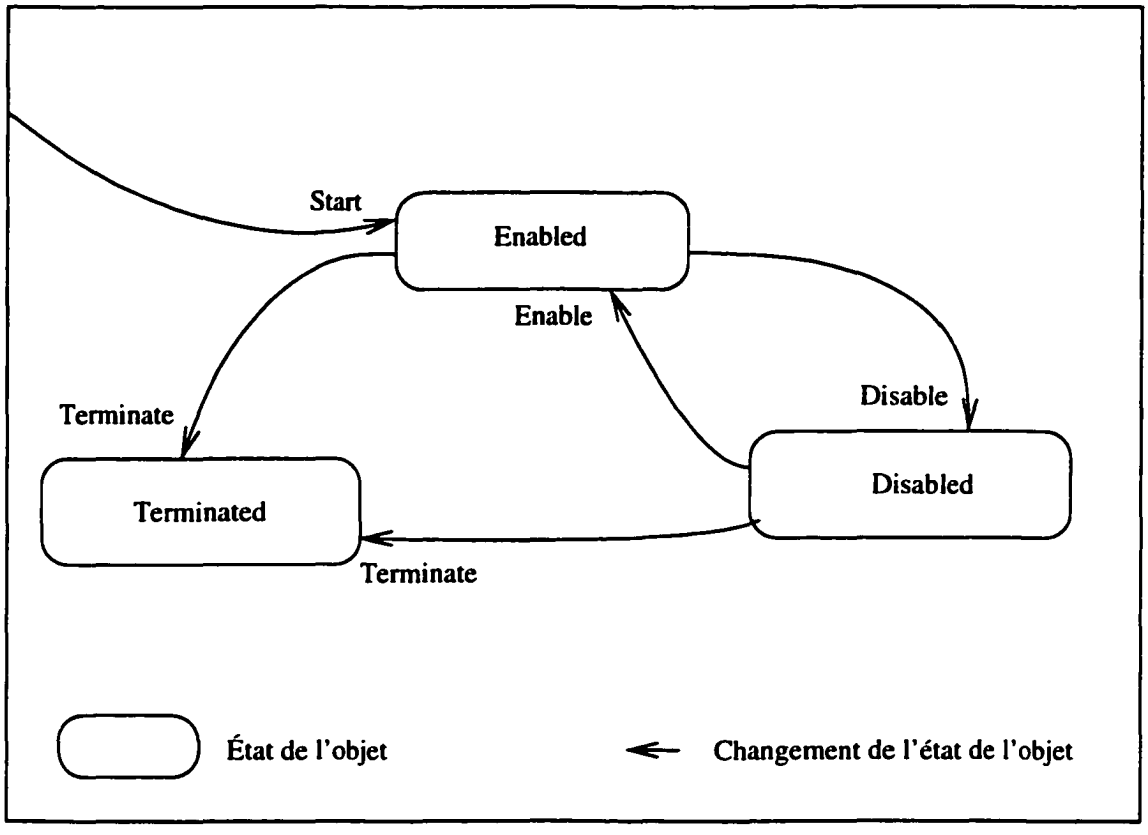

FIG. 8 – *Diagramme d'états de la classe* **WfProcessInstance**

Chapitre 3

Conception

Après l'analyse de notre système de gestion de workflows distribué, nous devons décider de la manière avec laquelle nous allons résoudre les problèmes détectés dans la section problématique de l'introduction. Dans cette phase, nous allons répondre à la question « Comment faire? ».

Dans cette phase, nous organisons le système de gestion de workflows distribué en sous-systèmes. Puis, nous allouons les composants logiciels aux sous-systèmes. Enfin, nous définissons l'architecture du système.

Ce chapitre est composé de quatre sections. Dans la première section, nous présentons brièvement les systèmes distribués et les techniques d'implémentation de ces systèmes. Dans la deuxième section, nous décomposons le système de gestion de workflows distribué en des groupes de composants à savoir les composants logiciels, les données du système et les applications utilisées par le système. Nous nous intéressons par la suite à la description des composants logiciels du système. Dans la troisième section, nous présentons des alternatives d'implémentation des composants du système. Dans la dernière section, nous décrivons l'architecture de référence de l'organisme WfMC. Puis nous présentons l'architecture de notre système de gestion de workflows distribué.

3.1 Systèmes distribués

Un système informatique distribué est un système comprenant des ressources physiques et logiques géographiquement dispersées et reliées par un système de communication qui leur permet d'échanger de l'information en vue de coopérer à une tâche commune, et possédant un haut degré de cohésion et de transparence.

Dans ce qui suit nous présentons les caractéristiques d'un système distribué et les techniques d'implémentation des systèmes distribués.

3.1.1 Caractéristiques d'un système distribué

Les principales caractéristiques d'un système distribué sont [4] :

- La communication – Dans un système distribué, les composants du système communiquent et coopèrent afin de réaliser une tâche. Cette communication doit respecter des règles via l'utilisation de protocoles.
- L'hétérogénéité – Les systèmes distribués peuvent utiliser des environnements hétérogènes comportant plusieurs protocoles (TCP/IP, IPX/SPX et X25).
- L'intégration – Afin de préserver les investissements et d'offrir de nouveaux services, tout en ayant un système cohérent et opérationnel, le concepteur doit être capable d'intégrer les anciens composants d'un système informatique dans un nouveau système.

3.1.2 Techniques d'implémentation des systèmes distribués

Un système distribué a besoin de mécanismes qui permettent la recherche des ressources partagées sur le réseau, la réalisation de traitements transactionnels et la persistance des données partagées.

Ces besoins peuvent être satisfaits en utilisant des techniques de communication [12].

Parmi ces techniques, nous pouvons citer :

- La communication par messages – C'est la communication par échange de messages en utilisant des protocoles de communication tels que le protocole TCP.
- L'appel de procédure à distance – Cette technique permet l'appel et l'exécution de procédures situées dans des processus s'exécutant sur d'autres machines.
- Les objets distribués – L'utilisation d'objets distribués repose sur le mécanisme d'invocation des méthodes sur des objets distants.

L'appel d'une méthode sur un objet distant est un mécanisme qui permet l'exécution éloignée du code d'une méthode sur l'objet. Ce mécanisme est une extension de l'appel de procédure à distance (RPC) dans le contexte des objets [4]. Avec ce mécanisme, les objets distribués communiquent d'une manière transparente par l'intermédiaire d'objets souches. En effet, pour chaque objet, il doit exister deux souches. La première souche est la souche serveur ayant comme fonctions principales de déballer les arguments de la méthode et d'emballer les résultats. La deuxième souche est la souche cliente. Elle a comme fonctions principales d'emballer les arguments de la méthode et de déballer les résultats.

La figure 9 schématise le processus d'invocation d'une méthode distante. Selon ce scénario d'appel d'une méthode distante, le client invoque la souche cliente (1). La souche emballe les arguments (2) et envoie une requête à la souche serveur (3). La souche serveur déballe les arguments (4) et appel l'objet réel (5). À la fin de l'exécution de la méthode, l'objet retourne les résultats à la souche serveur (6) qui emballe ces résultats (7) et envoie le message à la souche cliente (8). Cette dernière déballe les résultats (9) et retourne les résultats à l'objet client (10).

En effet le mécanisme d'objets distribués offre une représentation physique des données et des références des objets sous forme de suites d'octets. Il permet aussi la gestion des connexions et l'ordonnancement des messages.

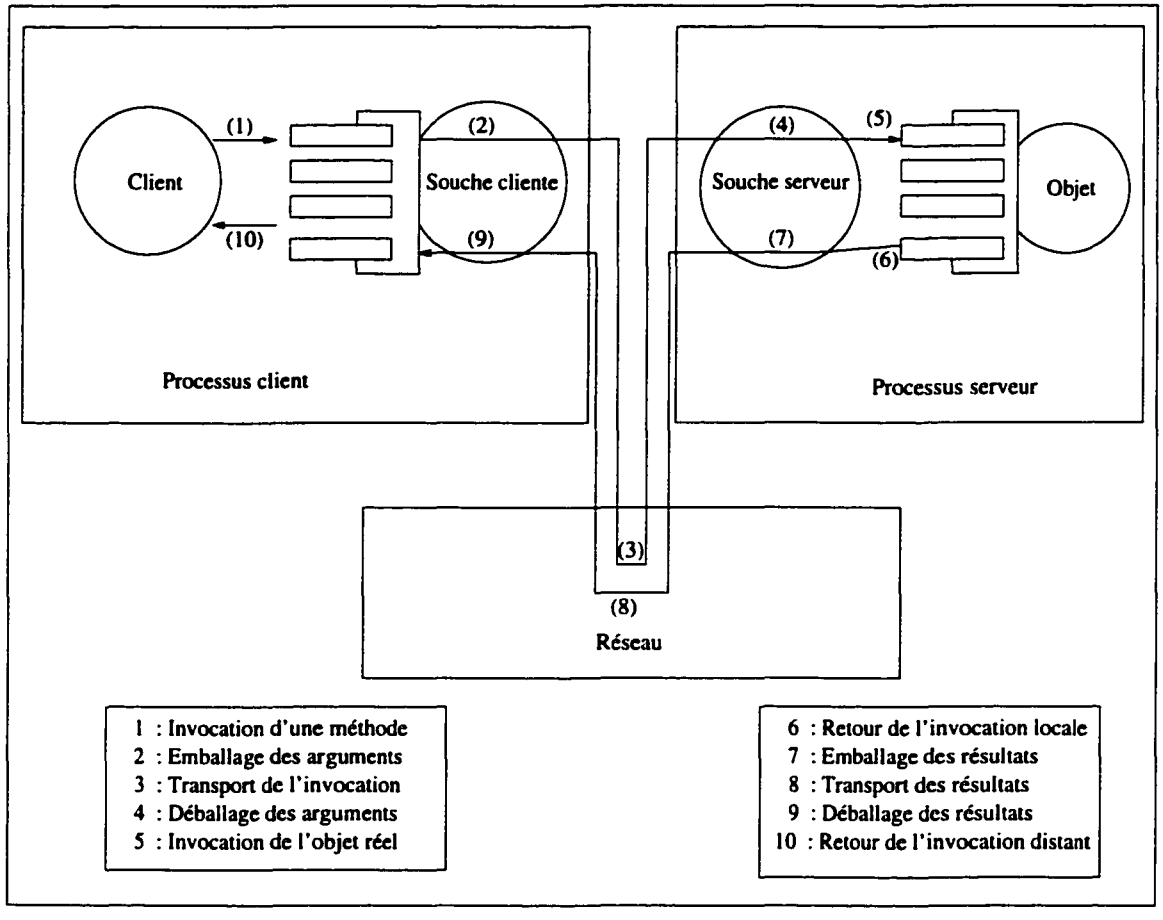

FIG. 9 – *L'appel d'une méthode distante tiré de [4]*

3.2 Description des composants d'un système de gestion de workflows distribué

La figure 10 illustre un modèle générique d'un système de gestion de workflows. Ce système est composé de trois catégories de composants logiciels.

La première catégorie regroupe les composants du système de gestion de workflows qui fournissent diverses fonctionnalités à savoir le gestionnaire de workflows, les bons de travail, le pointeur des bons de travail et l'interface utilisateur. La deuxième catégorie regroupe les données utilisées par le système de gestion de workflows distribué. Enfin, la troisième catégorie regroupe les applications et les systèmes de gestion de données qui peuvent être utilisés dans le système de gestion de workflows distribué.

En plus de ces composants, le modèle contient un outil de définition de workflows. La description de ce composant n'est pas pertinente dans cette étude.

3.2.1 Gestionnaire de workflows

Le gestionnaire de workflows a comme fonctions l'interprétation de la description d'un workflow et le contrôle de son exécution. Ces fonctions permettent l'instanciation de procédures, l'instanciation d'activités et le séquencement d'activités. Ce gestionnaire permet aussi l'ajout des bons de travail à la liste des travaux d'un participant et l'appel des applications utilitaires.

Le gestionnaire de workflows maintient les données de contrôle d'un workflow. Il utilise les définitions d'un workflow et les données de contrôle de ce dernier pour naviguer d'une activité à une autre. Il permet aussi la gestion des instances d'activité et des instances de procédure.

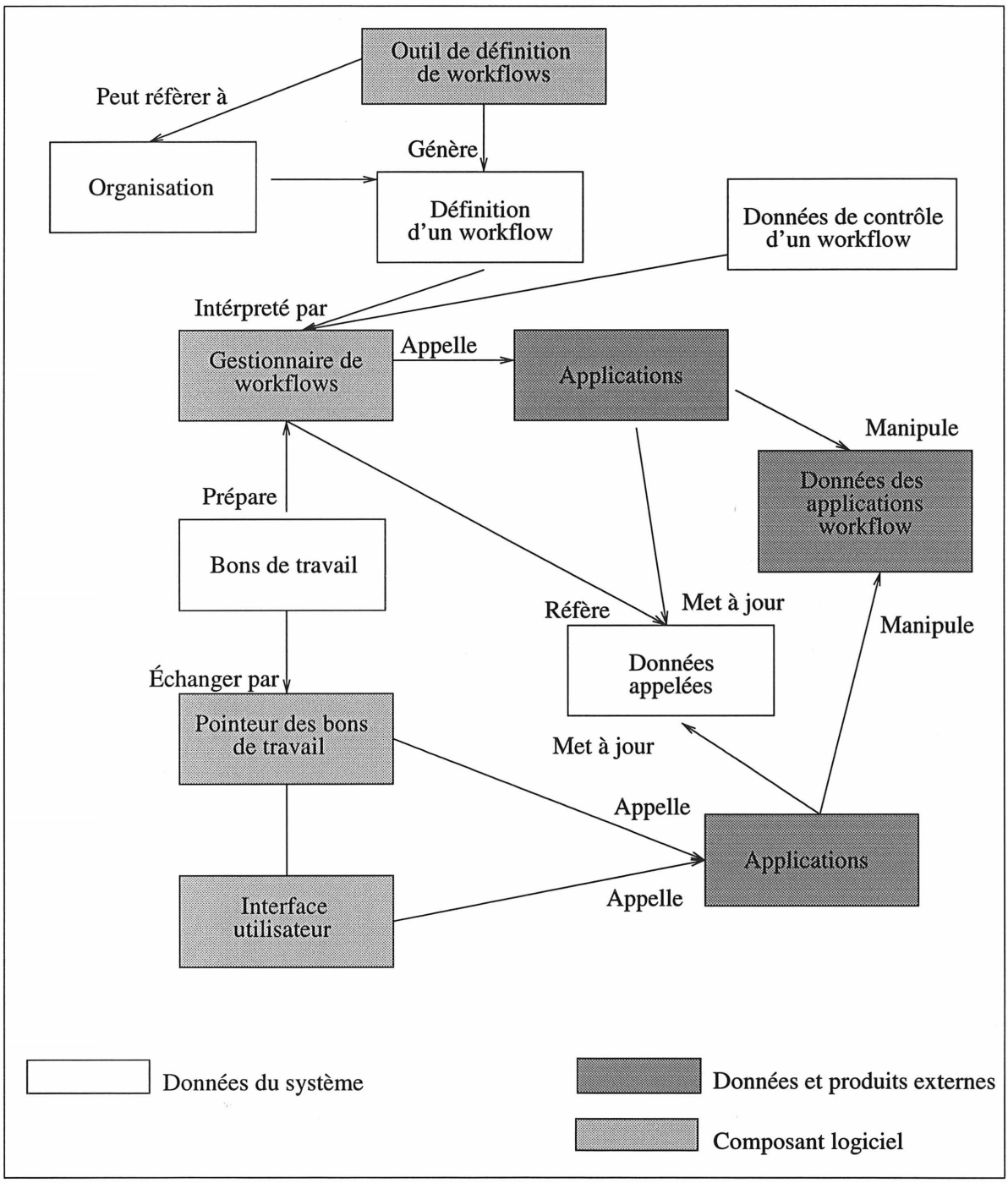

FIG. 10 – *Structure générique d'un système de gestion de workflows*

3.2.2 Données de contrôle d'un workflow et données applicatives

Dans un workflow les transitions permettent le passage d'une activité à une autre. Ces transitions utilisent deux types de données: les données de contrôle d'un workflow et les données de contrôle de l'exécution d'un workflow.

Les données de contrôle d'un workflow font partie d'un workflow. Par exemple, dans le workflow *circuit de vente*, présenté en annexe C, les données *Acceptation* et *Stock* sont des données de contrôle. Ces données permettent la prise de décision et le choix de l'activité suivante.

Les données de contrôle internes du système (appelées données de contrôle de l'exécution d'un workflow) indiquent l'état d'un workflow. Elles décrivent l'état des instances d'activité et des instances de procédure (e.g., *Completed*, *Started*)

3.2.3 Bons de travail

La liste des bons de travail est un réservoir de données qui contient les bons de travail à traiter par un participant. Lorsqu'un participant interagit avec un workflow, le gestionnaire de workflows ajoute ou retire des bons de travail à cette liste.

La liste des bons de travail peut être visible ou invisible aux participants, et ceci selon le domaine d'application. Dans le cas d'une application de gestion, il est préférable que le participant visualise la liste des travaux et traite le travail qu'il choisit. Dans le domaine médical, il est préférable que cette liste soit invisible aux participants [9]. Dans ce domaine, le médecin n'a pas la possibilité de choisir le patient à traiter ou l'analyse à effectuer, mais il peut insister sur une analyse particulière pour mener à bien son diagnostic.

Dans ce travail, nous utilisons le terme corbeille comme synonyme de liste des bons de travail.

Dans notre travail, nous choisissons la liste des bons de travail visible, qui est le cas le plus utilisé.

3.2.4 Pointeur des bons de travail

Ce composant gère les interactions entre un participant et des bons de travail. En effet, chaque participant a besoin de travailler sur sa corbeille. Il récupère une copie de cette liste. Puis, il retire et ajoute des bons de travail à cette liste.

Le pointeur des bons de travail est l'interface client d'un *gestionnaire de workflows*. Il exécute les requêtes demandées par l'utilisateur tels que l'exécution d'un bon de travail, la création ou la gestion d'une instance de procédure et l'insertion d'un événement workflows à une instance de procédure.

En plus de ses fonctions, le pointeur des bons de travail maintient les données workflow gérer par le client workflows

3.3 Alternatives d'implémentation

Dans la section précédente nous avons identifié un ensemble de composants du système de gestion de workflows distribué. Il existe plusieurs architectures pour l'implémentation de ces composants. Les critères de choix d'une architecture sont les fonctionnalités des composants et la plate-forme réseau choisie. Dans notre travail nous construisons une architecture d'un système de gestion de workflows en se basant sur :

- les fonctionalités d'un *gestionnaire de workfows* et d'un *client workflows*,
- le bus de communication.

Le composant *gestionnaire de workflows* est un mécanisme de manipulation des workflows. Ce composant peut être centralisé ou distribué.

Le composant *pointeur des bons de travail* est un mécanisme de distribution des *bons de travail*. Il peut être centralisé ou distribué.

3.3.1 Gestionnaire de workflows

Le composant *gestionnaire de workflows* regroupe un ou plusieurs *serveurs workflows.* Chaque *serveur workflows* est responsable de l'exécution de tout ou d'une partie d'une instance de procédure.

Nous distinguons deux architectures d'implémentation du *gestionnaire de workflows* à savoir l'architecture distribuée et l'architecture centralisée. Dans cette première architecture les données de contrôle des instances de procédure sont accessibles aux *serveurs workflows* qui traitent les instances de procédure. Cette accessibilité peut est réalisée en mettant en place un serveur maître qui gère les données de contrôle ou les données partagées (voir la figure 11 a). Dans la deuxième architecture, le *gestionnaire de workflows* comporte un seul *serveur workflows.* Le serveur exécute toutes les instances de procédure (voir la figure 11 b).

Il existe deux possibilités d'implémentation de la liste des *bons de travail* (voir la figure 11). Dans la première implémentation, nous associons à chaque participant une liste qui accompagne le gestionnaire (voir la figure 11 a), alors que dans la deuxième, nous associons une seule liste pour tous les participants (voir la figure 11 b).

3.3.2 Client workflows

C'est l'interface de communication entre un participant et le *gestionnaire de workflows.* Cette interface permet à un participant de récupérer et de visualiser sa liste des bons de travail, de lancer l'exécution d'une instance d'activité ainsi que de créer et d'arrêter une instance de procédure.

Il y a quatre modèles d'implémentation de la liste des bons de travail : le modèle hôte, le modèle de fichiers partagés, le modèle de messages électroniques et le modèle d'appels de procédure (voir la figure 12 où la terminologie anglaise originale est utilisée).

Le modèle hôte est un modèle centralisé. Dans ce modèle le *pointeur de bons de travail*

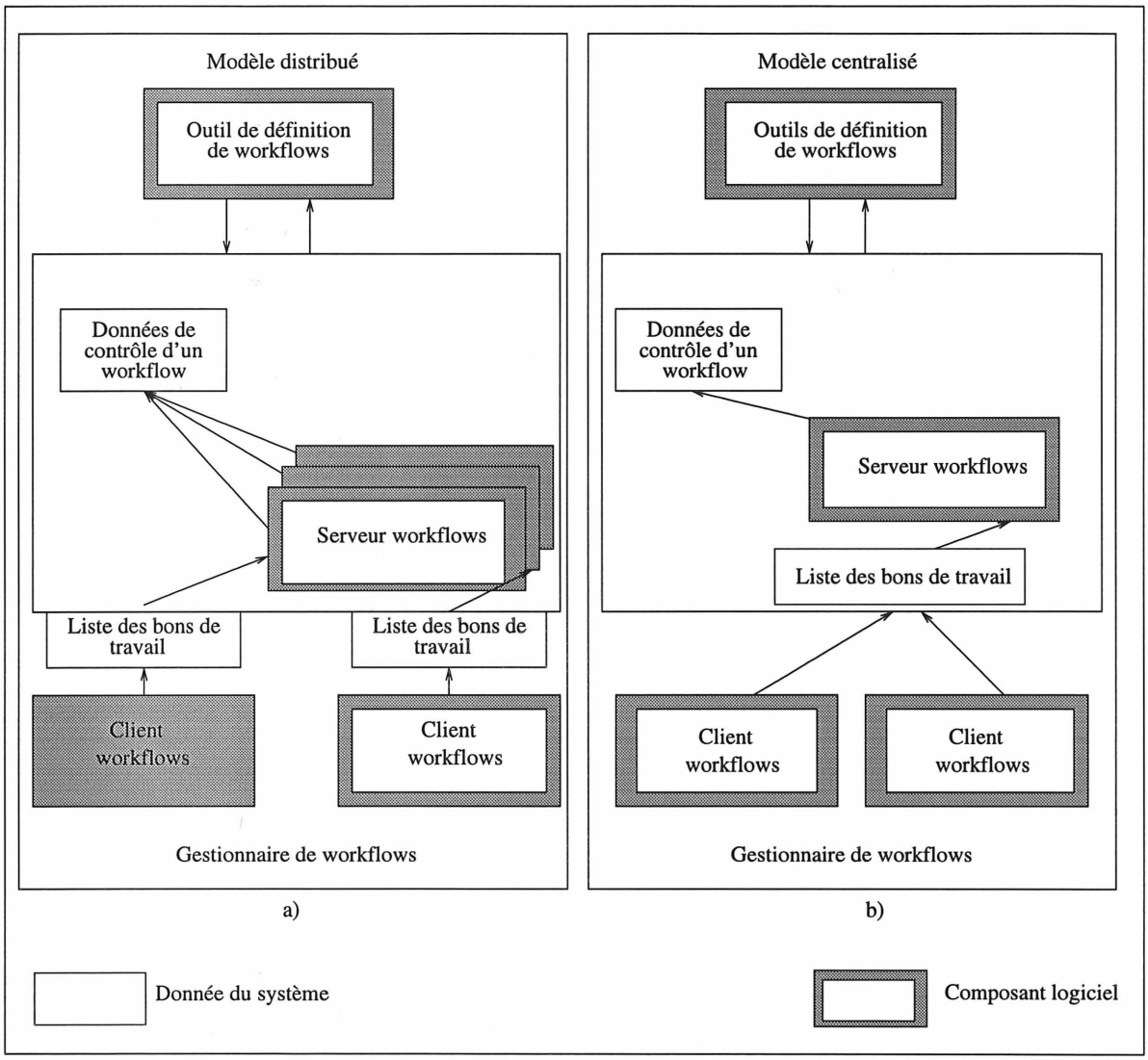

FIG. 11 – *Implémentation d'un gestionnaire de workflows tirée de [12]*

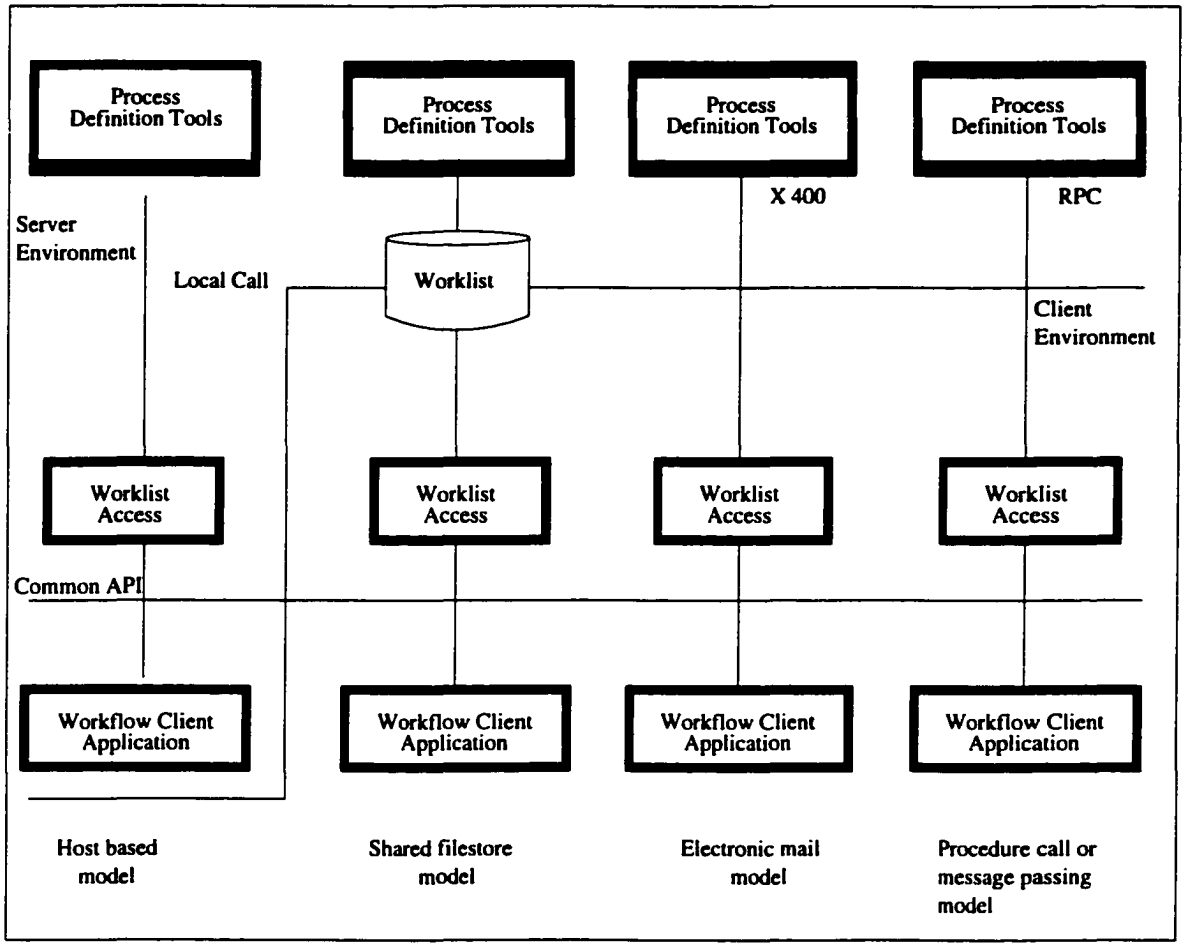

FIG. 12 – *Implémentation des bons de travail tirée de [12]*

existe sur la même machine où se trouve les *bons de travail.* La communication entre les deux composants se fait localement. Ce modèle est utilisé par les systèmes interactifs basés sur des terminaux.

Le modèle de fichiers partagés est un modèle distribué. Le *pointeur de bons de travail* est considéré comme un composant du client. Le *gestionnaire de workflows* et le *client workflows* communiquent via des données partagées. Ces données représentent les *bons de travail.* Les données partagées sont gérées par un gestionnaire de données qui est accessible directement par le *pointeur de bons de travail.*

Le modèle de messages électroniques est un modèle distribué. Le *gestionnaire de workflows* et le *client workflows* communiquent via des messages électroniques. Ces messages sont transmis par des protocoles de communication [12]. Ils peuvent être formater selon des normes [30].

Le modèle d'appels de procédure est un modèle distribué. Dans ce modèle la communication entre le client et le serveur se fait via des appels de procédure.

3.4 Alternatives d'architectures des systèmes de gestion de workflows

Dans cette section nous présentons l'architecture du modèle du WfMC et notre modèle de gestion de workflows distribué.

3.4.1 Architecture du modèle de la WfMC

Le WfMC a présenté un modèle de référence d'un système de gestion de workflows [6, 29, 12]. Cet organisme de normalisation a identifié dans l'architecture de son système les composants et les interfaces du système (voir la figure 13). Les interfaces qui composent le *gestionnaire de workflows* sont appelées WAPI. Elles sont utilisées pour l'accès et la

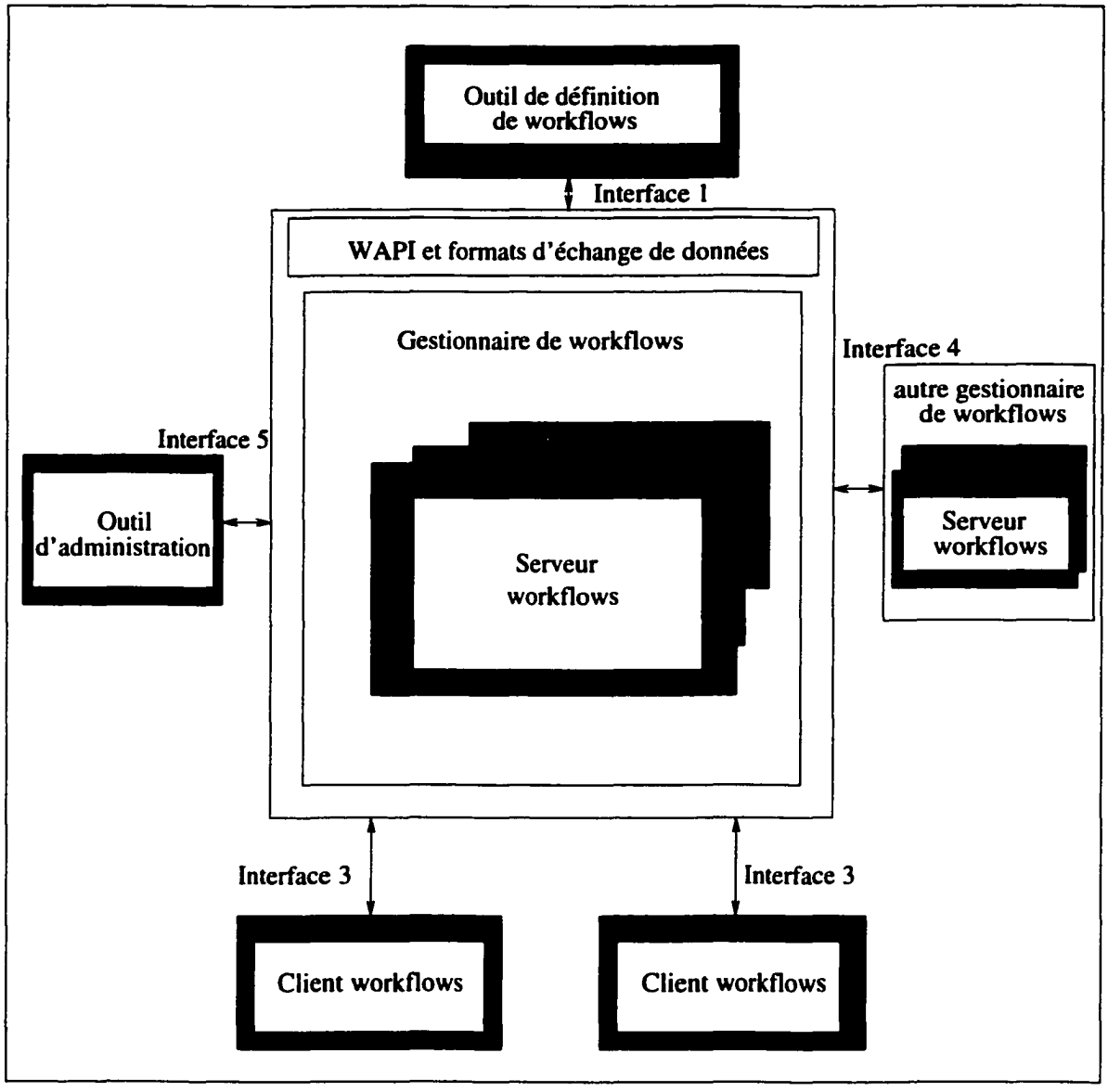

FIG. 13 – *Modèle de référence de la WfMC tiré de [12, 6]*

régulation entre le *gestionnaire de workflows* et les autres composants d'un workflow [12].

Les composants de ce système sont :

- Serveur workflows – C'est un service logiciel qui regroupe un ou plusieurs moteurs workflow qui permettent la création, la gestion et l'exécution des instances de workflow. Les applications peuvent communiquer avec ce service en utilisant les WAPIs.
- Moteur workflow – C'est un service qui offre un environnement d'exécution de tout ou une partie d'une procédure.
- Application appelée – C'est une application appelée par le système de gestion de workflows distribué. Cette application automatise une activité.
- Application cliente – C'est une application qui interagit avec un moteur workflow. Elle peut demander à un serveur workflows la recherche et la manipulation des données qui définissent un workflow, l'instanciation d'une procédure de travail ainsi que la gestion d'une corbeille.
- Outil de définition de workflows – Cet outil permet la modélisation des procédures de travail et l'enregistrement de la description des procédures.
- Outil d'administration – Cet outil permet la supervision du fonctionnement des procédures de travail. En effet, avec cet outil, l'administrateur peut changer le routage d'une procédure en cours d'exécution ou changer de sa définition.

3.4.2 Architecture de notre système de gestion de workflows distribué

Le système de gestion de workflows distribué est formé de cinq composants principaux qui sont le *bus de communication*, le *client workflows*, le *serveur workflows*, l'*outil de définition d'un workflow* et l'*outil d'administration.*

Ces composants sont formés d'un ensemble d'objets distribués. Ces composants communiquent selon une architecture client/serveur.

Comme il est illustré à la figure 14, notre système de gestion de workflows distribué est composé principalement d'un seul gestionnaire workflows et de plusieurs clients workflows qui peuvent être répartis géographiquement. Le premier aspect de distribution de notre système est que les composants de ce dernier **communiquent** entre eux en utilisant le bus RMI.

Le deuxième aspect de distribution est que les composants peuvent s'exécuter dans des environnements **hétérogènes**. Par exemple, nous pouvons exécuter le serveur workflows dans un environnement Windows NT (éventuellement avec une machine PC) et exécuter des clients dans un environnement Sun (système d'exploitation Solaris).

Le troisième aspect de distribution de notre système de gestion de workflows distribué est la distribution des ressources. En effet, les ressources utilisées par un workflow peuvent être répartis géographiquement mais accessibles au serveur workflows.

Comme technique d'implémentation de notre système de gestion de workflows distribué, nous avons utilisé la technique des objets distribués.

La figure 14 schématise ces composants ainsi que la communication entre ces composants. Dans ce qui suit nous nous intéressons aux trois premiers composants.

Bus de communication

Le *bus de communication* est le mécanisme qui permet la communication entre les composants d'un système de gestion de workflows distribué sur plusieurs ordinateurs. Ainsi, ce bus permet le transport des requêtes entre les différents composants du système de gestion de workflows distribué. Il permet aussi la communication entre les composants du système et la coopération entre ces composants.

Comme technique de communication entre les objets qui forment les composants, nous avons choisi la technique des objets distribués. Cette technique permet au système de gestion de workflows distribué de répondre aux critères des systèmes distribués en plus de faciliter la distribution des objets. Il existe plusieurs normes de bus de communication

qui répondent à notre choix. Parmi ces bus, nous pouvons citer le bus CORBA [23, 4] et le bus RMI [25].

Dans notre implémentation, nous utilisons un seul langage de programmation qui est Java [7]. Afin de bénéficier des facilités qu'offre le langage Java, nous avons choisi le bus de communication RMI de Java pour gérer les objets distants.

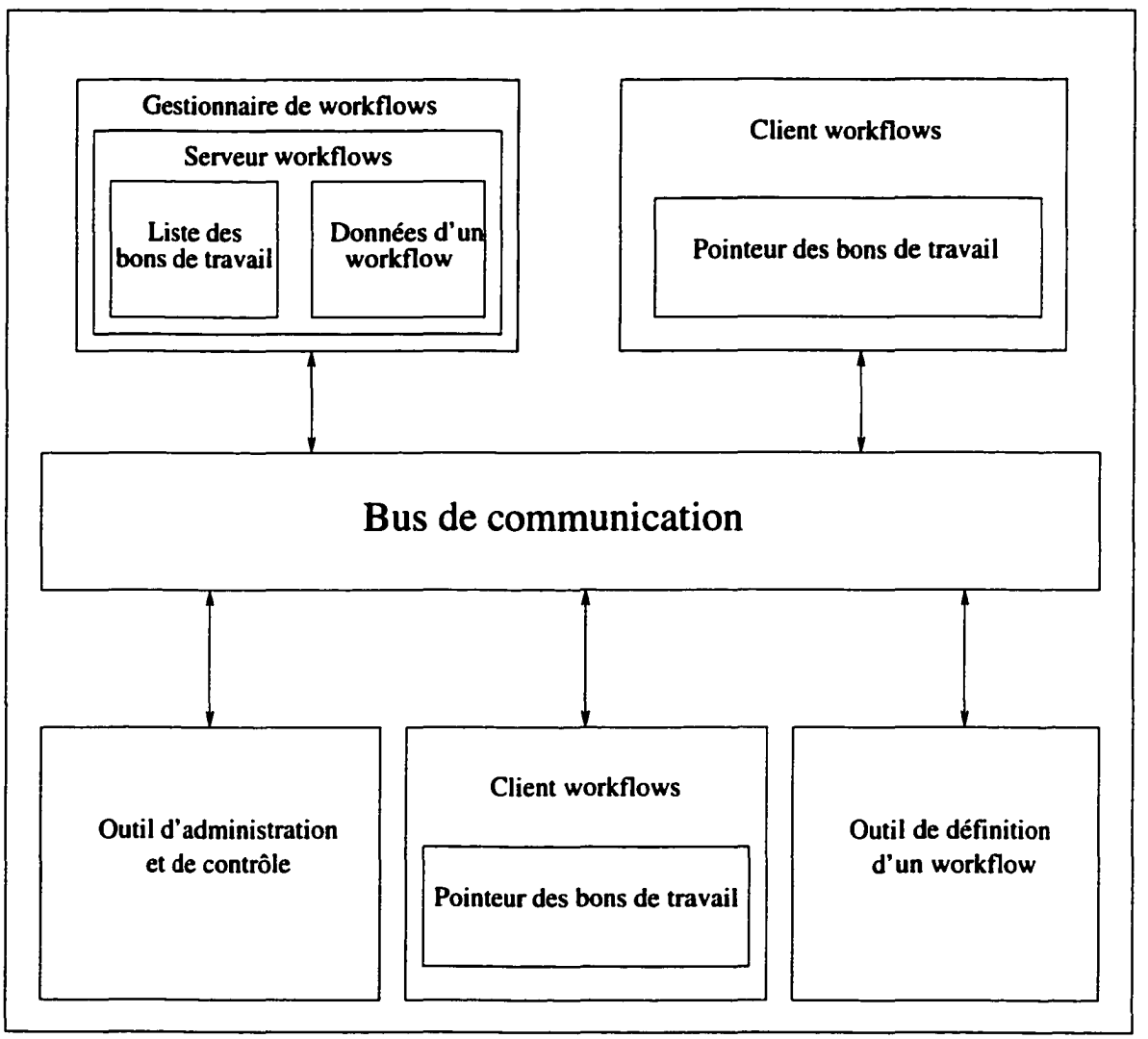

FIG. 14 – *Architecture du système de gestion de workflows distribué*

Client workflows

Le *client workflows* est l'interface qui permet à un participant de gérer ses *bons de travail.* En effet, avec ce composant l'utilisateur donne des ordres au *serveur workflows* et reçoit les réponses du système. Ces ordres se résument en l'ouverture d'une session sur le *gestionnaire de workflows*, l'allocation de ressources, l'instanciation de procédures et l'exécution d'instances d'activité. Ce composant communique avec le *pointeur des bons de travail* pour gérer la corbeille du participant.

Notre système de gestion de workflows distribué utilise plusieurs *clients workflows* qui peuvent être répartis sur plusieurs ordinateurs.

Gestionnaire de workflows

Dans la section précédente nous avons présenté deux alternatives d'implémentation d'un *gestionnaire de workflows.* Dans notre architecture nous avons choisi le bus RMI comme bus de communication entre les objets distribués. Or, le bus RMI dispose d'un seul serveur d'objets nommé « Registry » [25]. Ce dernier gère les références des objets distants. Cette contrainte nous a obligé à utilisé le modèle centralisé. Avec ce modèle, le *gestionnaire de workflows* est composé d'un seul *serveur workflows.* Dans le modèle distribué, nous devons étudier les interactions entre les *serveurs workflows* [12].

Le *gestionnaire de workflows* permet l'exécution d'instances de procédure selon des ordres d'un participant reçus à partir d'un *client workflows.* Ce gestionnaire gère aussi les données des bons de travail et les données internes qui permettent la gestion des instances de procédure. Il intègre un système de gestion de réservoirs de données. Dans notre système, nous avons choisi, comme système de gestion des données, un système de gestion de bases de données. Ce choix nous permet de déléguer cette tâche à un outil externe au système.

3.5 Conclusion

Dans ce chapitre nous avons présenté l'architecture de référence de la WfMC et l'architecture de notre système de gestion de workflows distribué.

Les principaux composants de ces deux architectures sont un *gestionnaire de workflows*, des *clients workflows*, un *outil de définition d'un workflow* et un *outil d'administration et de contrôle.*

Dans son architecture, le WfMC propose des interfaces de programmation des workflows et il propose des architectures de communication entre les composants du système de gestion de workflows, en utilisant principalement l'appel de procédures distantes (RPC) [12]. Dans notre travail nous avons présenté un système de gestion de workflows distribué qui utilise la technique d'objets distribués comme moyen de communication entre les composants du système et une base de données pour la gestion des définitions des workflows.

Chapitre 4

Implémentation

Dans les deux chapitres précédents nous avons présenté l'analyse et la conception de notre système de gestion des workflows distribué. Ce système est développé avec le langage de programmation Java. La base de données a été implémentée à l'aide du SGBD d'Oracle sous Windows NT 4.0.

Dans la première section, nous présentons l'architecture globale du système de gestion de workflows distribué, puis les architectures d'un client workflows et d'un serveur workflows. Dans la deuxième section, nous décrivons les moyens logiciels utilisés. Dans la troisième section, nous décrivons la base des données workflows. Dans la dernière section, nous présentons la structure des différents modules qui forment le système de gestion de workflows distribué.

4.1 Architecture du système de gestion de workflows distribué

Le système de gestion de workflows distribué est un système à architecture client/serveur. Le client et le serveur communiquent via des appels de méthodes distantes. Le

système de gestion de workflows distribué est formé de trois composants logiciels. La figure 15 illustre ces composants. Le premier composant est le *client workflows* qui est, d'une part, une interface utilisateur et, d'autre part, une interface de communication avec le serveur.

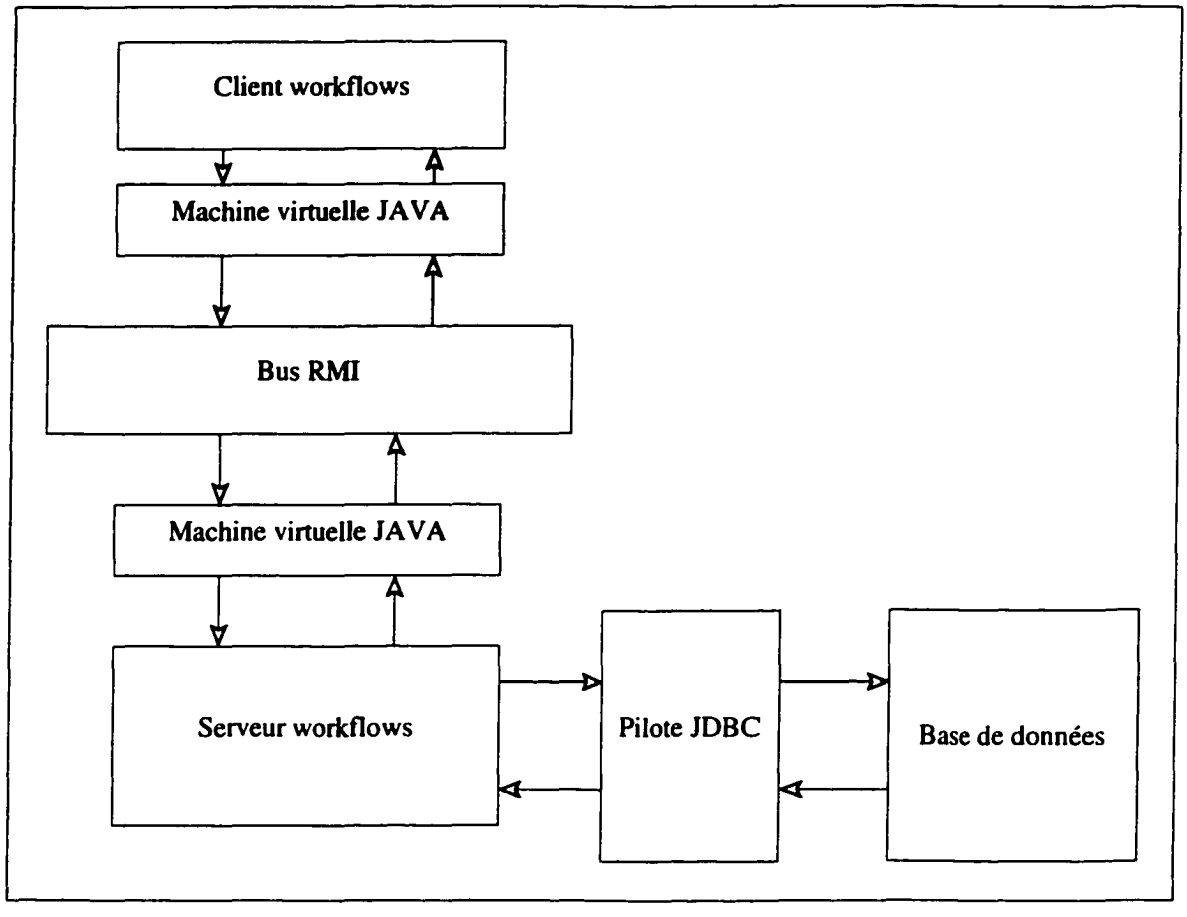

FIG. 15 – *Architecture globale du système de gestion de workflows distribué*

Le deuxième composant est le *serveur workflows* qui regroupe un ensemble d'objets. Ces objets permettent la manipulation de la base des données ainsi que la définition et la gestion de workflows. Enfin, le troisième composant est la base de données où le système de gestion des workflows distribué enregistre les données workflow.

4.2 Moyens logiciels utilisés

Afin que notre système de gestion de workflows distribué soit à architecture d'objets distribués et indépendant de la plate-forme, nous l'avons écrit en langage Java [7] (version 1.2). Parmi les avantages de Java est que son compilateur produit un code indépendant de la machine sur laquelle il a été compilé. En plus, l'interpréteur Java exécute du code Java selon les spécificités de la machine sur laquelle il est installé.

Les appels de méthodes distantes sont réalisées via le bus RMI [25] de Java et les requêtes à la base de données son traitées avec le JDBC de Java en utilisant le pilote *thin* [26].

Les données workflow sont manipulées avec le SGBD d'Oracle 8.0. Ce système de gestion de bases de données permet la gestion d'un nombre important de données dans un environnement multi-utilisateurs [18]. Ce SGBD est à architecture client/serveur dans laquelle le serveur est responsable de la gestion des données et des connexions demandées par des applications clientes.

4.3 Structure de la base de données

Pour construire la base de données nous avons projeté les classes définies dans un modèle objet en des entités d'un modèle relationnel. Pour faire les liens entre les objets, nous avons transformé les identificateurs des objets en index de tables. Puis, nous avons ajouté à certaines tables les identificateurs des super-classes.

Le tableau 14 illustre les tables qui forment la base de données. La description detaillée de ces tables est donnée en annexe B. Cette annexe contient les attributs des tables ainsi que les types de données et les tailles des champs. La description des champs sont les mêmes que les attributs des objets introduits dans le chapitre portant sur l'analyse.

TAB. 14 – *Liste des tables de la base de données*

Nom	Description
Activity	descriptions des activités qui forment les procédures de travail
ActivityInstance	exécutions d'activités
Condition	conditions à évaluer pour exécuter les transitions workflow
Data	descriptions des données utilisées par les workflows
DataInstance	données utilisées dans l'exécution de workflows
Event	événements qui permettent le déclenchement des transactions workflow
Resource	ressources manipulées par les workflows
Participant	participants au système de gestion de workflows distribué
Process	descriptions des procédures de travail
ProcessInstance	exécutions de procédures de travail
ProcessMgr	descriptions des gestionnaires de workflows
Transition	descriptions des transitions des procédures
WorkList	listes des bons de travail à exécuter par les utilisateurs

4.4 Structure des modules

Notre système de gestion de workflows distribué est composé de deux composants: le *client workflows* et le *serveur workflows*. Le client représente l'interface utilisateur. Le serveur regroupe les objets responsables de la gestion des workflows.

4.4.1 Module client

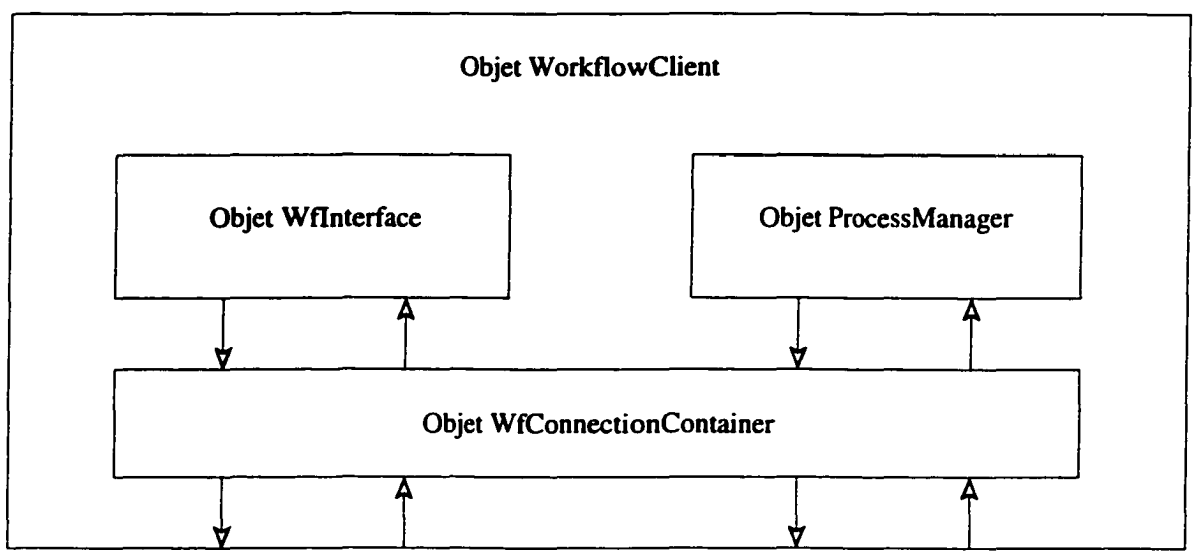

FIG. 16 – *Architecture du client workflows*

Le module client est représenté par l'objet **WorkflowClient**. La figure 16 illustre cet objet qui est composé de trois objets: **WfInterface**, **ProcessManager** et **WfConnectionContainer**.

Dans la classe **WfConnectionContainer** nous implémentons des méthodes où nous faisons appel aux objets distants.

L'objet **ProcessManager** concerne la gestion des procédures de travail et les instances de procédures. Avec l'interface qu'offre cet objet, l'utilisateur peut créer de nouvelles instances de procédure et gérer les instances de procédure existantes en changeant

leurs états courants.

L'objet **WfInterface** est le bureau workflow de l'utilisateur. Cet objet offre à l'utilisateur une interface graphique qui lui permet de se connecter au *serveur workflows* et de récupérer sa corbeille. La liste des bons de travail est affichée dans un tableau ce qui permet à l'utilisateur de choisir le bon de travail à exécuter.

4.4.2 Module serveur

Le module serveur est composé d'un ensemble d'objets qui gèrent les workflows en plus de l'objet principal appelé **WorkflowServer** (voir la figure 17). Ce dernier enregistre les autres objets qui composent le serveur dans le bus RMI de Java. Ces objets sont: **WfProcessMgr**, **WfProcess**, **WfProcessInstance**, **WfActivity**, **WfActivityInstance**, **WfTransition**, **WfEvent**, **WfCondition**, **WfData**, **WfResource**, **WfParticipant**, **WfWorkList** et **WfDataInstance**. Chacun de ces objets permet la manipulation de ses propres attributs (voir le chapitre 2).

L'objet **WfWorkList** est un tableau des bons de travail. En effet, cet objet est un vecteur d'objets **WfWorkItem** (défini dans le chapitre 2). Le client workflows récupère sa corbeille, puis gère l'objet **WfWorkList**.

Dès son lancement, le serveur workflows récupère une connexion à la base de données. Puis, il transmet cette connexion aux objets qui le constituent sous forme d'un paramètre. Les objets qui composent le serveur utilisent le JDBC pour envoyer des requêtes SQL au SGBD d'Oracle. Ce dernier, en plus de gérer la base de données, exécute les requêtes et retourne les résultats à l'objet concerné.

4.5 Conclusion

Dans notre implémentation, nous avons fait l'hypothèse qu'une instance d'activité ne peut utiliser qu'une seule ressource (en général un programme). Notre modèle peut être

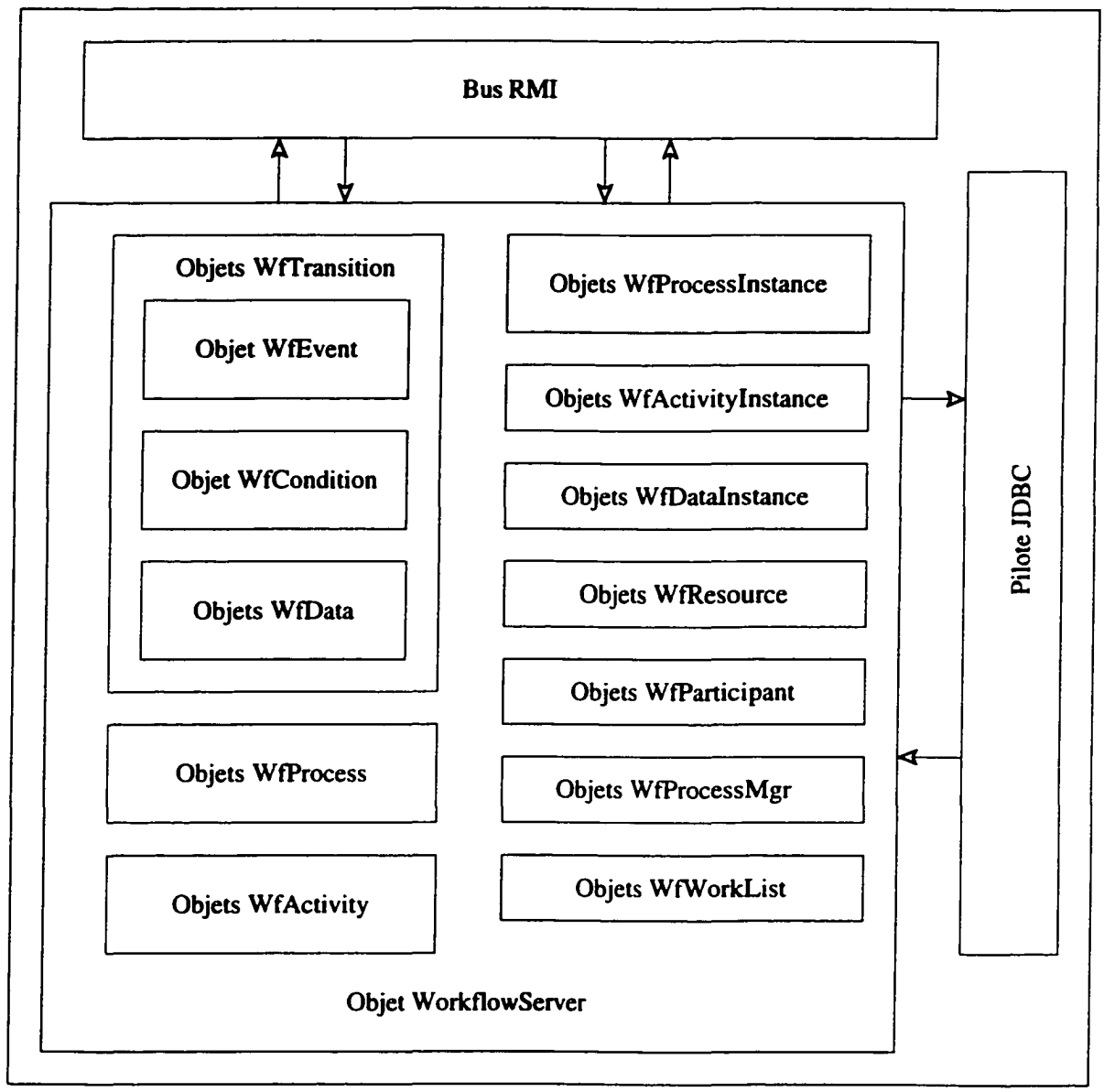

FIG. 17 - *Architecture du serveur workflows*

étendu en ajoutant une classe et une table pour l'allocation des ressources. Cette nouvelle classe serait en relation avec les classes **WfActivityInstance** et **WfResource**.

Conclusion

Dans ce chapitre, nous présentons d'abord la contribution de notre travail de recherche. Ensuite, nous critiquons les choix effectués dans ce mémoire. Enfin, nous dégageons une piste pour des travaux futurs de recherche.

Contributions

Les contributions apportées par notre travail de recherche sont regroupées en quatre points.

- Nous avons appliqué l'approche OMT pour l'analyse et la conception d'un système de gestion de workflows distribué. Ainsi, nous avons construit un méta-modèle et une architecture d'un système de gestion de workflows distribué.
- Nous avons conçu un système de gestion de workflows distribué flexible qui permet l'intégration du mode conception et du mode exécution d'un workflow. Cette flexibilité a été matérialisée par la création de trois classes qui représentent le mode exécution: **WfProcessInstance**, **WfActivityInstance** et **WfDataInstance**, qui sont respectivement des sous-classes des classes **WfProcess**,**WfActivity** et **WfData**. Pour les classes du mode exécution, nous avons ajouté la notion d'état aux objets **WfProcessInstance** et **WfActivityInstance**.
- Notre système de gestion des workflows distribué est développé en langage Java

(version 1.2), ce qui permet la portabilité du système. La communication entre les objets distants est réalisée via le bus de communication RMI.

- Nous avons choisi comme réservoir de données, une base de données Oracle centralisée pour tous les objets. Le concepteur de workflows est appelé à formaliser ses processus d'affaires et à enregistrer les spécifications workflow dans la base de données.

Critiques du travail

Les points forts de ce travail sont l'application d'une approche orientée objets dans la conception du système, la flexibilité du système, l'utilisation d'un langage orienté objets dans le développement du système et l'utilisation d'une base de données Oracle comme réservoir de données.

L'utilisation d'une approche orientée objets nous a permis de détecter les actions et les événements des différents composants du système de gestion de workflows distribué. Cette approche nous permet d'étendre les modèles développés.

Dans ce travail, nous estimons qu'il y a deux points faibles.

- Nous avons utilisé une architecture distribuée. L'architecture développée ne prend pas en considération les transactions workflow. Et comme suite logique, nous ne traitons pas la récupération des données après une panne.
- Le système de gestion de workflows distribué que nous avons développé ne traite pas les erreurs. En effet, il y a deux types d'erreurs dans un tel système : l'incohérence des données et les erreurs de conception de workflows (construction d'un cycle infini [17]).

Travaux futurs de recherche et perspectives

Les travaux futurs que nous suggérons se résument en ce qui suit.

- Le développement d'algorithmes qui gèrent les transactions workflow [2] et la récupération des données après une panne [22].
- La faiblesse du bus RMI a été démontrée dans la gestion des objets distants sur Internet, ce qui porte les développeurs Java à utiliser les *servlets* [14]. Ainsi, nous proposons la construction d'un système qui utilise le bus CORBA pour la gestion des objets distants.
- Pour tenir compte de tous les composants d'un système de gestion de workflows distribué, il faudrait développer une interface de formalisation des workflows, un outil d'administration et de vérification des workflows et éventuellement un coordonnateur des serveurs workflow.

Annexe A

Glossaire

Dans ce glossaire, nous donnons les définitions de termes utilisés dans ce mémoire. Pour les termes traduits de la langue anglaise, nous associons à chaque terme le terme anglais équivalent.

Activité (Activity) : Division d'une procédure. Le rôle de chaque activité est la transformation d'une donnée par un agent.

Bon de travail (Work Item) : Représentation du travail qu'un participant doit effectuer dans le cadre de sa participation à une procédure.

Client workflows : Module d'un système de gestion de workflows distribué qui permet l'accès aux serveurs workflows, la récupération de la liste des bons de travail et l'exécution des activitées.

Conception workflow : Transformation d'une procédure de travail en un workflow.

Corbeille (WorkList) : Liste des bons de travail qu'un participant est tenu de réaliser dans le cadre de sa participation aux différentes procédures.

Donnée workflow : Toute information structurée et manipulée par un système de gestion de workflows distribué.

Exécution workflow : Transformation d'une instance de procédure ou d'activité.

Formalisation workflow : Description d'un workflow selon les attributs de la base de données workflow.

Gestionnaire de corbeilles (Worklist Handler) : Programme permettant la récupération de la corbeille d'un participant.

Gestionnaire de workflows (Workflow Enactment Service) : Logiciel composé d'un ou de plusieurs moteurs de workflows qui servent à définir et gérer des procédures. Le gestionnaire de workflows permet aussi la gestion des données workflow.

Information workflow : Toute information non structurée relative à un workflow.

Instance d'activité : Objet d'une activité (par exemple, pour chaque instance du workflow « circuit de vente » nous avons une instance de l'activité « contacter client »).

Instance de workflow : Objet d'un workflow. Nous pouvons créer plusieurs instances d'un même workflow (par exemple, pour le workflow « circuit de vente » nous pouvons avoir plusieurs ventes).

Moteur de workflows (Workflow Process Engine) : Logiciel dont le rôle est la gestion de toute ou une partie d'une procédure d'un workflow.

Procédure (Process) : Ensemble d'activités et d'opérations reliées en série ou en parallèle permettant d'atteindre un objectif.

Procédure de travail : Procédure qui représente l'organisation et la politique d'une organisation.

Requête workflow : Appel d'une méthode appliquée à une instance d'une procédure ou à une instance d'activité.

Serveur workflow : Module d'un système de gestion de workflows distribué qui permet l'exécution des procédures de travail et la gestion des données workflow.

Spécification workflow : Description détaillée d'un workflow en langage naturel ou à l'aide d'une méthode formelle.

Système de gestion de workflows (Workflow Management System) : Ensemble de programmes et de données permettant de définir et de gérer des workflows.

Transaction workflow : Ensemble de requêtes workflow destinées à une instance d'un workflow.

Workflow (Workflow) : Représentation informatique d'une procédure de travail.

Annexe B

Description des tables

TAB. 15 – *Description des tables de la base de données*

Nom de table	Nom du champ	Type	Taille	Index
Activity				
	ProcessId	Numeric	10	
	ActivityName	Varchar	30	Yes
	ActivityId	Numeric	10	Yes
	ActivityDescription	Varchar	40	
	ParticipantId	Numeric	10	
	ResourceId	Numeric	10	
ActivityInstance				
	ProcessInstanceId	Numeric	10	
	ActivityInstanceId	Numeric	10	Yes
	ActivityId	Numeric	10	
	ActivityState	Numeric	10	
Condition				
	TransitionId	Numeric	10	
	ConditioName	Varchar	20	
	ConditionId	Numeric	10	Yes
	Data1Id	Numeric	10	
	Data2Id	Numeric	10	
	Criterion	Varchar	10	

TAB. 16 – *Description des tables de la base de données (suite)*

Nom de table	Nom du champ	Type	Taille	Index
Data				
	DataName	Varchar	30	
	DataId	Numeric	10	Yes
	DataDescription	Varchar	30	
	DataValue	Varchar	50	
	DataType	Numeric	10	
DataInstance				
	DataId	Numeric	10	
	DataInstanceId	Numeric	10	Yes
	DataValue	Varchar	50	
	ProcessInstanceId	Numeric	10	
Event				
	EventName	Varchar	30	
	EventId	Numeric	10	Yes
	EventType	Numeric	10	
Participant				
	ParticipantId	Numeric	10	Yes
	ParticipantName	Varchar	30	Yes
	ParticipantDescription	Varchar	40	
	ParticipantRole	Varchar	20	
Process				
	ProcessName	Varchar	20	Yes
	ProcessDescription	Varchar	30	
	ProcessVersion	Numeric	10	
	ProcessId	Numeric	10	Yes
ProcessInstance				
	PocessId	Numeric	10	
	ProcessInstanceId	Numeric	10	Yes
	ProcessState	Numeric	10	

TAB. 17 – *Description des tables de la base de données (suite)*

Nom de table	Nom du champ	Type	Taille	Index
ProcessMgr				
	ProcessMgrName	Varchar	20	
	ProcessMgrId	Numeric	10	Yes
	Description	Varchar	30	
	Version	Numeric	10	
Resource				
	ResourceId	Numeric	10	Yes
	ResourceName	Varchar	20	
	ResourceCategory	Numeric	10	
	ResourceLocation	Numeric	10	
	ResourceAllocated	Numeric	10	
Transition				
	ProcessId	Numeric	10	
	TransitionName	Varchar	20	
	TransitionId	Numeric	10	
	ScrActivityId	Numeric	10	
	DesActivityId	Numeric	10	
	Branch	Numeric	10	
	EventId	Numeric	10	
	ConditionId	Numeric	10	
WorkList				
	WorkName	Varchar	20	
	WorkId	Numeric	10	Yes
	ProcessInstanceId	Numeric	10	
	ActivityInstanceId	Numeric	10	
	ParticipantId	Numeric	10	
	ResourceId	Numeric	10	

Annexe C

Étude de cas : workflow circuit de vente

La figure 18 illustre la procédure de travail *circuit de vente*. L'objet de cette procédure est l'organisation et la réglementation d'un circuit de vente.

Cette procédure est appliquée dans le cadre d'une entreprise ayant un dépôt central de marchandise et un ensemble de points de vente en gros. Trois participants contribuent à la procédure : une sécretaire, un agent livreur-recouvreur et un agent pour la gestion de stock.

C.1 Description de la procédure de travail

La sécretaire peut recevoir des commandes de clients par téléphone ou par fax. Elle enregistre la commande. Si le crédit alloué au client (Quota) est inférieur à son solde alors l'agent livreur-recouvreur contacte le client pour lui demander le paiement d'au moins sa commande. Si le client s'excuse pour le paiement alors la commande est annulée. Si le crédit alloué au client est supérieur à son solde ou après règlement de la commande, le gestionnaire de stock consulte le stock. Si les produits commandés sont disponibles en

quantités demandées, la sécretaire planifie la livraison et à la livraison elle produit une facture client. S'il existe des produits où les quantités demandées sont inférieures aux quantités disponibles alors la sécretaire commande la marchandise du dépôt central. Le délai de livraison permis est de deux jours. Si le gestionnaire de stock reçoit la marchandise dans les deux jours, la sécretaire contacte le client pour demander la confirmation de sa commande. Si le client confirme la commande, elle planifie la livraison de la marchandise. Si le client refuse la livraison, elle annule la commande

C.2 Formalisation du workflow *circuit de vente*

Liste des gestionnaires de procédures

ProcessMgr	
ProcessMgrName	first manager
ProcessMgrId	1
Description	gestionnaire des workflows numéro 1
Version	1
End	

Liste des procédures

Process	
ProcessName	circuit vente
ProcessId	1
ProcessDescription	
ProcessVersion	1
End	

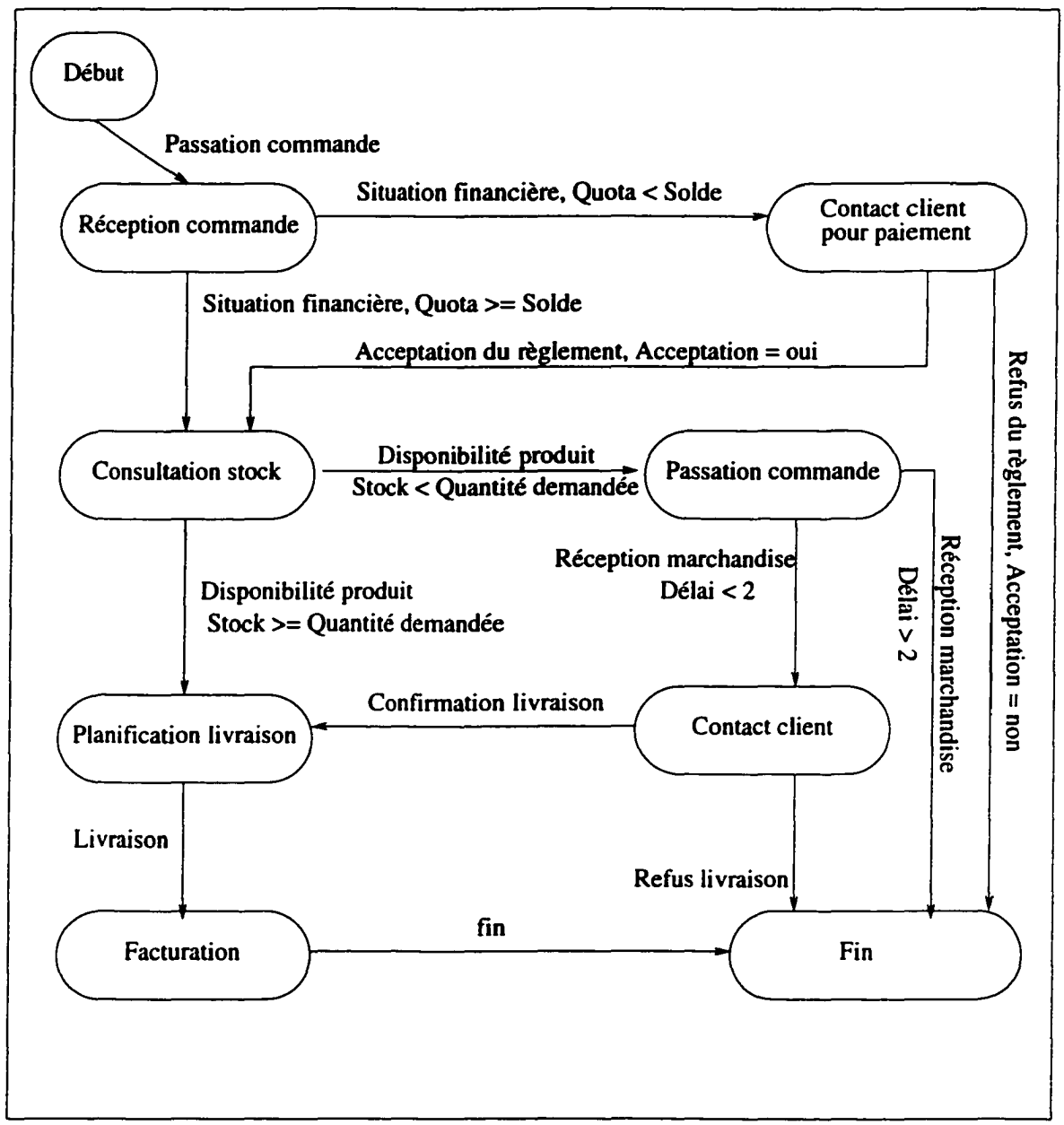

FIG. 18 – *Diagramme du workflow « circuit de vente »*

Liste des activités

Activity	
ProcessId	1
ActivityName	initial
ActivityId	1
ActivityDescription	initial
ParticipantId	1
ResourceId	0
End	
Activity	
ProcessId	1
ActivityName	réception commande
ActivityId	2
ActivityDescription	réception commande client
ParticipantId	2
ResourceId	1
End	
Activity	
ProcessId	1
ActivityName	contact client
ActivityId	3
ActivityDescription	demande de règlement
ParticipantId	3
ResourceId	3
End	
Activity	
ProcessId	1
ActivityName	planification livraison
ActivityId	4
ActivityDescription	mise à jour du plan des livraisons
ParticipantId	2
ResourceId	4
End	

Activity	
ProcessId	1
ActivityName	consultation stock
ActivityId	5
ActivityDescription	verifier si le stock disponible satisfait la commande
ParticipantId	4
ResourceId	5
End	
Activity	
ProcessId	1
ActivityName	contact client
ActivityId	6
ActivityDescription	vérifier avec le client s'il accepte la substitution des produits demandés
ParticipantId	2
ResourceId	0
End	
Activity	
ProcessId	1
ActivityName	facturation
ActivityId	7
ActivityDescription	
ParticipantId	2
ResourceId	6
End	
Activity	
ProcessId	1
ActivityName	fin
ActivityId	8
ActivityDescription	fin de la procédure
ParticipantId	0
ResourceId	0
End	

```
Activity
ProcessId           1
ActivityName        passation commande
ActivityId          9
ActivityDescription passation commande au dépôt central
ParticipantId       2
ResourceId          2
End
```

Liste des conditions

```
Condition
TransitionId    2
ConditionName   situation financière non permis
ConditionId     1
Data1Id         1
Data2Id         2
Criterion       >=
End
Condition
TransitionId    4
ConditionName   situation financière bonne
ConditionId     2
Data1Id         1
Data2Id         2
Criterion       <
End
Condition
TransitionId    3
ConditionName   acceptation du règlement
ConditionId     3
Data1Id         3
Data2Id         4
Criterion       =
End
```

```
Condition
TransitionId      9
ConditionName     produits non disponibles
ConditionId       4
Data1Id           7
Data2Id           8
Criterion         <
End
Condition
TransitionId      8
ConditionName     refus de règlement
ConditionId       5
Data1Id           3
Data2Id           4
Criterion         =
End
Condition
TransitionId      13
ConditionName     réception marchandise dans le délai
ConditionId       6
Data1Id           5
Data2Id           6
Criterion         >
End
Condition
TransitionId      10
ConditionName     réception marchandise après le délai
ConditionId       7
Data1Id           5
Data2Id           6
Criterion         <
End
Condition
TransitionId      5
ConditionName     produits disponibles
ConditionId       8
Data1Id           7
Data2Id           8
Criterion         >=
End
```

Liste des données

```
Data
DataName          quota
DataId            1
DataDescription   crédit alloué au client
DataValue         0
DataType          1
End
Data
DataName          solde du client
DataId            2
DataDescription
DataValue         0
DataType          1
End
Data
DataName          acceptation règlement
DataId            3
DataDescription
DataValue         oui
DataType          1
End
Data
DataName          valeur de l'acceptation
DataId            4
DataDescription
DataValue         oui
DataType          1
End
Data
DataName          délai de livraison
DataId            5
DataDescription
DataValue         2
DataType          2
End
```

Data
DataName délai de livraison permi
DataId 6
DataDescription
DataValue 2
DataType 2
End
Data
DataName stock
DataId 7
DataDescription
DataValue 0
DataType 2
End
Data
DataName quantité demandée
DataId 8
DataDescription
DataValue 0
DataType 2
End

Liste des événements

Event
EventName commande recu
EventId 1
EventType 2
End
Event
EventName acceptation règlement
EventId 2
EventType 2
End

Event
EventName refus règlement
EventId 3
EventType 2
End
Event
EventName disponibilité des produits
EventId 4
EventType 2
End
Event
EventName réception marchandise
EventId 5
EventType 2
End
Event
EventName confirmation livraison
EventId 6
EventType 2
End
Event
EventName refus livraison
EventId 7
EventType 2
End
Event
EventName livraison
EventId 8
EventType 2
End
Event
EventName fin
EventId 9
EventType 2
End

Liste des participants

Participant	
ParticipantId	1
ParticipantName	client
ParticipantDescription	participant externe
ParticipantRole	lancement de la procédure
End	
Participant	
ParticipantId	2
ParticipantName	Isabelle
ParticipantDescription	s'occupe des dossiers commerciaux
ParticipantRole	sécretaire
End	
Participant	
ParticipantId	3
ParticipantName	Michel
ParticipantDescription	s'occupe des livraisons et des recouvrements
ParticipantRole	livreur-recouvreur
End	
Participant	
ParticipantId	4
ParticipantName	Daniel
ParticipantDescription	s'occupe de la gestion de stock
ParticipantRole	gestion de stock
End	

Liste des ressources

Ressource	
ResourceId	1
ResourceName	gestion des commandes
ResourceCategory	1
ResourceLocation	19300101
ResourceAllocated	1
End	

Ressource
ResourceId 2
ResourceName tenue de stock
ResourceCategory 1
ResourceLocation 0
ResourceAllocated 1
End
Ressource
ResourceId 3
ResourceName fiche client
ResourceCategory 1
ResourceLocation 0
ResourceAllocated 1
End
Ressource
ResourceId 4
ResourceName plan des livraisons
ResourceCategory 1
ResourceLocation 0
ResourceAllocated 1
End
Ressource
ResourceId 5
ResourceName fiche de stock
ResourceCategory 1
ResourceLocation 19300127
ResourceAllocated 1
End
Ressource
ResourceId 6
ResourceName facture
ResourceCategory 1
ResourceLocation 19300204
ResourceAllocated 1
End

Ressource

ResourceId	7
ResourceName	produits de substitution
ResourceCategory	1
ResourceLocation	16300201
ResourceAllocated	1

End

Liste des transitions

Transition

ProcessId	1
TransitionName	passation commande
TransitionId	1
ScrActivityId	1
DesActivityId	2
Branch	0
EventId	0
ConditionId	0

End

Transition

ProcessId	1
TransitionName	situation financière n'est pas en règle
TransitionId	2
ScrActivityId	2
DesActivityId	3
Branch	0
EventId	1
ConditionId	1

End

Transition

ProcessId	1
TransitionName	situation financière en règle
TransitionId	4
ScrActivityId	2
DesActivityId	5
Branch	0
EventId	1
ConditionId	2

End

Transition

ProcessId	1
TransitionName	produit disponible
TransitionId	5
ScrActivityId	5
DesActivityId	4
Branch	0
EventId	4
ConditionId	8

End

Transition

ProcessId	1
TransitionName	facturation
TransitionId	6
ScrActivityId	4
DesActivityId	7
Branch	0
EventId	8
ConditionId	0

End

Transition

ProcessId	1
TransitionName	fin
TransitionId	7
ScrActivityId	7
DesActivityId	8
Branch	0
EventId	9
ConditionId	0

End

Transition

ProcessId	1
TransitionName	refus règlement
TransitionId	8
ScrActivityId	3
DesActivityId	8
Branch	0
EventId	3
ConditionId	5

End

Transition

ProcessId	1
TransitionName	produit non disponible
TransitionId	9
ScrActivityId	9
DesActivityId	5
Branch	0
EventId	4
ConditionId	4

End

Transition

ProcessId	1
TransitionName	réception marchandise
TransitionId	10
ScrActivityId	9
DesActivityId	6
Branch	0
EventId	5
ConditionId	4

End

Transition

ProcessId	1
TransitionName	confirmation livraison
TransitionId	11
ScrActivityId	6
DesActivityId	4
Branch	0
EventId	6
ConditionId	0

End

Transition	
ProcessId	1
TransitionName	règlement client
TransitionId	3
ScrActivityId	3
DesActivityId	4
Branch	0
EventId	2
ConditionId	3
End	
Transition	
ProcessId	1
TransitionName	refus règlement
TransitionId	12
ScrActivityId	6
DesActivityId	8
Branch	0
EventId	7
ConditionId	0
End	
Transition	
ProcessId	1
TransitionName	réception marchandise hors délai
TransitionId	13
ScrActivityId	9
DesActivityId	8
Branch	0
EventId	5
ConditionId	7
End	

Annexe D

Interfaces du système de gestion de workflows distribué

D.1 La création d'une instance de procédure

L'utilisateur peut utiliser l'interface client pour créer une nouvelle instance de procédure (voir la figure 19). Ainsi le serveur crée l'enregistrement suivant dans la base de données.

```
ProcessInstance
ProcessId          1
ProcessInstanceId  1
ProcessState       1
End
```

D.2 L'exécution d'un bon de travail

Pour exécuter un bon de travail, l'utilisateur clique sur le bon de travail. Il récupère la ressource allouée au bon de travail (voir la figure 20).

Soit un utilisateur qui exécute le bon de travail suivant :

```
ProcessInstance    1
WorkName
WorkId             1
ProcessInstanceId  1
ActivityInstanceId 5
ParticipantId      2
ResourceId
End
```

Ce bon de travail utilise l'instance de l'activité dont l'identifiant est « 5 » et qui a la définition suivante :

```
ActivityInstance
ProcessInstanceId  1
ActivityInstanceId 5
ActivityId         5
ActivityState      1
End
```

À l'exécution du bon de travail, la ressource génère les deux enregistrements suivants :

```
DataInstance
DataId             1
DataInstanceId     7
DataValue          6
rocessInstanceId   1
End
DataInstance
DataId             1
DataInstanceId     8
DataValue          5
rocessInstanceId   1
End
```

À la fin de l'exécution du bon de travail, le serveur récupère une transition (dont l'activité source est « 5 »), celle ayant comme identifiant « 9 » par exemple (voir l'annexe C). Puis, il évalue la condition correspondante (ayant comme identifiant « 4 »). La

valeur de retour de cette évaluation est « faux ». Le serveur récupère une autre transition, celle ayant comme identifiant « 5 »(voir l'annexe C). Puis, Il évalue la condition correspondante (ayant comme identifiant « 8 »). La valeur de retour de cette évaluation est « true ». Le serveur génère alors un bon de travail et une instance d'activité. La définition de ces objets est la suivante:

```
WorkList

  WorkName            planification livraison - 01
  WorkId              1
  ProcessInstanceId   1
  ActivityInstanceId  6
  ParticipantId       2
  ResourceId          4
  End

ActivityInstance

  ProcessInstanceId   1
  ActivityInstanceId  6
  ActivityId          4
  ActivityState       1
  End
```

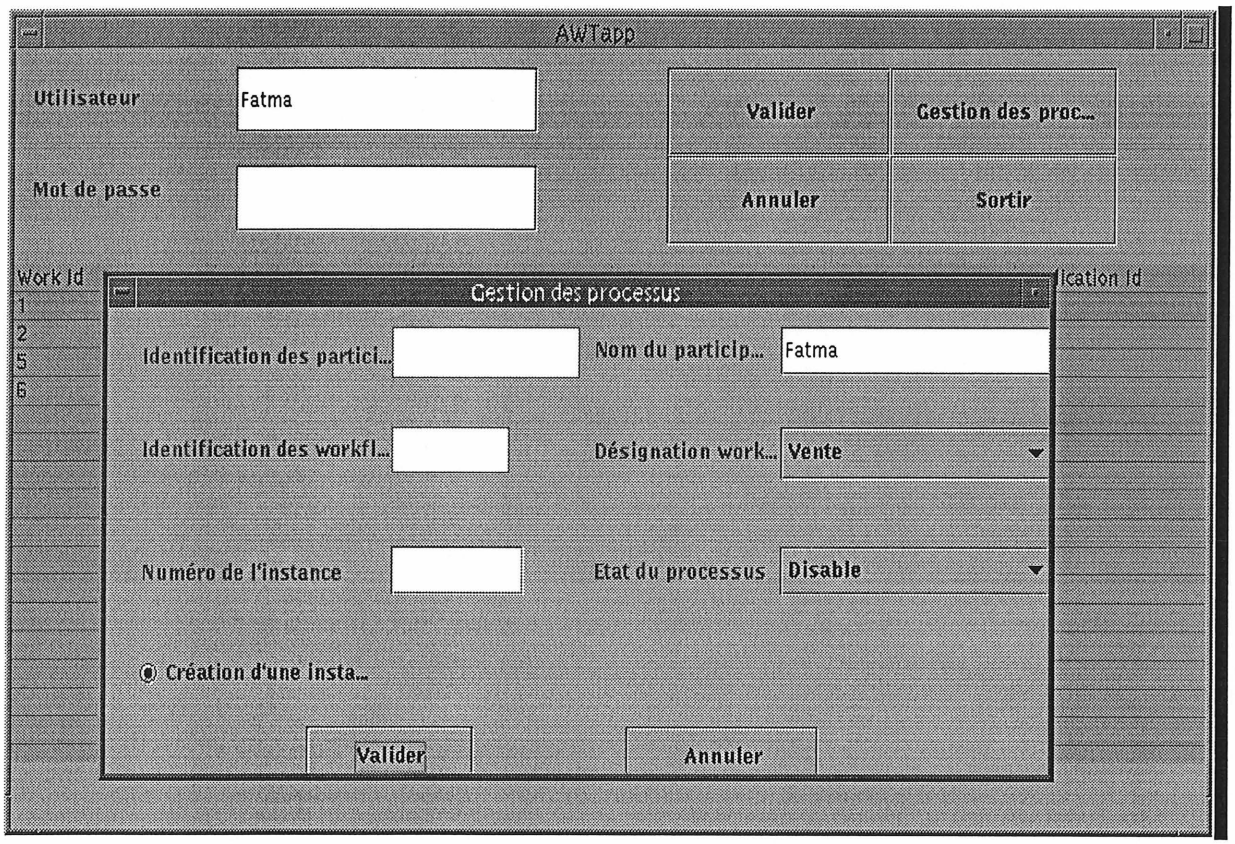

FIG. 19 – *Interface pour la gestion des instances de procédure*

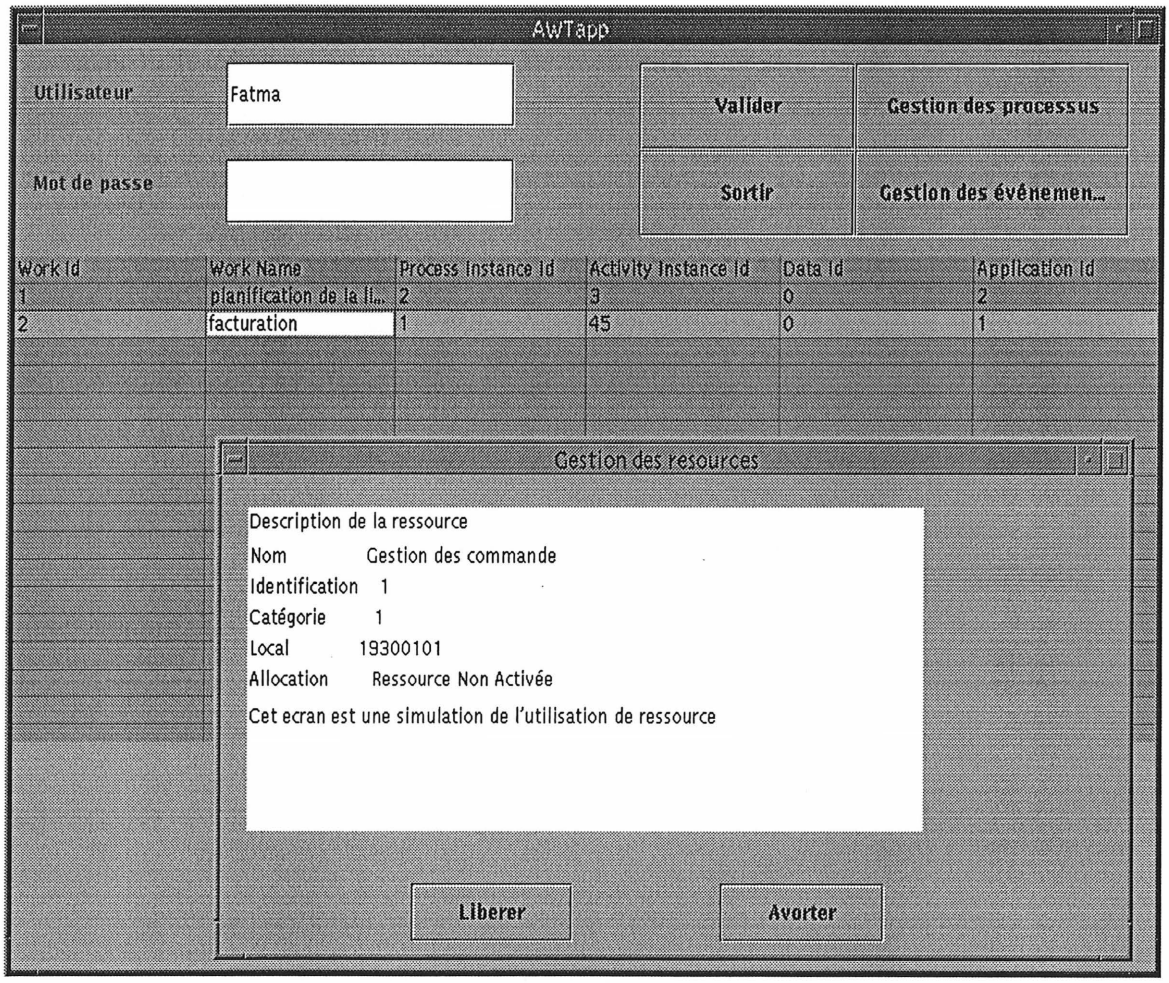

FIG. 20 – *Interface client workflows*

Bibliographie

[1] D. K. C. Chan and K. R. P. H. Leung. Valmont : A language for workflow programming. *Proceedings of the 31st International Conference on System Sciences*, January 1998. Kona Coast, Hawaii.

[2] A. Cichocki, A. Helal, M. Rusinkiwiez, and D. Woelk. *Workflow and Process Automation : Concepts and Technology.* Kluwer Academic Publishers, 1998.

[3] J. Desharnais, M. Frappier, and A. Mili. State transition diagrams. In *Handbook on architectures of information systems.* P. Bernus, K. Mertins, G. Schmidt, Eds., Springer Verlag, 1998.

[4] J. M. Geib et C. Gransart et P. Merle. *Corba : des concepts à la pratique.* Masson, 1997.

[5] D. Harel. On visual formalisms. *Communications of the ACM*, 31(5) :514–530, 1988.

[6] D. Hollingsworth. *Workflow Management Coalition, The Workflow Reference Model*, January 1995. www.wfmc.org.

[7] C. Horstmann and G. Cornell. *Core Java, Volume I-Fundamentals.* The SUNSOFT Press, Java series, 1997.

[8] IDHEAP. *Motown, générateur de workflow*, novembre 1996. www.unil.ch/idheap/motownf.htm.

[9] J. Jaffar, M. Maher, and G. Neumann. An architecture and prototype implementation of a system for individualised workflows in medical information systems. *Proceedings of the 32nd International Conference on System Sciences*, January 1999. Maui, Hawaii.

[10] R. Klamma, P. Peters, and M. Jarke. Workflow support for failure management in federated organisations. *Proceedings of the 31st International Conference on System Sciences*, January 1998. Kona Coast, Hawaii.

[11] M. M. Kwan and P. R. Balasubramanian. Adding Workflow Analysis Techniques to the IS Development Toolkit. *Proceedings of the 31st International Conference on System Sciences*, January 1998. Kona Coast, Hawaii.

[12] P. Lawrence. *Workflow Handbook 1997.* Wiley, 1997.

[13] K. Leung and J. Chung. The liaison workflow engine architecture. *Proceedings of the 32nd International Conference on System Sciences*, January 1999. Maui, Hawaii.

[14] S. McPherson. *Java Servlets and Serialisation with RMI*, September 1998. www.developper.java.sun.com/developper/technicalArticles/RMI/rmi/.

[15] OMG. *Task Management Common Facilities*, November 1995. www.OMG.org.

[16] OMG. *Workflow Management Facility*, July 1998. www.omg.org.

[17] S. Onoda, Y. Ikkai, T. Kobayashi, and N. Komoda. Definition of deadlock patterns for business processes workflow models. *Proceedings of the 32nd International Conference on System Sciences*, January 1999. Maui, Hawaii.

[18] Oracle Inc, 1999. *Oracle 8 Documentation.*

[19] S. Reddy. *Requirements for Simple Workflow Access Protocole*, August 1998. www.ics.uci.edu/~ietfswap/swapreq.txt.

[20] S. Rivard and J. Talbot. *Le développement de systèmes d'information, méthodes et outils.* Presses de l'Université du Québec, 1996.

[21] J. Rumbauch. *Modélisation et conception orientées objet.* Masson et Prentice-Hall International, 1995.

[22] H. Shuster, S. Jablonski, and C. Bubler. Client/server qualities: A basis for reliable distributed workflow management systems. In *Proceedings of the 17th International Conference on Distributed Computing Systems*, May 1997. Baltimore, Maryland.

[23] J. Siegel. OMG overview: CORBA and the OMA in Entreprise Computing. *Communications of the ACM*, 41(10) :37–43, 1998.

[24] D. Stirrup, M. Pearce, and G. Noakes. *XML Based Process Management Standard Launched by Workflow Management Coalition - Wf-XML*, July 1999. www.wfmc.com.

[25] Sun MicroSystem. *Java Remote Method Invocation Specification*, December 1999. www.java.sun.com.

[26] Sun MicroSystem. Lesson: JDBC basics, 2000. www.java.sun.com//docs/books/tutorial/jdbc/basics/.

[27] K. Swenson. *Simple Workflow Access Protocol (SWAP)*, August 1998. www.ics.uci.edu/~ietfswap/swap-prot.txt.

[28] M. Weske. Flexible modeling and execution of workflow activities. *Proceedings of the 31st International Conference on System Sciences*, January 1998. Kona Coast, Hawaii.

[29] WfMC, 1998. www.wfmc.org.

[30] Work Group 4. *Workflow Management Coalition, Workflow Standard-Interoperability Wf-XML Binding*, January 2000. www.AIIM.org.