Eléments de flexibilité des systèmes de workflow pour la définition et l'exécution de procédés coopératifs

Daniela Grigori

Département de formation doctorale en informatique **École doctorale IAEM Lorraine**
UFR STMIA

Eléments de flexibilité des systèmes de workflow pour la définition et l'exécution de procédés coopératifs

THÈSE

présentée et soutenue publiquement le 26 novembre 2001

pour l'obtention du

Doctorat de l'université Henri Poincaré – Nancy 1

(spécialité informatique)

par

Daniela Grigori

Composition du jury

Président :	M. Laurent Romary, Directeur de recherche INRIA, Loria – INRIA Lorraine
Rapporteurs :	M. Wojciech Cellary, Professeur à l'Université des Sciences Economiques de Poznañ Mme. Anne Doucet, Professeur à l'Université P. et M. Curie-Paris 6
Examinateurs :	M. Fabio Casati, Directeur de recherche, HP Laboratories, Palo Alto M. François Charoy, Maître de Conférence à l'Université Nancy 2 M. Claude Godart, Professeur à l'ESSTIN, Université Henri Poincaré, Nancy I

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503

Mis en page avec la classe thloria.

Résumé

Les systèmes actuels de workflow sont utilisés efficacement pour automatiser les procédés administratifs et de production, mais sont inadaptés pour les procédés coopératifs dans le domaine de la co-conception et/ou co-ingénierie.

Pour améliorer l'efficacité des systèmes de workflow pour le travail coopératif nous proposons :

- un modèle de workflow coopératif qui permet une exécution souple et flexible des procédés,
- des outils intelligents destinés à aider la compréhension et le bon déroulement du travail coopératif.

L'idée au coeur du modèle de workflow coopératif est que les procédés coopératifs se décrivent comme les procédés d'administration et de production, mais s'exécutent de manière différente. Ils nécessitent une plus grande flexibilité, à la fois du flot de contrôle et du flot de données. Pour permettre la flexibilité du flot de contrôle, nous proposons le concept d'anticipation qui permet à une activité de commencer à s'exécuter même si ses conditions de démarrage ne sont pas complètement satisfaites. L'anticipation augmente le parallélisme entre les activités et en améliore la qualité grâce au retour d'expérience rapide qu'elle favorise.

L'échange de données flexible entre activités est permis en encapsulant les activités dans des transactions coopératives. Ceci permet de dépasser le modèle d'enchaînement de boites noires, considéré par les workflows traditionnels, tout en maintenant le niveau de contrôle requis. Nous proposons une adaptation du modèle d'exécution du système de workflow pour permettre l'intégration du gestionnaire de transactions coopératives.

Pour permettre l'optimisation de l'exécution des procédés coopératifs, nous proposons des outils pour l'analyse, la prédiction et la prévention des exceptions. Ces outils utilisent des techniques de fouille de données et d'entrepôt de données appliquées aux données d'exécutions de procédés stockées dans un entrepôt de données.

Mots-clés: coopération, coordination, workflow, exceptions.

Abstract

Current workflow systems are efficiently used to automate administrative and production processes, but they show their limits when one wants to model the cooperative interactions as they occur in more interactive and creative processes, typically co-design and/or co-engineering processes. In order to improve the effectiveness of workflow systems for cooperative work we propose :

- a model of cooperative workflow which allows a flexible process execution,
- intelligent tools for process analysis, prediction and optimisation.

The core idea of the cooperative workflow model is that cooperative processes are described like administration and production processes, but are executed in a different way. They require a greater flexibility, at the same time of control flow and data flow. The control flow flexibility is based on the concept of anticipation that allows an activity to start its execution even if its activation conditions are not yet satisfied. Anticipation increases parallelism between activities and improves their execution quality by supporting rapid feedback. The flexible data exchange

between activities is allowed by encapsulating activities in cooperative transactions. This activity model surpasses the « black box » activity model, considered by traditional workflows, while maintaining the necessary level of control. We propose an adaptation of the workflow execution model to allow the integration of the workflow system with the cooperative transactions manager.

The second contribution of this thesis consists in a technique and suited tools for exceptions analysis, prediction and prevention. These tools use data mining techniques that are applied to process execution data stocked in a data warehouse.

Keywords: cooperation, coordination, workflow, exceptions

Remerciements

A l'issue de ces trois ans de travaux, le moment est venu pour moi, de remercier toutes les personnes qui m'ont entouré, pour mener à bien ce projet. La liste est longue et je vais essayer de n'oublier personne.

Je voudrais tout d'abord exprimer ma reconnaissance envers tous les membres du jury pour la grande attention qu'ils ont bien voulu porter à mon travail.

- Je tiens à remercier Claude Godart, mon directeur de thèse, qui m'a encadré et dirigé dans mes recherches tout au long de ces trois années. Je lui dis ma gratitude pour l'aide compétente qu'il m'a apportée, pour ses encouragements et pour la confiance qu'il m'a toujours témoignée.
- Mes plus chaleureux remerciements vont également à François Charoy, qui m'a encadré et dirigé dans mes recherches, pour sa disponibilité, pour les nombreuses discussions que nous eûmes ces années durant, pour ses critiques constructives, le tout accompagné d'un humour qui n'appartient qu'à lui. Son soutien fut constant et chaleureux, je l'en remercie vivement.
- Je remercie très sincèrement mes rapporteurs Anne Doucet et Wojciech Cellary pour avoir lu de manière approfondie ma thèse, pour leurs remarques constructives et leurs conseils.
- Je remercie Laurent Romary pour avoir accepté d'être président du jury, pour la lecture approfondie du manuscrit et pour ses suggestions.
- Je suis très reconnaissante à Fabio Casati, dont j'ai fait la connaissance lors d'un stage au centre de recherche Hewlett Packard Laboratories, à Palo Alto. Ce stage a marqué le début d'une collaboration fructueuse et d'une amitié sincère.

Je voudrais remercier France télécom R&D et en particulier Madame Hélène Saliou pour le support financier de ce travail qui a été réalisé en partie dans le cadre d'une CTI.

Fabio Casati, Ming Chen Shan et Umesh Dayal m'ont accueilli au sein de leur équipe au centre de recherche Hewlett Packard Laboratories pour un stage scientifique de trois mois. Je leur remercie pour l'accueuil chaleureux et pour m'avoir offert la possibilité de collaborer avec eux sur un sujet intéressant, dans un endroit ensoleillé.

Tous les membres de l'équipe ECOO ont permis de faire de cette expérience de thèse une expérience riche tant scientifiquement que humainement. Je les remercie tous beaucoup de leur aide, de leurs conseils et de leur soutien.

Merci à tous les amis que j'ai côtoyé durant tous ces années : Ferouz, Liliana, Horatiu, Dana, Sorin, Stefan, Radu, Luminita, Simona, Dmitri, Julia, Hala, Pascal, Olivier, Adrian, Bernard, Madalina et Rafael.

Enfin, et surtout, je voudrais remercier ma famille, en particulier mes parents et ma soeur, qui m'ont toujours soutenu et encouragé. Un merci spécial à mon fils Alexandru pour sa patience pendant cette thèse parfois envahissante, pour son intérêt pour les systèmes de workflow :-) et... pour énormément de raisons.

A mon fils, Alexandru, et à mes parents

Table des matières

Table des figures

Chapitre 1

Introduction

Le travail présenté dans cette thèse a pour cadre général le *travail coopératif assisté par ordinateur* (CSCW en anglais[1]). Il s'est déroulé principalement dans le cadre du projet ECOO du LORIA dont l'objectif est le développement de services pour l'hébergement d'équipes et d'entreprises distribuées(*Virtual Teams and Virtual Enterprises* en anglais), pour des applications créatives de type co-ingénierie et/ou co-conception.

Dans ce projet, l'effort porte principalement sur la coordination d'une équipe distribuée. L'approche qui y est développée intègre deux points de vue : la coordination implicite et la coordination explicite. L'idée de la « coordination implicite » est que, si les membres d'une coopération reçoivent la bonne information, au bon moment sur ce que font les autres, cette information génère une communication qui suffit à l'auto-coordination de l'équipe distribuée. A l'opposé, la coordination explicite part de l'idée qu'une bonne coordination nécessite la définition formelle d'un procédé et les moyens de la mettre en œuvre automatiquement. Si ces approches peuvent apparaître initialement comme des alternatives, notre expérience montre qu'elles peuvent se compléter efficacement, en synergie [God01].

Le travail présenté dans cette thèse se place dans le cadre de la coordination dite explicite. Notre objectif est d'offrir un cadre pour la coopération des membres d'une équipe distribuée incluant des outils pour la définition et la mise en œuvre des procédés.

Le concept d'équipe distribuée désigne le fait que de nombreux projets sont le résultat de la coopération de plusieurs acteurs, remplissant différents rôles, distribués dans le temps et dans l'espace. Au cours du projet, les différents partenaires alternent des périodes d'isolation avec des périodes d'interaction. Au cours de ces dernières, ils vont être amenés à coopérer en se communiquant des informations à la fois partielles[2] et intermédiaires[3] pour pouvoir réaliser leur tâche. Dans le cas d'un projet de conception relativement complexe, les partenaires ne possèdent pas la maîtrise et la connaissance intégrale de l'activité globale exécutée. Il est nécessaire d'avoir une représentation commune du projet, un plan, qui sert à la coordination des partenaires. Ce plan ne peut pas être complètement défini a l'avance ; il doit pouvoir être modifié pendant la durée du projet directement par ses participants.

Différents domaines de recherche proposent des solutions pour la coordination des équipes distribuées, en particulier : les environnements de développement de logiciel, les systèmes de workflow et les collecticiels. Ils abordent les problèmes liés à la coordination et à la coopération

[1] CSCW : Computer Supported Cooperative Work

[2] Ce sont des éléments d'information permettant aux autres partenaires du projet de démarrer leur activité au plus tôt sans devoir attendre que les informations complètes soient publiées

[3] Ces échanges ayant lieu en cours d'activité, les informations partagées sont potentiellement sujettes à des nouvelles modifications

de différentes manières et pour des cadres applicatifs différents.

Les environnements de développement centrés procédés proposent des solutions pour la gestion des procédés logiciel en prenant en compte deux dimensions : les données et les dépendances entre les activités. Le support pour la coordination est en général assez limité, leur objectif principal étant la gestion des produits logiciel. D'autre part, quoi que les modèles développés dans le domaine de développement de logiciels pourraient être généralisés pour supporter d'autres procédés d'ingénierie [Ost98], les systèmes actuels ont des caractéristiques et des limites qui empêchent leur acceptation hors de leur domaine d'origine.

Les collecticiels permettent la coopération à travers des données partagées pour les procédés non-structurés. Ces logiciels ignorent les aspects organisationnels et n'offrent pas de mécanismes pour mesurer l'avancement d'un projet. La plupart des systèmes n'incluent pas de support pour la modélisation et la gestion des dépendances entre activités, leur objectif principal étant de capter des pratiques réelles de travail.

Les systèmes de workflow se focalisent sur la coordination de procédés très structurés. Ils incluent des outils pour la modélisation, l'exécution et la surveillance de ces procédés. Cependant, ces systèmes ne prennent pas en compte la coopération entre activités à travers des données partagées.

Pour dépasser ces limites, des systèmes de workflow flexibles et adaptables, et, dans le domaine des collecticiels, des systèmes qui combinent des fonctionnalités de workflow basiques avec la coopération asynchrone sur des documents partagés ont été proposés. Mais, il n'existe pas encore une approche qui fournisse un support extensif intégrant les divers aspects des procédés coopératifs, en particulier la coopération à travers les données partagées et la coordination.

Dans cette thèse nous proposons un modèle de workflow coopératif supportant la coordination des procédés coopératifs. L'idée de base de ce modèle est que les procédés coopératifs se décrivent comme des procédés de type administratif ou de production (qui sont efficacement supportés par les systèmes de workflow traditionnels), mais qu'ils s'exécutent de façon différente, nécessitant une plus grande flexibilité, à la fois des flots de contrôle et des flots de données. La flexibilité du flot de contrôle est assurée en introduisant le concept d'« anticipation » qui permet à une activité de commencer à s'exécuter même si ses conditions d'exécution ne sont pas parfaitement satisfaites, comme cela se passe dans la réalité. L'anticipation permet plus de parallélisme entre les activités et un retour d'expérience rapide. La flexibilité du flot de données est assurée en encapsulant les activités dans des transactions coopératives. Ceci permet de dépasser les limites du modèle « activités - boîtes noires » considérées par les workflows traditionnels, tout en maintenant le niveau de sécurité requis.

Une deuxième contribution de cette thèse est la définition d'une stratégie d'optimisation des procédés basée sur la compréhension et la prédiction des exceptions. J'ai démarré ce travail au Centre de recherche Hewlett Packard à Palo Alto lors d'un séjour de 3 mois en collaboration avec Fabio Casati, Umesh Dayal et Ming Chen Shan. Le travail et la collaboration ont été ensuite continués depuis Nancy. Cette direction de recherche est motivée par le manque d'outils intelligents intégrés aux systèmes de workflow pour permettre l'optimisation des procédés.

Les systèmes de workflow journalisent une grande quantité d'information comme la date de démarrage et de terminaison du procédé et des activités, les données consommées et produites par chaque activité, les pannes et les autres exceptions, et l'attribution des ressources à chaque activité. Ces informations ne sont pas exploitables directement par l'utilisateur. L'idée de base de notre méthode d'analyse et d'optimisation est d'analyser des histoires d'exécutions passées pour en extraire des régularités qui correspondent à l'apparition d'exceptions, cela en utilisant des techniques de fouille de données.

1.1 Organisation de la thèse

Dans le chapitre 2 nous étudions la problématique de la coordination de procédés coopératifs. Nous montrons la nécessité d'un nouveau concept de workflow coopératif, en analysant les limites des modèles traditionnels de workflow. Ensuite, nous identifions les besoins de flexibilité d'un workflow coopératif concernant l'exécution et l'échange de données. La seconde partie de ce chapitre présente les principes directeurs qui nous ont guidés dans la définition de notre modèle de workflow.

Dans le chapitre 3, nous dressons un état de l'art en considérant les trois approches introduites ci-dessus :

- dans le domaine des environnements de développement centré procédé, nous essayons de cerner comment les fonctionnalités de gestion de procédé et des données sont intégrées.
- dans le domaine des systèmes de workflow, nous présentons les approches pour des workflows flexibles et transactionnels et nous déterminons dans quelle mesure ils permettent l'exécution flexible des activités coopératives et l'échange de données entre elles.
- dans le domaine des collecticiels, nous nous intéressons au support de la coopération à travers les données partagées.

Les résultats de nos travaux concernant le support à la coordination des activités coopératives sont présentés au chapitre 4. Nous y développons un nouveau modèle de workflow coopératif. Ce modèle permet de décrire un procédé en utilisant un modèle simple, similaire à celui utilisé pour les procédés administratifs ou de production. Nous commençons tout d'abord pas présenter ce modèle, ensuite nous décrivons son interprétation coopérative par un moteur de workflow adapté. La dernière partie du chapitre présente une proposition pour la flexibilité du flot de données basée sur l'intégration d'un gestionnaire de transactions coopératives avec le système de workflow.

Les propositions pour la flexibilité des workflows coopératifs ont fait l'objet de plusieurs mises en œuvre qui sont présentées au chapitre 5. La première mise en œuvre réalise l'intégration d'un moteur de workflow flexible dans le prototype Motu. La seconde mise en œuvre intègre des services de transactions coopératives dans un système de workflow commercial.

Le chapitre 6 présente une stratégie d'optimisation des procédés basée sur la compréhension, la prédiction et la prévention des exceptions. Nous y décrivons l'implantation d'un outil qui met en œuvre cette stratégie et nous décrivons les résultats expérimentaux obtenus.

Le dernier chapitre dresse le bilan de notre travail par rapport aux besoins que nous avons présentés lors de la problématique. Nous terminons cette thèse en présentant les directions de recherche auxquelles nous nous intéressons dans le futur.

Chapitre 2

Problématique

2.1 Cadre du travail

L'évolution actuelle des réseaux et du web permet à des groupes de personnes de coopérer pour former des équipes distribuées. Ce concept désigne un groupe de personnes temporellement et/ou géographiquement distribuées et rassemblées autour d'un projet. Ainsi, des personnes avec des compétences complémentaires peuvent se regrouper pour réaliser un projet qui n'est pas à la portée d'une seule personne. Ce coopération permet de conjuger ls efforts de chacue participant pour atteindre un objectif commun. Pour ce faire, les activités des différents participants ne s'exécutent pas de manière isolée, mais intergissent au cours de leur exécution en partageant des données communes. Une représentation explicite du procédé qui contient la description des tâches, leur enchaînement et leur affectation en fonction des qualifications des personnes assure la coordination des participants. Le support pour l'exécution et le contrôle du procédé apporte une synergie qui assure que l'objectif est atteint efficacement.

Notre objectif est d'outiller les membres d'une équipe distribuée pour leur permettre de définir, exécuter et faire évoluer les procédés que l'équipe distribuée doit suivre pour atteindre son objectif. L'idée générale est de définir un modèle de procédé « enactable »[4], dans l'idée des modèles de « workflow » définis par la WfMC [5], mais qui soit adapté aux applications créatives qui nous intéressent.

Pour cela, il est nécessaire de développer un nouveau concept de workflow. En effet, les modèles de workflow actuels sont trop rigides, ne fournissent pas la flexibilité nécessaire à la subtilité des interactions qui existent dans les applications créatives.

Notre objectif est donc de développer un modèle de workflow qui conserve autant que possible les qualités des modèles traditionnels, en particulier leur simplicité de définition et d'utilisation, mais qui les généralisent à destination des domaines d'application qui nous intéressent (co-ingénierie et co-conception).

Le reste de ce chapitre est organisé comme suit. La section 2.2 introduit brièvement l'idée de workflow traditionnel, en montre les avantages et les limites, et en particulier insiste sur le besoin de flexibilité. Puis la section 2.3 donne les principes directeurs qui ont guidé ce travail. La dernière section synthétise notre problématique.

[4] enactable : exécutable sous le contrôle d'agents humains
[5] Workflow Management Coalition (www.wfmc.org)

2.2 Limites des modèles de workflow actuels, besoin d'une plus grande flexibilité

Nous introduisons les concepts et la terminologie de base pour les systèmes de workflow à partir du modèle et glossaire de workflow de la WFMC [Coa95], [Coa99], [Coa96].

2.2.1 Workflow

Une *définition de procédé workflow* (ou plus simplement un *modèle de procédé*) est la représentation formelle d'un procédé. Un modèle de procédé est composé d'un ensemble de sous-procédés et de activités élémentaires (taches) organisés dans un graphe orienté (le *flot de contrôle*) qui définit l'ordre d'exécution des activités dans le procédé. Les arcs dans le graphe peuvent être étiquetés avec des prédicats de transition définis sur les données de procédé. La signification d'un tel arc est la suivante : quand une activité est terminée, les taches connectées aux arcs sortants sont exécutées seulement si les prédicats de transitions sont évalués à vrai. Quelques exemple de modèle de procédés seront présentés dans la thèse (par exemple, les figures 2.1 et 2.3).

Les *données pertinentes pour les procédés* son des données créées et utilisées par une instance de procédé. Ces données sont mises à la disposition des sous-procédés et des activités. Le système de gestion de workflow les utilise aussi pour évaluer les prédicats de transitions. La plus part des systèmes de workflow définissent la circulation des données entre activités en utilisant des connecteurs de données. Un connecteur relie l'activité qui produit la donnée avec l'activité qui consomme la données (*flot de données*). Le flot de données n'est pas indépendant du flot de contrôle, il respecte des restrictions par rapport à ce dernier. En général, les activités sont exécutées atomiquement, et les modifications des données sont rendues visibles à la fin de l'exécution des activités.

Pour exécuter un procédé, le WfMS créé une instance de procédé à partir du modèle de procédé. Un modèle peut être instancié plusieurs fois et plusieurs instances peuvent être exécutées en même temps. Le WfMS programme les activités pour exécution (conformemment au flot de contrôle) et les affecte pour exécution aux acteurs humains ou automatisés.

La plupart des systèmes de workflow définissent un schéma organisationnel qui décrit la structure de l'organisation. Les acteurs sont groupés par compétences ou par l'unité organisationnelle à laquelle ils appartiennent. Dans la définition d'un modèle de procédé, les procédés et les activités sont associés aux éléments du schéma organisationnel (rôles et unités organisationnelles) et pas aux agents particuliers.

Au moment de l'exécution, quand une activité est choisie pour l'exécution, le système de gestion de workflow détermine tous les agents qui ont la permission de l'exécuter et insère l'activité dans leurs *listes de travail*. Quand un acteur choisit l'activité de sa liste de travail, l'activité est supprimée des listes de travail des autres agents. Quand l'exécution d'une activité est terminée, le système traite la nouvelle activité qui doit être activée.

2.2.2 Avantages et limites des modèles de workflow traditionnels

Les remarques ci-dessous sont à considérer par rapport aux définitions de la WfMC [Coa95], [Coa99] qui nous sert de référence ici. Il est clair qu'elles sont à relativiser en fonction d'un produit donné et que bon nombre de travaux de recherche ont été réalisés pour dépasser ces limites. Ils seront considérés plus loin dans la section 3.

Le concept de workflow a été introduit pour modéliser et exécuter des procédés administratifs et de production. Le principal intérêt de l'approche est de séparer clairement la modélisation de

« la logique de l'entreprise » des programmes de l'entreprise, comme le développement des bases de données dans les années 70 a permis de déconnecter la modélisation des données d'une entreprise des programmes de cette entreprise.

Cela permet en particulier un découplage entre le modèle de procédés et le modèle organisationnel, ce qui simplifie et améliore la gestion des ressources : lorsqu'une activité est exécutable, elle peut être automatiquement ajoutée, avec les ressources nécessaires, à la liste des tâches des agents qualifiés disponibles, et lorsqu'un agent l'a choisie, elle est automatiquement retirée des listes des tâches des autres agents.

Une autre raison du succès des systèmes de workflow est leur « simplicité d'utilisation » grâce à un langage graphique de description de procédés et une interface de contrôle des tâches tout aussi simple.

L'inconvénient majeur associé à ce modèle est sa rigidité. Dans les applications administratives et de production, il s'agit de la difficulté à gérer les cas particuliers et les exceptions. Dans les applications créatives, il s'agit de l'inadaptation des modèles et systèmes traditionnels à supporter les interactions coopératives. La section suivante approfondit ces limites et caractérise le besoin de flexibilité correspondant.

2.2.3 Besoins de flexibilité pour un workflow coopératif

Besoin de flexibilité du flot de contrôle

Anticipation de l'exécution d'une activité, retour d'expérience rapide Les modèles de workflow actuels imposent des dépendances entre activités du type *fin - démarrage*, c'est à dire qu'une activité ne peut pas démarrer tant que l'activité précédente n'est pas terminée.

Cette sémantique n'est pas adaptée aux besoins des applications créatives de type co-ingénierie ou co-conception. Considérons par exemple un processus d'ingénierie dans le domaine du bâtiment (figure 2.1), auquel participent un architecte, un maître d'ouvrage et une personne du bureau d'études (BET). Le maître d'ouvrage est en charge de la collecte des plans administratifs. Il les transmet au BET qui les utilise pour réaliser les plans techniques. En même temps, l'architecte prend des photos et dessine le plan architectural. Finalement, le BET vérifie qu'il n'y a pas d'incohérences entre le plan architectural et le plan technique, et les valide.

A un certain niveau d'abstraction, le procédé peut être décrit comme dans la figure 2.1. Interprété avec une sémantique classique, ce modèle impliquerait que l'activité de validation ne pourrait commencer tant que les activités qui produisent le plan technique et architectural ne sont pas terminées. Si le procédé est décrit de la sorte dans les procédures architecturales (que l'on trouve dans les manuels d'architecte), en réalité, on peut observer que les activités se superposent, s'exécutent en parallèle en s'échangeant des résultats intermédiaires.

Les activités « Plans architecturaux » et « Plans techniques » produisent plusieurs versions des plans et peuvent les échanger en cours d'exécution. Le BET peut commencer son activité plus tôt, invalider rapidement les propositions de l'architecte et du BET qui posent des problèmes et ainsi fournir à l'architecte un retour rapide sur ses premières propositions. Mais, il n'en reste pas moins que, pour valider complètement les plans, il doit attendre la fin des deux activités précédentes.

Un autre exemple d'activité qui peut anticiper par rapport à la définition du procédé est l'activité « Plans techniques » qui peut commencer dès que l'acteur en charge de l'activité « Plans administratifs » précédente met à sa disposition une partie des plans (par exemple de canalisation électrique, etc.).

Ce discours est illustré par les exécutions décrites dans la figure 2.1 qui représentent deux

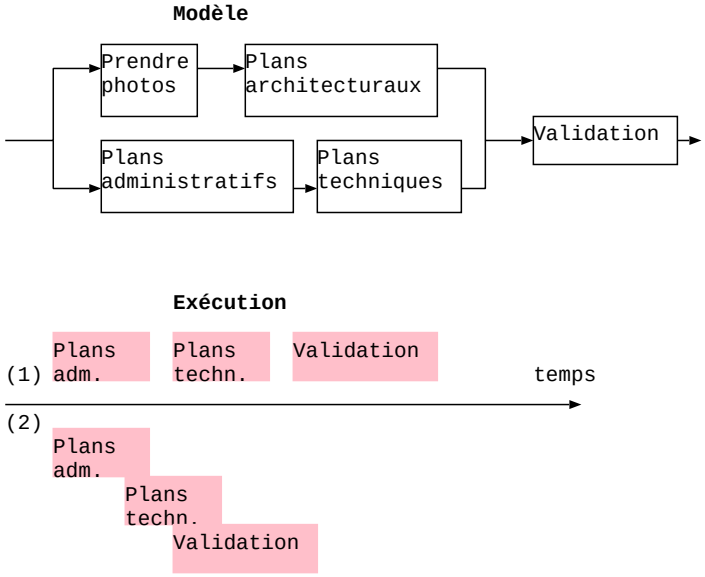


FIG. 2.1 – Exécution rigide (1) ou flexible (2)

exécutions (interprétations) possibles de la branche inférieure du procédé décrit au dessus : la première exécution (1) est une exécution strictement séquentielle et correspond à une interprétation traditionnelle de ce modèle, tandis que la deuxième exécution (2) représente une exécution « coopérative » (les exécutions des activités se superposent dans le temps).

Clairement, l'exécution séquentielle est trop contraignante, voir complètement irréaliste pour un procédé coopératif. Certes, on pourrait essayer d'affiner cette modélisation en bouclant sur certaines parties du procédé, mais là encore l'expérience montre que le modèle reste trop rigide, et difficilement applicable [God99], ne serait-ce que parce qu'une activité qui prend du retard dans une boucle a un impact sur toutes les autres. En d'autres termes, il n'est pas possible de modéliser les comportements sous-jacents au procédé ci-dessus avec la sémantique d'un modèle de workflow traditionnel.

Dans les applications coopératives, les activités se superposent et commencent leur travail avec des résultats intermédiaires (en opposition aux résultats finaux) des activités précédentes, même si toutes les conditions pour leur exécution ne sont pas complètement satisfaites, par exemple si :

- quelques unes des activités précédentes ne sont pas encore terminées, mais ont fourni des brouillons, des résultats partiels ;
- les conditions de démarrage sont actuellement évaluées à faux ou ne peuvent pas être évaluées (un certain visa manque, certains tests n'ont pas été passés) ;
- les données d'entrée ne sont pas complètes (mais les données disponibles sont suffisantes pour commencer à travailler).

Pour généraliser, un WfMS[6] pour les applications coopératives doit permettre un support informatique formel et efficace, comme un WfMS traditionnel pour les applications traditionnelles, tout en permettant une certaine flexibilité dans l'interprétation du modèle, comme cela se passe dans la réalité.

[6] système de gestion de workflow (Workflow Management System,)

Besoin de flexibilité du flot de données

Dans les modèles de workflow traditionnels, l'échange des données entre activités est, soit non considéré, et la cohérence des données reste donc complètement à la charge des programmeurs d'application, soit trop restrictif. Dans ce second cas, les activités sont considérées comme des boîtes noires qui consomment toutes leurs données d'entrée au démarrage et rendent visibles leurs résultats à la fin de leur exécution. En général, cela est mis en oeuvre en encapsulant les activités dans des transactions traditionnelles (ACID[7]). Une autre contrainte est qu'une activité peut lire une donnée d'une autre activité seulement si les deux activités sont connectées par le flot de contrôle.

Ces restrictions sont trop fortes pour les types d'interactions qui nous intéressent dans les applications coopératives.

Considérons l'exemple de suivi d'erreurs suivant, décrit dans la figure 2.2. Les deux branches contiennent chacune une activité de modification de code suivie d'une activité de test unitaire. La dernière activité est le test d'intégration.

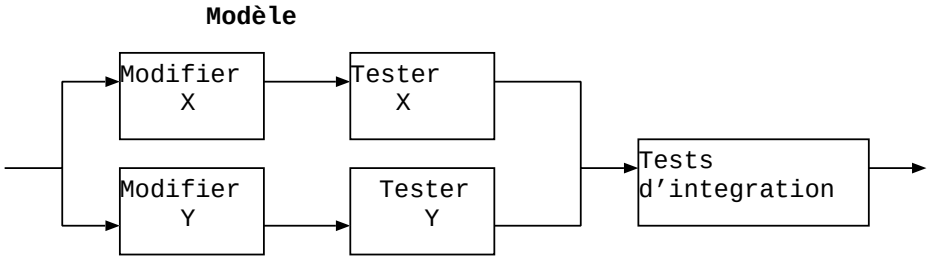


FIG. 2.2 – Procédé de suivi d'erreurs

L'activité de modification a comme données d'entrée le code de la classe X et un rapport d'erreur. On suppose que pendant l'exécution, l'acteur en charge de l'activité « Modifier X » se rend compte qu'il a besoin de modifier le module Z, qui est utilisé par la classe X. Même si cette donnée ne figure pas parmi les données d'entrée de l'activité, l'acteur correspondant devrait pouvoir y accéder. De plus, si l'acteur de « Modifier Y » veut également la modifier, il ne doit pas être bloqué par la première activité, qui peut être de longue durée. D'autre part, l'activité « Modifier X » peut publier plusieurs versions successives de ses données de sortie (du module X). L'activité de test devrait pouvoir être notifiée sur la production de la première version, et l'acteur en charge pourrait décider de commencer son travail à partir de ce résultat. Ensuite, il devrait pouvoir se synchroniser, à son initiative, avec les nouvelles versions arrivant et obligatoirement avec la dernière version.

Il est clair que laisser la modélisation de ce genre de comportement complètement à la charge des programmeurs d'application est risqué, et même impossible car ce genre d'interaction est généralement non totalement prévisible. Encapsuler les activités dans une transaction traditionnelle (ACID) empêche toute interaction en cours d'exécution, donc la réalisation de ce scénario.

Il est nécessaire de trouver un nouveau modèle de gestion de flot de données aussi simple à utiliser et aussi sûr qu'un modèle de transaction traditionnel, mais suffisamment flexible pour permettre des scénarios comme celui défini ci-dessus. En particulier, il doit :

– permettre aux activités de s'échanger des données en cours d'exécution (ce ne sont plus de boîtes noires),

[7] ACID dénote les propriétés suivantes : Atomicité, Cohérence, Isolation, Durabilité

- permettre à des activités situées dans des branches parallèles de modifier les mêmes données,
- ne pas imposer que tous les flots de données soient complètement définis à l'avance (pour pouvoir supporter les styles de travail opportuniste).

Besoin d'augmenter le parallélisme entre procédés

Considérons le cas où une activité dans un procédé est modélisée comme un sous-procédé. Dans les WfMS actuels, le sous-procédé est aussi une « boîte noire », c'est à dire que les données produites ne sont disponibles aux activités du procédé principal qu'après la terminaison de toutes ses activités.

Pour augmenter le parallélisme entre un procédé et ses sous-procédés, les données produites par les activités du sous-procédé devraient être mises à la disposition du procédé principal dès qu'elles sont produites. Une solution serait d'inclure toutes les activités dans un seul procédé, mais dans ce cas, on perd la modularité du modèle. Le problème est d'augmenter le parallélisme entre un procédé et ses sous-procédés en préservant la modularité.

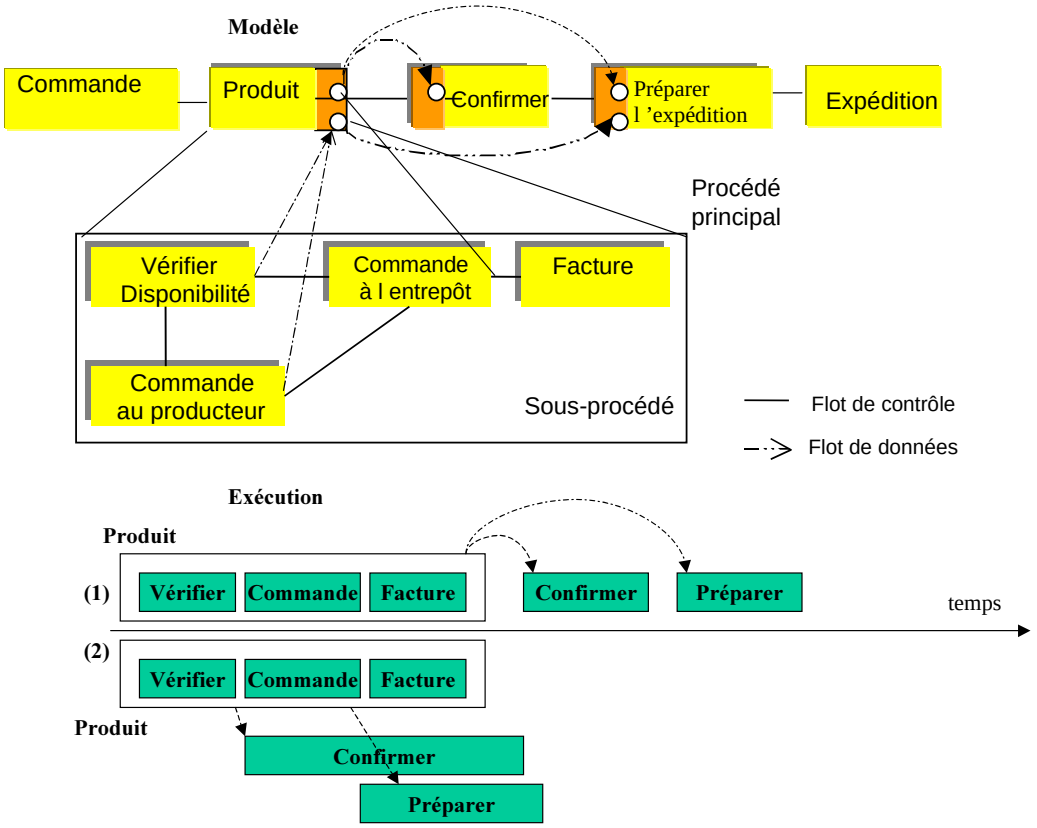


FIG. 2.3 – Parallélisme entre procédés

Cela est vrai dans les applications coopératives, mais également dans les applications de production comme la figure 2.3, prise dans [Hag99] l'illustre. Elle représente un procédé typique de gestion de la livraison des produits par une compagnie de vente au détail. L'activité « Produit » gère la commande courante ; elle peut être implémentée comme un sous-procédé qui vérifie si

l'élément est disponible, autrement il est commandé ; finalement une facture est produite. Les données de sortie du sous-procédé sont l'entrepôt où le produit est disponible et la date de livraison. Pour accélérer l'exécution du procédé, on pourrait préparer l'accusé de réception pour le client, dès qu'on connaît la date où le produit sera en stock ou la date à laquelle il sera reçu du producteur. De même, la préparation de l'expédition pourrait être lancée dès qu'on connaît l'entrepôt qui livrera le produit et la date à laquelle il sera disponible.

La figure 2.3(1) montre une exécution classique, séquentielle dans laquelle les activités sont des boîtes noires ; dans l'exécution (2) le parallélisme est augmenté ; les activités se superposent et rendent visibles leurs résultats pendant leur exécution. Le sous-procédé rend visible les résultats de ses activités.

Besoin de support pour l'évolution du modèle

La définition et la gestion des procédés est une tâche critique et longue dans chaque organisation qui veut implanter un système de workflow. En effet, alors que définir un procédé est techniquement facile, définir de « bons » procédés est extrêmement difficile, et le contrôle des procédés peut être compliqué s'il n'est pas correctement modélisé.

Pour faciliter l'acceptation des systèmes de workflow, son intégration doit être mise en oeuvre de façon graduelle. Le passage de l'exécution d'un procédé sans utiliser un support informatique pour la coordination, à l'exécution contrôlée par un système de workflow est une étape trop grande pour être effectuée en une seule fois. Un système de workflow peut aider les utilisateurs à faire cette transition de deux façons :

- **Définition incomplète et modifications dynamiques**
  La première étape est la définition d'un modèle qui décrit l'enchaînement des activités mais qui permet son exécution de façon très flexible. Les utilisateurs peuvent démarrer l'exécution du procédé même avec une spécification incomplète.
  Ensuite, pendant l'exécution du procédé il doit être possible de modifier et compléter sa définition. Les opérations de modification dynamique doivent être aussi simples que celles pour la définition statique. En même temps, les opérations ne doivent pas conduire à des erreurs d'exécution ou à des états incohérents.
- **Analyse et amélioration du procédé**
  La deuxième étape dans la démarche d'automatisation du procédé est d'assurer que le procédé réel suit le modèle ou, ce qui est équivalent, que le modèle reflète la réalité. L'analyse des traces d'exécutions peut aider à comprendre la manière dont les gens travaillent et vérifier les hypothèses faites pendant la modélisation. En particulier, l'un des problèmes les plus difficiles est celui de la gestion des exceptions (déviations par rapport à l'exécution optimale du procédé). Tandis que la recherche dans le domaine du workflow s'est concentrée sur le problème de modélisation et de traitement des exceptions, peu de travail a été effectué pour comprendre les causes des exceptions et les prévenir. Les systèmes de workflow doivent intégrer plus d'intelligence pour aider l'administrateur du procédé à améliorer son exécution.

Besoin d'avoir un modèle simple

Le modèle de procédé est un support basique pour la conscience de groupe[8]. Le modèle offre un langage pour discuter sur un procédé coopératif en cours d'exécution ; il permet à ses participants de communiquer entre eux sur ce modèle, sur les actions réalisées ou futures, sur

[8] en anglais, group awareness

les documents créés et ceux qui doivent être créés dans le futur : les modèles leur permettent de partager leurs connaissances sur le procédé coopératif auquel ils participent [Ago96]. Comme support de la conscience de groupe, le modèle doit être simple et intuitif.

L'acteur utilise le plan (le modèle) pour se situer dans le contexte du procédé coopératif : il peut agir en concordance avec lui, ou le modifier pendant l'exécution du procédé pour mieux résoudre sa tâche. L'administrateur l'utilise pour analyser les performances (à partir des données de l'historique), pour évaluer ses points faibles, pour concevoir les changements, pour communiquer les changements à ses acteurs. Pour que tous les utilisateurs puissent l'utiliser efficacement, son interprétation doit être simple et évidente.

Toutes ces fonctions du modèle impliquent un objectif de conception supplémentaire : l'utilisation d'un modèle de workflow de haut niveau d'abstraction qui doit être facile à utiliser et doit supporter la visualisation de ses éléments.

2.3 Principes directeurs

Dans cette section, nous présentons quelques principes directeurs qui nous ont guidés dans notre travail sur la flexibilité du workflow.

2.3.1 Procédés compétitifs et coopératifs : même description mais différentes interprétations mentales

L'idée fondamentale est venue de l'observation suivante. Concrètement, quand nous regardons les descriptions papier des procédés administratifs ou de production d'une part, et des procédés de co-conception ou co-ingénierie d'autre part, nous remarquons qu'elles sont représentées de la même manière, en utilisant les mêmes formalismes de base : rectangles, cercles, flèches annotés... Cependant, il est clair que, quand nous regardons un tel diagramme, nous ne l'interpréterons pas de la même manière dans le contexte d'un procédé administratif ou dans celui d'un procédé de co-conception. Par exemple, dans un contexte administratif, nous interprétons deux rectangles connectés par une flèche en tant qu'ordonnancement strict des deux tâches représentées par les deux rectangles ; dans notre esprit, la deuxième tâche ne peut pas commencer son exécution avant que la première ne se soit terminée. Cependant, quand nous considérons le même schéma dans le contexte d'un procédé de co-conception, notre interprétation est beaucoup plus flexible. Nous savons que ce schéma est une indication générale, mais que les tâches peuvent se superposer dans le temps, peuvent itérer. En fait, dans le cas des procédés administratifs ou de production, nous voyons les tâches comme des boîtes noires, s'exécutant séquentiellement et partageant leurs données seulement au début et à la fin de leur exécution. Dans le cas des processus créatifs, nous voyons les tâches comme des boîtes blanches, rendant visibles quelques résultats intermédiaires, se superposant et itérant leurs exécutions.

C'est ce que la figure 2.4 illustre. À partir du même modèle de procédé, et selon le contexte compétitif ou coopératif, l'esprit interprète le même schéma de manière différente : dans le cas des procédés coopératifs, les dépendances sont relâchées (dans certains cas, elles peuvent même être interprétées comme dépendances de type fin-fin).

Par exemple, si nous prenons « Choisir livre » pour A, « Choisir mode paiement » pour B et « Vérifier capacité paiement » pour C, qui est un procédé administratif typique, nous comprenons que ces trois activités s'exécuteront une après l'autre (voir figure 2.4, interprétation compétitive). Si l'on prend « Editer » pour A, « Revue » pour B et « Modifier » pour C, notre esprit infère la possibilité pour ces activités de s'exécuter de manière coopérative, c'est à dire que les exécutions

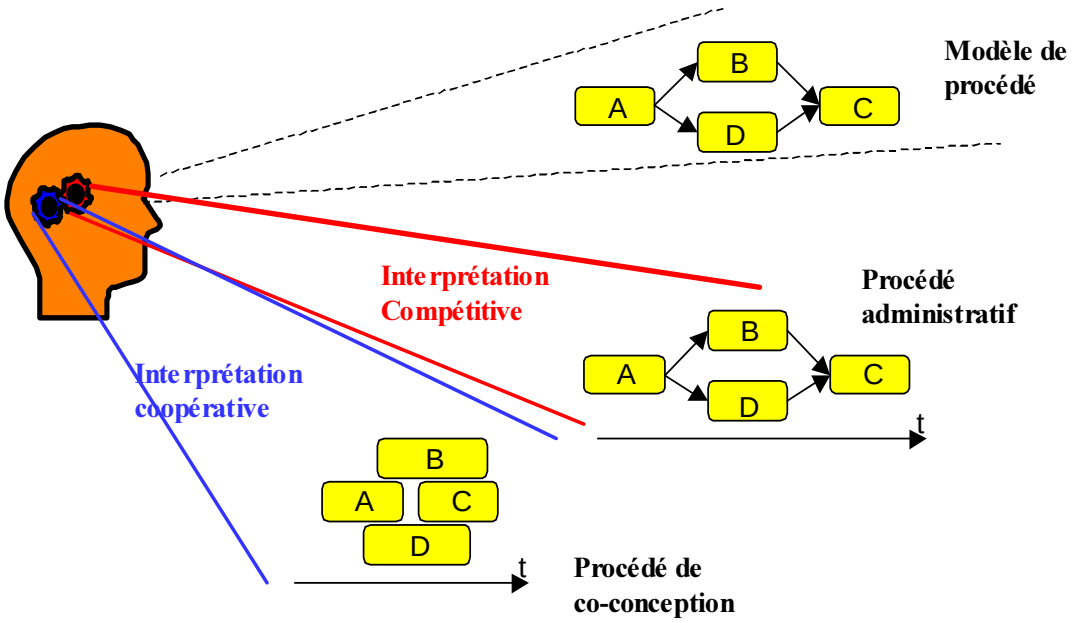


FIG. 2.4 – Traduction mentale différente de la même description graphique

des activités « Editer », « Revue » et « Modifier » peuvent se superposer et partager des versions successives du document (voir la figure 2.4, interprétation coopérative).

Partant de l'observation que les procédés compétitifs et coopératifs sont décrits à un haut niveau de conceptualisation de la même manière, en utilisant la même représentation graphique, nous considérons que la meilleure approche est de décrire les procédés coopératifs comme les procédés traditionnels, mais de changer la façon dont les modèles sont interprétés par le moteur de workflow, afin d'augmenter le parallélisme et les interactions entre activités.

2.3.2 Un modèle de coopération basé sur la gestion des résultats intermédiaires

Le panorama des travaux qui abordent la coopération montre la diversité des approches et leur divergence dans l'analyse de ce concept. Ceci est particulièrement dû à la complexité du phénomène de coopération et à la diversité des objectifs des travaux.

Notre modèle se base sur une certaine analyse d'usage, y compris notre propre expérience dans le développement de logiciel, et des discussions avec des concepteurs dans le domaine de la construction de bâtiments. Comme introduit dans la section précédente, le principe sur lequel notre vision de la coopération est établie est que les gens coopèrent en échangeant des brouillons, des croquis, des résultats intermédiaires afin d'augmenter l'efficacité du travail de l'équipe distribuée : anticipation du travail, feedback rapide, meilleure cohésion. Naturellement, des résultats intermédiaires sont progressivement élaborés et un objet de coopération existera en plusieurs versions avant que la valeur finale soit produite. Pour préserver l'autonomie des participants, l'initiative pour rendre un résultat intermédiaire visible est sous la responsabilité

des agents humains. Un espace de travail privé contenant des objets en modification est associé à chaque activité et une zone de travail commune est partagée entre les activités (cf. Figure 2.5). Pour rendre un résultat intermédiaire visible, un agent humain le transfère à partir de son espace de travail privé vers l'espace de travail partagé. Ce modèle est clairement inspiré du paradigme « Copy/Modify/Merge » [Fei91] largement répandu dans le domaine du génie logiciel.

Un besoin important d'un workflow coopératif est que les activités puissent partager des résultats intermédiaires pendant leur exécution.

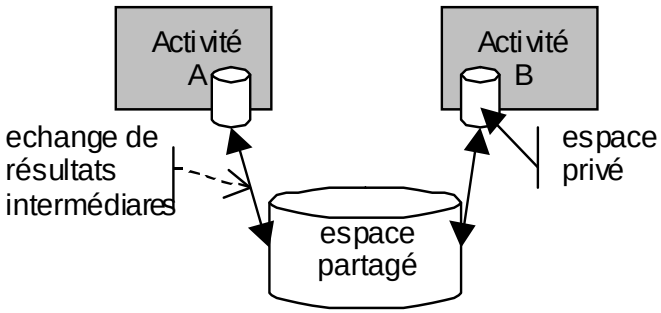


FIG. 2.5 – Coopération par échange des résultats intermédiaires

2.3.3 Composants logiciels pour l'implantation d'un workflow

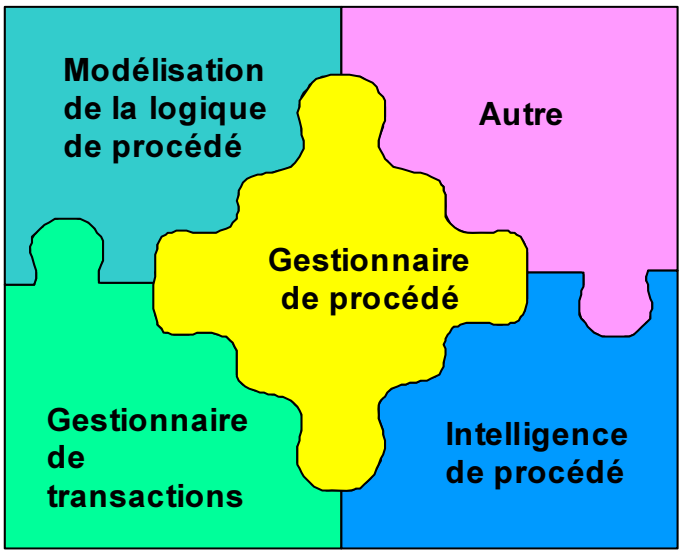


FIG. 2.6 – Développement d'une application en intégrant des composants

Dans notre vision, une application logicielle doit être l'assemblage de composants logiciels qui doivent parfaitement s'intégrer les uns aux autres, comme les morceaux d'un puzzle (voir la figure 2.6). En conséquence, nous pensons que, si aujourd'hui il est difficile de développer un logiciel en

assemblant simplement des composants logiciels, c'est parce que les composants logiciels ne sont pas bien définis et se superposent : les « bus logiciels » sont étendus pour intégrer des transactions, les logiciels de gestion de versions sont étendus pour intégrer la gestion des procédés, et, ce qui nous concerne principalement ici, les systèmes de workflow mélangent souvent pour des raisons pratiques les considérations de flot de contrôle et de flot de données.

Le schéma 2.6 dépeint une architecture approximative de notre système de gestion de workflow. Il précise quatre composants principaux : le système de modélisation de la logique de procédé, le gestionnaire d'activités, le gestionnaire de transactions et le composant d'intelligence de procédé. Le premier composant est un outil qui permet aux utilisateurs de définir les modèles de procédés. Le gestionnaire d'activités (le moteur de workflow) contrôle l'exécution des procédés en se basant sur leurs modèles. Il affecte les activités pour exécution aux participants en tenant compte de dépendances définies dans le modèle de procédé. Le gestionnaire de transactions assure la cohérence des échanges de données entre activités. Le composant d'intelligence de procédé comprend différents outils permettant de valoriser les connaissances sur les exécutions passées. En tenant compte de nos observations sur l'interprétation d'un modèle coopératif (section 2.3.1) nous remarquons que, si actuellement le composant de modélisation de la logique de procédé peut être réutilisé en tant que tel, les composants de gestion d'activités et de gestion de transactions doivent être clairement identifiés et spécifiés.

Un système modulaire facilite l'adoption d'un système de workflow. Bon nombre de conflits dans la mise en place d'une application de workflow peuvent résulter des problèmes d'intégration du système de gestion de workflow avec les outils et les pratiques de travail existants.

Un système de workflow doit avoir la capacité de s'intégrer facilement et avec flexibilité aux autres composants logiciels : gestionnaires de transactions, outils pour l'analyse et l'optimisation des procédés.

2.3.4 Les procédés compétitifs : un cas spécial des procédés coopératifs

Une dernière considération fondamentale est qu'un procédé traditionnel, administratif ou de production (on l'appelle « procédé compétitif ») est un procédé coopératif mais avec plusieurs contraintes supplémentaires qui limitent son exécution : intuitivement, un processus compétitif est un processus coopératif qui ne rend pas visible des résultats intermédiaires et dont les dépendances modélisées entre activités sont strictement respectées.

En d'autres termes, le modèle normal (ou canonique ou classique) de workflow devrait être un modèle coopératif de workflow, comme celui décrit dans cette thèse, et pas celui admis actuellement (typiquement modèle de WfMC [Coa95], [Coa99]), qui est une restriction d'un modèle de workflow coopératif.

2.4 Synthèse

Notre objectif est d'offrir un support informatique pour la coordination et la coopération des membres d'une équipe distribuée. Une coopération efficace se base sur l'échange d'informations (documents, données partagés). Pour améliorer l'efficacité de la coopération, les activités de l'équipe doivent être coordonnées. L'outil doit permettre de définir, exécuter et faire évoluer les procédés que l'équipe distribuée doit suivre pour atteindre son objectif.

Les systèmes de workflow ont comme objectif la coordination des procédés ; ils offrent des outils pour la description, la modélisation, l'analyse, l'exécution et la coordination des procédés. Cependant, ces systèmes sont trop rigides, ils ne permettent pas d'exécutions et d'échanges de données flexibles.

Notre objectif est donc de définir un modèle de workflow coopératif, aussi simple à utiliser que le modèle classique de workflow pour les procédés administratifs, mais adapté aux applications qui nous intéressent. Nous avons identifié les besoins suivants pour un workflow coopératif :

- exécution flexible des activités, permettant aux activités de démarrer plus tôt leur exécution par rapport aux conditions définies dans le flot de contrôle,
- parallélisme augmenté entre activités et procédés,
- échange de données flexible et contrôlé (échanges spontanés, publication des versions intermédiaires pendant l'exécution des activités),
- modèle simple, qui peut être modifié dynamiquement,
- le système de workflow doit inclure des outils intelligents pour analyser et améliorer le procédé à partir des exécutions passées.

Chapitre 3

État de l'art

3.1 Introduction

Plusieurs domaines de recherches comme le travail assisté par ordinateur, les systèmes de workflow et les environnements de développements de logiciel ont comme objectif d'offrir un support pour les procédés coopératifs. Ils suivent des paradigmes différents et satisfont seulement un sous-ensemble des besoins identifiés dans la problématique. Nous présentons les approches suivantes :

- les *environnements de développement centrés procédés*, conçus pour les procédés de développement logiciel qui sont fortement interactifs, semi-structurés et centrés données,
- les *systèmes de workflow* qui se concentrent sur la coordination des activités dans des procédés très structurés,
- les *modèles de transactions avancées* dont l'objectif est d'assurer la cohérence d'une base de données commune qui est accédée par des activités d'une application non-traditionelle.
- les *outils d'aide au travail coopératif* qui supportent la coopération des participants, sans offrir une coordination explicite.

Quoi que la tendance actuelle soit à la convergence des résultats de ces différents axes, nous présentons individuellement chaque approche, en mettant en évidence ses limites pour la mise en oeuvre des procédés coopératifs.

3.2 Environnements de développement de logiciel

Les procédés de développement de logiciel sont caractérisés, comme tout procédé d'ingénierie, par la complexité du produit et du procédé d'une part et par la dynamique de l'environnement dans lequel le procédé se déroule, d'autre part.

Les participants aux procédés ont besoin d'échanger des informations pendant leur travail d'une manière contrôlée ; ils intègrent et influencent les activités des autres utilisateurs en travaillant en parallèle sur les mêmes données.

Généralement, dans les environnements de développement de logiciel, l'échange et le partage d'informations sont réalisés en utilisant un référentiel ([Kai93], [Kai87], [Hon88]). Ce référentiel stocke tous les objets créés pendant le projet de développement, comme les sources, les binaires, la documentation de conception, et les tests. Dans [Cel96] un modèle de données versionnées pour le referenciel est présenté qui permet de représenter les différentes formes dans lesquelles les composants logiciels et les éléments du procédé de développement peuvent exister.

En général, dans les environnements de développement, chaque utilisateur a son *espace de*

travail privé dans lequel il peut modifier des objets. Ces objets doivent être d'abord lus à partir de l'espace partagé (check-out). Après la modification, les objets modifiés doivent être intégrés dans l'espace partagé (check-in). Ainsi, la coopération est définie par la capacité des utilisateurs à accéder aux et à produire des versions intermédiaires des objets partagés pendant leur travail. Évidemment, une synchronisation est nécessaire pour coordonner les accès aux objets partagés et pour maintenir la consistance du référentiel et des espaces privés. Si les participants ont le droit de travailler simultanément sur les mêmes objets dans leurs espaces privés, des mécanismes additionnels sont nécessaires pour combiner les versions développées en parallèle dans une seule version acceptée par les différents participants [Hon88].

Une sous-classe d'environnements de développement, appelée centré procédé [Kai93], fournit des mécanismes additionnels pour spécifier le procédé. Le procédé est défini comme un ordre partiel des activités d'ingénierie, des contraintes sur ces activités et les objets et les applications utilisés dans ces activités.

Nous présentons brièvement des approches qui appartiennent à cette classe d'environnements parce qu'elles intègrent à la fois la gestion de données et la gestion du procédé.

Les briques conceptuelles d'un environnement de développement [Joe97b] sont l'espace de procédé, l'espace référentiel et l'environnement de travail (figure 3.1). L'espace de procédé définit les tâches à faire et la façon dont elle doivent être exécutées, en spécifiant un ordre partiel des étapes du procédé. Il spécifie le comportement de l'exécution dynamique (états d'exécutions, flot de contrôle et de données) des instances de procédé. L'espace référentiel consiste en l'espace produit et en l'espace version et il fournit les procédures de base pour définir, identifier et accéder aux versions, aux variantes et aux configurations. L'espace produit spécifie la structure du produit logiciel sans prendre en compte les versions (la composition et ses dépendances). L'espace version définit comment les versions sont représentées et gérées. L'environnement de travail est divisé en espace de travail (qui contient tous les documents, incluant les résultats intermédiaires dont un acteur a besoin pour son travail) et l'espace activité (qui contient la liste de tâches et fournit le support pour l'exécution en assurant l'automatisation, le contrôle et la conscience de groupe).

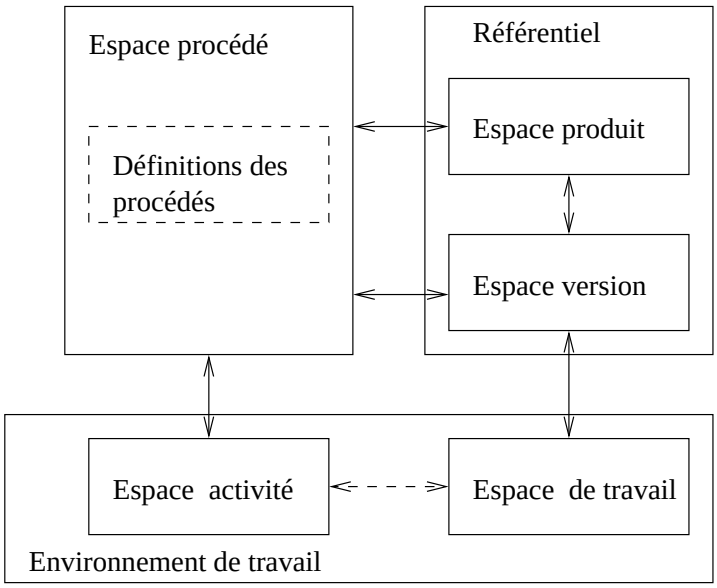


FIG. 3.1 – Éléments conceptuels d'un environnement de développement ([Joe97b])

Nous présentons une classification de ces environnements basée sur la manière d'intégrer les fonctionnalités procédés et les fonctionnalités de gestion des données. Nous mettons en évidence leurs points forts et faibles. La plupart proviennent des systèmes de gestion de configuration, enrichis par des fonctionnalités orientées procédé.

Nous distinguons les approches centrées activité et les approches centrées document. De plus, nous divisons les approches centrées activités en approches orientées flot de données et orientées changement. Les approches centrées document peuvent être centrées état du document ou objet.

3.2.1 Approches centrées activité

1. *Les approches orientées flot de données* se reposent sur la description des activités et du flot de données entre elles. Elles sont caractérisées par une séparation stricte entre procédés et données. Dans le modèle de procédé, le flot de données est représenté par des éléments de données qui sont consommés/produits par des activités (figure 3.2). Ces modèles ne font pas de supposition sur la représentation des données. Ainsi n'importe quel modèle d'objet (versionné ou non) peut être intégré. Les procédés peuvent être supportés à un haut niveau d'abstraction. Cependant, l'environnement de travail et le support pour la coopération sont assez pauvres.

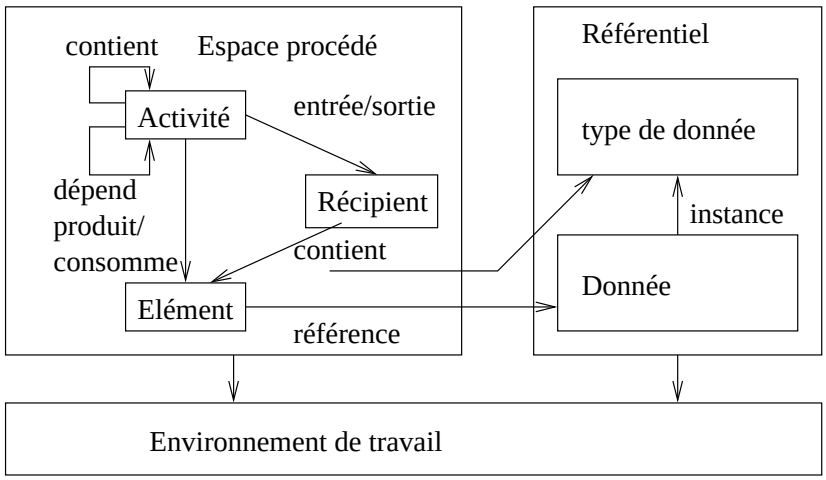


FIG. 3.2 – Approches orientées flot de données

Des exemples de ces approches sont les procédés basés sur les réseaux de Petri (SPADE [Ban93a]) ou centrées activité (DYNAMITE [Hei96]). DYNAMITE est conçu pour gérer la nature dynamique des procédés de développement de logiciel et fournir des facilités pour supporter les procédés de gestion des modifications. Il fournit des mécanismes spécifiques pour gérer le *feedback* et il supporte la coopération au niveau du procédé par le passage des résultats intermédiaires entre activités concurrentes (voir la figure 3.3.2). Les activités et les données impliquées dans le feedback sont versionnées dans le modèle de procédé assurant la tracabilité et la restauration de l'environnement de travail.

2. Dans *les approches orientées changement*, les activités, les transactions et les changements sont fortement liés entre eux et sont la base pour l'intégration du procédé et du modèle de version (figure 3.3). Une activité est réalisée dans une transaction, généralement humaine, dans l'environnement de travail. La transaction conduit à un changement des données qui est lié à une activité. Le flot d'information entre les environnements de travail n'est pas

représenté au niveau du procédé. Cette approche est basée sur un support pour un travail coopératif contrôlé entre activités qui détermine le contenu des environnements de travail respectifs. Tandis que les activités définissent le contexte, le système de gestion de versions, qui est à la base de ces systèmes, est chargé de fournir des fonctionnalités pour le contrôle des versions et de l'espace de travail et pour supporter le travail coopératif.

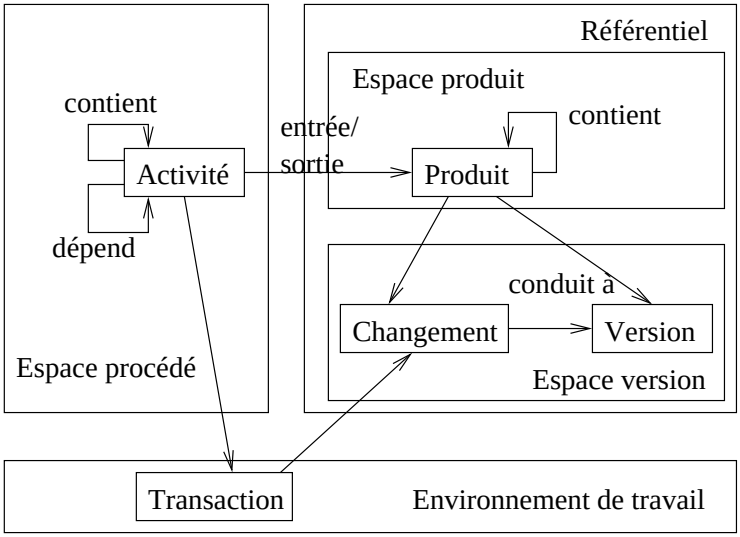


FIG. 3.3 – Approches orientées changement

Évidemment, cette approche se combine très bien avec le modèle de versions orienté changement et permet ainsi une bonne intégration du formalisme du procédé et des versions. Cependant, les systèmes qui suivent cette direction supportent l'intégration de la gestion des données et du procédé au niveau technique tandis que l'intégration est moins bien supportée au niveau conceptuel (EPOS [Con91], COO [God96], ClearGuide [Leb96]) ou ont des capacités très limitées pour la gestion des procédés. (Asgard [Mic96]).

3.2.2 Approches centrées produit

1. L'*approche centrée état du produit* se concentre principalement sur le contrôle du cycle de vie du produit et assure des fonctionnalités de bas niveau pour la gestion des procédés basée sur des transitions d'états et des mécanismes d'événements. En général, il n'y a pas une notion (de haut niveau) d'activité. Ainsi, l'environnement de travail consiste seulement en l'espace privé géré par le système de gestion de versions. Les mécanismes à base d'événements peuvent supporter les fonctionnalités de bas niveau, mais ils ne sont pas adaptés pour être utilisés comme un formalisme général de modélisation de procédé parce qu'ils sont difficiles à comprendre par les utilisateurs humains. Ils manquent les concepts de haut niveau pour la spécification des procédés et l'exécution est difficile à contrôler.

   La plupart des systèmes de gestion de versions se retrouvent dans cette classe. Par exemple, ClearCase [Leb94] fournit un ensemble d'outils de contrôle de procédés qui supporte le contrôle d'accès, l'enregistrement des événements et des politiques dynamiques basées sur des événements. Cependant, ClearCase n'offre pas de possibilités de modélisation et d'exécution des procédés.

   ADELE-TEMPO [Bel94], est un gestionnaire de versions qui fournit des fonctionnalités

Besoin	EPOS	Adele/Tempo	APEL	ESCAPE
relation entrée/sortie	réseau d'activités et de données	implicite	paramètres in/out + dépendances de données	dérivé de la relation entre objets
Contrôle de la concurrence	pas intégré	implicite	oui	implicite
Coopération contrôlée	transactions coopératives	collaboration des rôles	politiques coopération variées	schémas de coopération
Coopération libre	superposition des espaces de travail	espaces publiques	espaces publiques	non

TAB. 3.1 – Comparaison

basées sur un mécanisme d'événements. Le formalisme de procédé TEMPO est basé sur le concept de rôle. Un modèle de procédé est défini comme un ensemble d'objets ayant un rôle. Ainsi, ce formalisme est de nouveau centré produit. Il supporte l'affectation des activités aux acteurs mais le flot de contrôle des procédés n'est pas modélisé. Par conséquent, il n'offre pas un support complet pour l'exécution des procédés.

Une autre approche pour l'intégration d'un support de gestion de procédés de haut niveau dans les gestionnaires de configuration et en particulier dans le système Adele est APEL [Est97]. Cette approche se base principalement sur les activités (approche orientée changement), mais intègre aussi des aspects basés sur les états des produits. Il fournit un support complet intégrant des fonctionnalités pour la gestion des procédés et pour la coopération. Malheureusement, l'exécution n'est pas assurée à ce haut niveau d'abstraction, mais nécessite la compilation dans le langage ADELE, c.a.d. dans des règles basées sur des événements.

2. *Approches objets* Ces approches adaptent les techniques de modélisation objets et les appliquent à la gestion des configurations et des procédés. Dans cette approche, les objets logiciel sont les briques de construction du modèle. Ils définissent la structure de l'espace produit, organisent leur propre versionnement et spécifient des opérations qui peuvent leur être appliquées. Le comportement d'exécution est spécifié par un diagramme d'états qui définit la coordination des opérations de l'objet.

   Dans cette approche, les aspects procédés et gestionnaire de configuration sont intégrés à un haut niveau d'abstraction et les aspects procédés et données sont mieux intégrés. Cependant, une approche basée exclusivement sur des objets a des limitations comme langage général de modélisation de procédés. Les points faibles sont la subordination intrinsèque des activités aux objets logiciel ou documents et la représentation implicite du flot d'informations.

   Une amélioration de cette approche qui définit et intègre des différentes classes d'objets pour les procédés, les artefacts et leurs versions dont le comportement est coordonné par des mécanismes de passage de messages et des événements parait être prometteuse. ESCAPE+ [Neu96] et SOCCA [Eng94] suivent cette direction. SOCCA ne prend pas le versionnement en considération, tandis que ESCAPE+ étend le système précédant avec des concepts de versions.

3.2.3 Conclusion

Les environnements de développement centrés procédé intègrent des fonctionnalités pour la gestion de procédés et la gestion des données. Deux approches principales ont été présentées pour l'intégration de la gestion des procédés et des données : une approche centrée activité et une approche centrée produit. Les environnements de la première catégorie, centrés activités ou basés sur les réseaux de Petri, offrent un bon support pour la modélisation des procédés, mais ils sont moins utiles pour la gestion des produits logiciel.

La plupart des environnements de la deuxième approche se base sur des systèmes de gestion de versions qui intègrent des mécanismes de bas niveau pour le contrôle de procédé, comme des règles et des événements.

Le tableau 3.1 synthétise les fonctionnalités des approches les plus prometteuses par rapport aux besoins suivants :

- relation entrée/sortie - la relation formelle entre les données en entrées/sorties des procédés (activités) et les produits logiciel,
- contrôle de la concurrence - les mécanismes pour supporter la coordination des activités qui travaillent de manière concurrente sur les mêmes versions,
- support pour la coopération contrôlée - notification des changements et propagation des résultats intermédiaires,
- support pour la coopération libre - l'interaction directe entre les participants au procédé qui n'est pas complètement contrôlée par le moteur de procédé.

Les modèles d'environnements de développement de logiciel ont influencé des travaux plus récents sur les systèmes de workflow et les collecticiels dont le champ d'application est plus large.

3.3 Systèmes de Workflow

Les systèmes de workflow facilitent la description, la modélisation, l'analyse, l'exécution et la coordination des procédés.

Une *définition de procédé workflow* (ou plus simplement un *modèle de procédé*) est la représentation formelle d'un procédé. Il spécifie principalement les activités et leurs interdépendances (le flot de contrôle et de données), les ressources et les programmes informatiques associés à chaque activité.

Un *système de gestion de workflow (WfMS)* est un système informatique qui transforme la représentation explicite d'un modèle de workflow dans un format interne et exécutable et fournit un environnement opérationnel pour l'exécution, l'administration et la surveillance des procédés.

L'approche workflow présente quelques avantages :

- elle sépare clairement la sémantique de l'exécution d'une activité de la logique du procédé (flot de contrôle et de données) ;
- le modèle de procédé est indépendant des modifications du schéma de personnel de l'organisation, parce qu'il associe des rôles (classes de personnes groupées par compétences ou par unité organisationnelle) à une activité ;
- l'efficacité du procédé est améliorée parce que le système fournit l'information appropriée et la procédure appropriée pour l'utiliser au juste moment. Le modèle décrit qui (les acteurs), quoi (l'activité a exécuter) et comment (les détails informatiques pour l'activité a exécuter) le procédé doit être exécuté.

Cependant, les systèmes de workflow ont été critiqués comme étant trop restrictifs. Souvent, un programme workflow est trop contraignant pour le participant humain : les contraintes de coordination imposées empêchent celui-ci de réaliser son travail de la manière qui lui est la plus

familière ou la plus simple. Une autre limite des technologies actuelles est le fait qu'elles ne prennent pas en compte la nature dynamique des environnements et des procédures de travail. Pour répondre à ces problèmes, bon nombre de projets de recherche ont proposé des modèles et systèmes de workflow plus flexibles [Kle98].

Une vue d'ensemble et une taxonomie pour la flexibilité sont fournies dans [Han98], [Hei99].

Notre classification des approches existantes se base sur l'utilisation du modèle de procédé. Nous distinguons trois approches principales :

- considérer le modèle de procédé comme ressource pour l'action [Suc87],
- autoriser la modification dynamique du modèle de procédé courant,
- supporter un degré de liberté plus élevé dans l'exécution en utilisant des modèles moins prescriptifs.

3.3.1 Modèle - ressource pour l'action

Cette direction de recherche se base sur la proposition de Lucy Suchmann [Suc87] qui affirme que « les plans (ici, le plan est considéré comme une sorte de modèle de procédé qui est intéressant par rapport aux pratiques de travail) sont des ressources pour l'action » ; elle indique que les plans peuvent jouer un rôle qui est différent de celui prescriptif. « La fonction des représentations abstraites n'est pas de servir de spécification pour les interactions locales, mais de nous orienter d'une manière qui nous permette d'exploiter certains événements inattendus de notre environnement et d'en éviter d'autres ... »[Suc87].

Nous presentons deux modèles basés sur cette approche.

MILANO [Ago96] analyse les modèles de procédés (plans) comme des artefacts qui supportent la conscience du groupe de participants et pas comme des prescriptions qui contraignent leur comportement. Les caractéristiques d'un tel modèle sont :

- la simplicité du plan ; le formalisme utilisé doit supporter les changements et faciliter la lecture du plan.
- un ensemble minimal d'informations requises pour générer un plan qui abstrait les étapes principales pour atteindre l'objectif. Des représentations multiples du même plan sont utilisées pour des utilisateurs différents (le responsable du procédé et l'utilisateur peuvent consulter deux vues différentes du même procédé).
- un calcul automatique des chemins exceptionnels dans un plan. Pour résoudre des situations exceptionnelles, l'utilisateur peut sauter des activités en avant ou en arrière et le système calcule automatiquement les états qui peuvent être atteints.
- des changements du plan cohérent avec une spécification désirée. Si pour chaque plan, une spécification critique minimale est donnée, le système vérifie si le changement proposé est correct par rapport à cette spécification. Les changements supportés sont : paralléliser des activités séquentielles, changer l'ordre des deux activités, et mettre en séquence deux activités qui auparavant étaient parallèles.
- une implantation sûre d'une modification du plan dans les instances en cours. Le formalisme théorique permet de vérifier si une modification peut être appliquée à une instance, en fonction de l'état courant (l'exécution de l'instance doit continuer correctement selon le nouveau modèle).

Le modèle proposé, basé sur des réseaux de Petri, est simple : il ne contient pas de boucles et il représente seulement le flot de contrôle. Le formalisme théorique est utilisé pour obtenir automatiquement des représentations différentes du modèle et pour vérifier la correction des changements par rapport à la spécification minimale.

Le modèle ICN (Information Control Net) étendu Dans [Nut96],[Ell96] l'auteur argumente qu'un modèle et un système de workflow flexible doivent permettre l'utilisation du procédé comme ressource pour l'action ou comme outil de coordination, selon les conditions de l'application. Un domaine-espace tridimensionnel (figure 3.4) est défini pour un modèle de workflow. Les trois axes sont :

- le degré de détail de la description du procédé,
- la conformité exigée par l'organisation et
- le degré d'abstraction opérationnelle.

Le modèle, basé sur les réseaux de contrôle d'information (ICN, Information Control Net), permet de représenter des parties du procédé à un niveau abstrait aussi bien qu'à un niveau très détaillé ; les représentations détaillées correspondent à des implantations exécutables. Un modèle peut comprendre des parties du procédé qui doivent être exécutées comme prescrites et des parties qui peuvent être exécutées en suivant seulement des recommandations. La troisième dimension de l'espace permet des représentations qui sont plus ou moins déclaratives ; les spécifications des objectifs et des intentions sont considérées comme déclaratives et un programme C comme opérationnel.

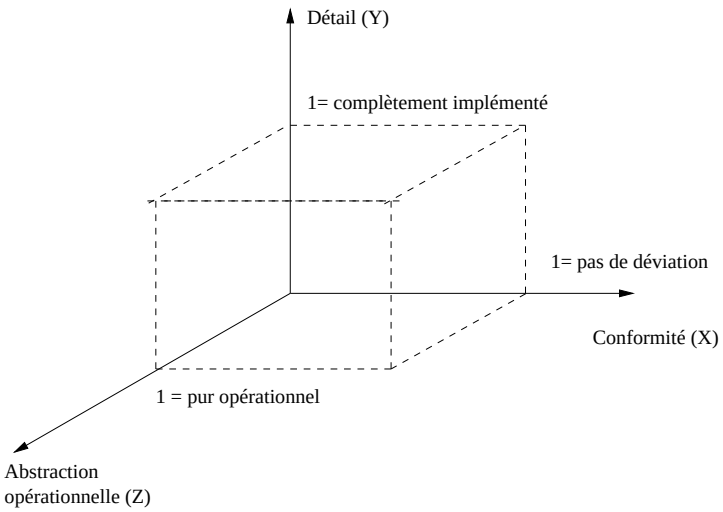


FIG. 3.4 – L'espace domaine du modèle

Le modèle ICN est étendu pour couvrir tout le domaine défini de cette manière. Un ICN est un graphe de précédence qui représente les activités dans le procédé. Chaque noeud du graphe appartient à une région dont le type est défini informellement par un point dans l'espace de domaine.

Afin de représenter les différents points dans ce domaine, le modèle de workflow est enrichi avec des activités et des régions de type objectif. Un noeud de type objectif contient la spécification pour une partie du procédé qui contient du travail non-structuré. Les détails de cette étape de travail sont décrits seulement dans les termes des objectifs, des intentions ou des directives. Pour supporter les noeuds de type objectif, le système d'exécution doit inclure un environnement virtuel et fournir de l'aide contextuelle. L'environnement virtuel supporte les modèles sociaux existants dans le groupe en offrant un support pour la communication non-structurée.

Conclusion Le modèle MILANO est simple, mais il représente seulement le flot de contrôle. Le deuxième modèle presenté supporte les parties de procédé qui ne sont pas structurées en offrant aux utilisateurs seulement des descriptions des objectifs à atteindre, une aide contextuelle et un environnement virtuel. La coordination est laissée à la responsabilité des utilisateurs. Le support pour l'échange de données flexibles n'est pas pris en compte dans ce modèle.

3.3.2 Flexibilité incluse dans le modèle

Une deuxième approche propose des spécifications de workflow moins rigides, plus expressives qui permettent plus de flexibilité aux participants. Le modèle de workflow est étendu pour capturer des dépendances et des pratiques de travail qui ne peuvent pas être modélisées facilement avec les éléments des modèles de workflow classiques.

Modélisation à haut niveau vs. modélisation à bas niveau pour définir des workflows flexibles

Nous distinguons deux approches pour la modélisation des workflow : des langages de modélisation graphiques, faciles à utiliser, à un haut niveau d'abstraction vs. des formalismes de spécification à un bas niveau d'abstraction qui sont orientés systèmes ou exécution (figure 3.5).

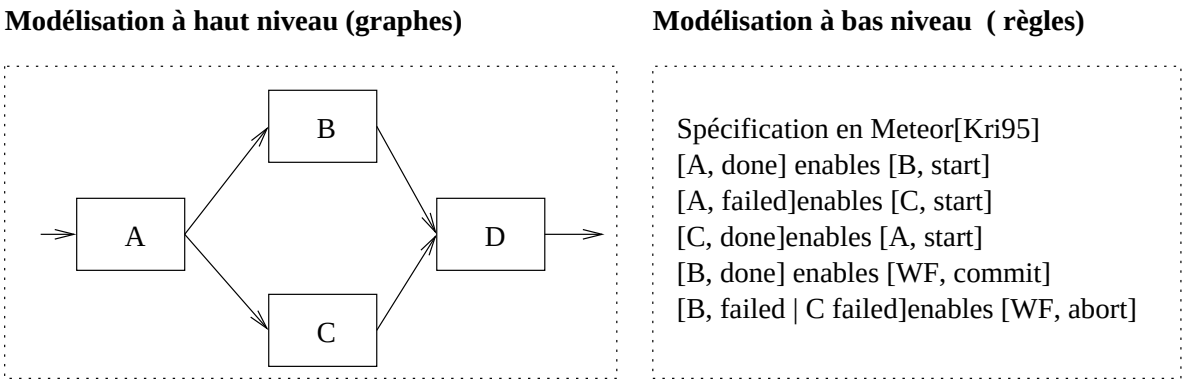


FIG. 3.5 – Modélisation à haut niveau vs. modélisation à bas niveau

L'approche de modélisation à haut niveau est basée sur des concepts de graphe ou de réseaux et permet de définir des workflows en dessinant des boîtes et des arcs (IBM Flowmark [Ley00], WFMC [Coa99], WASA [Wes98, Ell91], ADEPT [Rei98], Workflow Nets[VdA98]). Une activité est atomique et elle est vue comme un bloc monolithique qui consomme toutes ses informations d'entrée quand elle est démarrée et produit toutes les sorties quand elle se termine. Des diagrammes de transition d'états différents mais fixes, sont définis pour différents types d'activités : manuelles, automatiques, activités complexes. Les dépendances de flot de contrôle sont définies entre les activités. Elles établissent une dépendance de type fin-démarrage (une activité peut démarrer seulement quand l'activité précédente a terminé).

Les avantages de cette approche résident dans la modélisation à un haut niveau d'abstraction, dans la visualisation graphique des workflows, et dans un bon support pour l'analyse des workflows. Ceci sont des besoins fondamentaux pour faciliter l'intervention humaine dans le cas des modifications dynamiques. De l'autre coté, ces modèles ne peuvent pas supporter des flots de contrôle flexibles, ni d'échanges de données flexibles, en particulier entre des activités en cours d'exécution. La puissance d'expressivité est limitée.

L'approche de spécification de bas niveau a des caractéristiques différentes. La plupart des modèles se basent sur le formalisme des règles réactives et peuvent exprimer et contrôler des dépendances de grain fin entre activités, plus précisément des dépendances entre états des activités. Ceci permet de capturer le comportement des différents types de procédés, de définir des workflows moins restrictifs. Les exemples les plus proéminents de ce type sont $METEOR$ [Kri95] et $METEOR_2$ qui se focalisent sur l'exécution décentralisée des activités et sur les aspects systèmes.

Bien que l'exemple présenté dans la problématique pourrait être exprimé dans le formalisme de spécification des workflows de bas niveau basé sur des règles, le modèle résultant (défini comme un ensemble de règles) serait difficile à comprendre par les utilisateurs et donc inadéquate pour l'intervention humaine et les changements dynamiques. Il impliquerait aussi de prévoir et modéliser toutes les interactions possibles entre les participants. La figure 3.5 présente deux exemples de spécification de workflow pour illustrer les deux approches de modélisation.

Nous présentons quelques approches qui proposent des modèles flexibles et qui sont en même temps des spécifications de haut niveau.

MOBILE Le concept principal dans MOBILE[Jab96] est la modélisation descriptive qui est réalisée en définissant plusieurs perspectives d'un modèle de workflow, indépendantes l'une de l'autre. Les perspectives les plus importantes sont : la perspective fonctionnelle (spécifie la décomposition des workflows), la perspective comportementale (spécifie le flot de contrôle), la perspective informationnelle (spécifie les données et le flot de données), la perspective organisationnelle (spécifie l'affectation des acteurs au procédé) et la perspective opérationnelle (l'intégration des applications).

L'omission de certains aspects dans la définition d'une perspective réduit le nombre de restrictions sur l'exécution du workflow et augmente le nombre de chemins d'exécution permis. Par exemple, quand l'ordre d'exécution des activités ne doit pas être prescrit, la définition du flot de contrôle entre elles peut être omise. Comme résultat, l'exécution du workflow sera valide pour n'importe quelle séquence de ces activités qui ne contredit pas les autres perspectives. Ainsi, le nombre d'exécutions possibles sera plus large.

Dans [Jab94], un autre exemple de modélisation descriptive est présenté comme une technique pour une modélisation compacte. Les auteurs proposent de nouveaux éléments de modèles simples afin de mieux représenter les configurations de travail utilisées dans la pratique, à la place d'une composition des constructions élémentaires.

Un scénario de date-limite est présenté. La sémantique de « B est la date-limite de A » est défini de la manière suivante : une activité A peut être répétée plusieurs fois tant qu'une activité B n'a pas encore été exécutée ou sautée. Par exemple, A peut être l'étape de soumission pour une composition. Un groupe de personnes peut envoyer quelques compositions à leur chef. Le chef peut soit démarrer une étape finale B de collection pour rassembler toutes les compositions soumises ou il peut sauter cette étape et de cette manière annuler toute la procédure de soumission de la composition. En utilisant des concepts de modélisation simples comme dans la plus part des systèmes de workflow, le modèle résultant sera plus compliqué et difficile à lire. (voir la figure 3.6).

COO De manière similaire, dans [God99], de nouveaux opérateurs sont introduits pour permettre la modélisation des interactions imbriquées entre activités qui s'échangent des versions intermédiaires.

Un exemple de procédé dans le télé-enseignement est présenté où les étudiants résolvent des

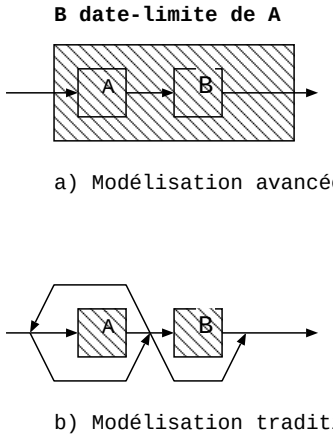


FIG. 3.6 – Modélisation d'une dépendance de type date-limite

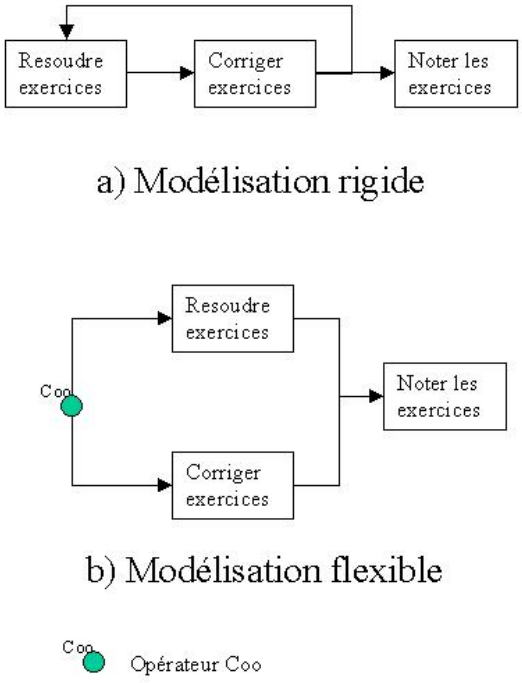


FIG. 3.7 – Dépendance rédacteur/relecteur

exercices proposés par le professeur. Ils peuvent transmettre au professeur quelques résultats partiels (sous-ensemble des exercices, des exercices incomplets) afin d'obtenir des commentaires, des suggestions, des corrections de la part du professeur. Ce processus peut itérer plusieurs fois. Cependant, comme le professeur doit partager son temps entre différents étudiants et d'autres activités, il n'est pas obligé de corriger chaque résultat intermédiaire, mais il peut le faire s'il veut et s'il a le temps. Il doit seulement corriger la version finale des exercices pour chaque étudiant avant une date-limite (qui peut être spécifique à chaque étudiant).

Ce processus peut être modélisé avec un modèle classique comme dans la figure 3.7-a. Cependant, ce modèle implique que le professeur corrige chaque résultat intermédiaire. Il est mieux modélisé en utilisant l'opérateur binaire COO qui permet à deux activités parallèles de s'échanger des résultats intermédiaires pendant leur exécution. Des paramètres supplémentaires permettent la modélisation de trois patrons de travail coopératif :

- *producteur/consommateur* Deux activités suivent un patron *producteur/consommateur* si la première a la responsabilité de fournir un artefact et la deuxième le lit pour l'intégrer dans son propre travail. Le consommateur doit lire la version finale du producteur.
- *écriture coopérative* Deux activités suivent un patron d'*écriture coopérative* si elles développent deux versions complémentaires du même artefact et elles doivent fournir un seul artefact avant de conclure. Elles peuvent rendre visibles quelques résultats intermédiaires avant de finir.
- *rédacteur/relecteur* Deux activités suivent un patron *rédacteur/relecteur* si la première produit un artefact et la deuxième lit une ou plusieurs versions de cet artefact pour faire la revue. Les rapports deviennent aussi des artefacts partagés. Le procédé de télé-enseignement illustre ce patron et sa modélisation en utilisant l'opérateur COO correspondant est présentée dans la figure 3.7-b).

L'approche CMI (Collaboration Management Infrastructure) définit un type spécial de dépendance, la « dépendance optionnelle répétitive » (figure 3.8) pour les activités pour lesquelles il n'est pas possible de savoir d'avance le moment où elles seront lancées et le nombre d'exécutions nécessaires. A partir du moment où une activité av_{opt} est autorisée par le flot de contrôle, elle peut être instantiée zéro ou plusieurs fois. Le moment et le nombre d'instances créées sont déterminées par l'acteur en charge. La primitive de dépendance répétitive optionnelle contient une partie optionnelle qui peut limiter l'intervalle de temps dans lequel l'activité av_{opt} peut être instanciée par le démarrage d'une autre activité (par exemple av_{term}, activité de fin). Dès que av_{term} est démarrée, des instances supplémentaires de l'activité av_{opt} ne peuvent plus être démarrées. Ainsi, dans l'intervalle entre la fin de l'activité s et le démarrage de l'activité av_{term}, les acteurs en charge de l'activité av_{opt} peuvent choisir d'exécuter plusieurs instances de cette activité, ou aucune.

L'activité av_{opt} ne peut pas avoir des connecteurs de données ou de contrôle sortants.

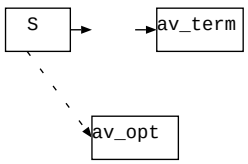


FIG. 3.8 – Dépendance répétitive optionnelle

[Geo00] décrit les facilités de l'infrastructure CMI pour supporter les procédés collaboratifs

de résolution des situations de crise :

- l'extension et le raffinement dynamique, mais contrôlé des procédés. Ceci peut être fait dans des points spécifiques du procédé qui sont prédéfinis dans le modèle.
- conscience de groupe et fenêtres d'opportunité. Les fenêtres d'opportunité sont supportées par les dépendances répétitives optionnelles mentionnées ci-dessus. Les informations qui doivent être envoyées aux utilisateurs peuvent être spécifiées par le concepteur du procédé (spécifications de conscience de groupe).
- la délégation des activités réalisée par des opérations de délégation et des primitives avancées d'attribution des rôles. Par exemple, une activité peut être attribuée a m participants. Au moment de l'exécution m instances de l'activité seront créées, chacune attribuée à un acteur différent.

Des exemples de mécanismes pour supporter la flexibilité des procédés collaboratifs sont les ressources de contexte et les activités de type « place[9] ». Les ressources de contexte sont des collections de ressources nommées. Elles servent à fournir un espace d'information partagé (entre deux activités qui peuvent appartenir à des procédés différents) et un domaine pour des rôles créés dynamiquement. Ces rôles sont accessibles seulement aux activités qui ont accès à la ressource de contexte qui les contient.

Les activités de type « place » permettent d'inclure dans le modèle des activités qui sont inconnues ou laissées ouvertes intentionnellement au moment de la spécification. Au moment de l'exécution, elles sont remplacées avec des modèles d'activités concrèts en utilisant une politique de résolution spécifiée. Cette politique spécifie comment sélectionner l'activité parmi les modèles existants ou comment la construire. La politique de construction et les activités de type « place » sont à la base de la modélisation tardive dans CMM.

DYNAMITE L'approche de DYNAMITE [Joe99] pour des modèles flexibles se base sur la définition et l'adaptation du comportement de l'activité.

La définition d'une activité décrit le comportement externe, indépendant du contexte (par exemple transactionnel ou non-transactionnel). Le comportement dépendant du contexte est défini par l'inclusion de l'activité dans une définition de procédé.

Le comportement de l'activité indépendant de contexte est donné en définissant un diagramme de transition d'états. Il définit les états (actif, suspendu, etc.) et les opérations qui peuvent être invoquées dans chaque état. Une règle ECA (événement-condition-action) peut être définie pour chaque transition. La définition d'une activité peut être héritée d'une définition abstraite pour définir des classes de comportement.

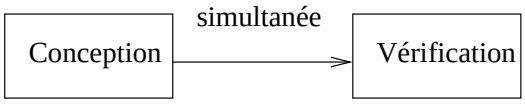
Le comportement dépendant du contexte est donné par les dépendances de flot de contrôle et les relations de groupe.

Des dépendances de contrôle arbitraires peuvent être spécifiées ou adaptées par le concepteur de workflow en spécifiant un ensemble de règles ECA. Quelques exemples de types de dépendances qui peuvent être réalisées avec ce concept sont ([Joear]) :

- la dépendance *simultanée* permet l'échange de données entre les activités actives simultanément, en assurant que la tâche dépendante ne se termine pas avant la tâche précédente. La définition de ce type de dépendance est illustrée dans la figure 3.9.
- une relation de groupe de type *exclusive* force ses membres à s'exécuter en exclusion mutuelle. En utilisant des branches parallèles, ceci permet de définir que certaines activités doivent être exécutées de manière séquentielle, sans définir leur ordre.

[9] placeholder, en anglais

– La dépendance de type *essai* supporte le branchement parallèle avec sélection finale. Quelques chemins alternatifs sont activés, mais quand la première branche atteint le point de synchronisation, les autres alternatives sont arrêtées et compensées (s'il est nécessaire et possible).



CONTROL FLOW DEPENDENCY simultaneous
For enable of target DEFINE:
ON finished BY predecessor simultaneous
started by predecessor simultaneous
IF (source.state=done OR source.state=active)

FOR finish OF target DEFINE:
IF source.state=done

[...le rest est similaire à la dépendance standarde de flot de contrôle]

FIG. 3.9 – Dépendance de type simultané

Les règles actives sont aussi utilisées pour modéliser les opérations de changement dynamiques [Hei96]. Chaque primitive de modification contient une pré-condition qui limite son application et génère un événement. Cet événement est traité par les instances affectées afin d'assurer la consistance comportementale des états d'exécution. Ainsi, une opération de changement peut être traitée de manière conceptuelle comme un changement d'état.

Pour supporter les workflows collaboratifs, la sémantique du flot de données est étendue en intégrant les versions dans le modèle de workflow au niveau conceptuel ([Joe00], [Joe97b]). Le flot de données peut être utilisé dans la spécification du flot de contrôle : la disponibilité des données peut être vérifiée et les activités réagissent à des événements liés à la production des données. Le flot de données est versionné et l'échange de résultats intermédiaires entre activités est permis en utilisant la dépendance de type *simultanée*.

Conclusion Les approches présentées introduisent de nouveaux éléments de modélisation (en général, de nouveaux types de dépendances de contrôle) pour représenter différentes situations pratiques, qui seraient autrement difficiles ou impossibles à définir avec les modèles classiques.

L'incovénient de ce type de flexibilité est le fait qu'elle doit être anticipée ou incluse dans la spécification du workflow. Dans certaines applications, cette flexibilité peut ne pas être suffisante et encore plus de flexibilité peut être requise. Dans les procédés coopératifs, fortement interactifs, il est difficile de prévoir et de limiter l'utilisateur à certains patrons de travail.

3.3.3 Modifications dynamiques

Pour supporter la nature dynamique des procédés et pour gérer les situations exceptionnelles, les systèmes de workflow devraient permettre les modifications des procédés en cours d'exécution.

Des exemples d'approches qui fournissent ce type de flexibilité sont les projets de recherche ADEPTflex[Rei98], Chautauqua [Ell97], WASA [Wes96] et WIDE [Cas96],[Cas98]. Ils se basent sur un modèle défini de manière traditionnelle et statique et fournissent des primitives explicites pour changer dynamiquement les instances d'un procédé courant. Ces primitives permettent d'ajouter/supprimer des tâches et de modifier le flot de contrôle et de données dans une instance

courante de procédé ; des contraintes sont imposées sur les modifications afin de garantir la correction syntaxique de l'instance du procédé résultant.

Nous donnons quelques exemples de ces opérations de modification, pour illustrer la variété des formes d'une seule opération. Par exemple, l'insertion d'une activité (figure 3.10) peut être simplement réalisée en ajoutant une activité entre deux activités dans l'ordre séquentiel. Ou ce peut être l'ajout d'une activité concurrente, ajoutant ainsi de nouveaux chemins dans le workflow. De manière similaire, une nouvelle activité peut être exécutée seulement dans certaines conditions, impliquant ainsi l'ajout des nouvelles conditions.

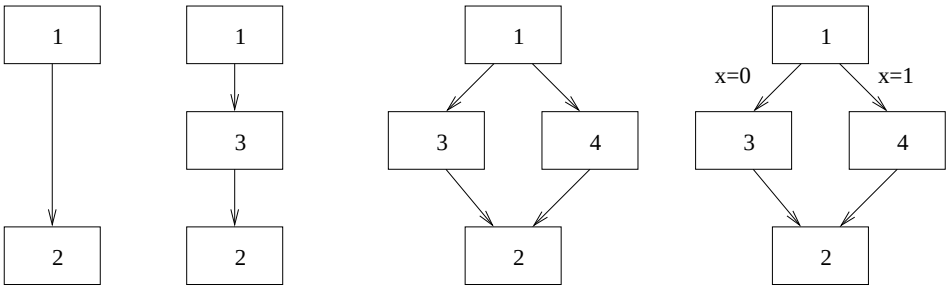


FIG. 3.10 – Insertion d'une activité

Quand l'ordre des activités est modifié (figure 3.11), deux activités peuvent être interchangées, des activités initiallement séquentielles peuvent être autorisées à s'exécuter en parallèle, ou les conditions d'exécution des activités peuvent changer.

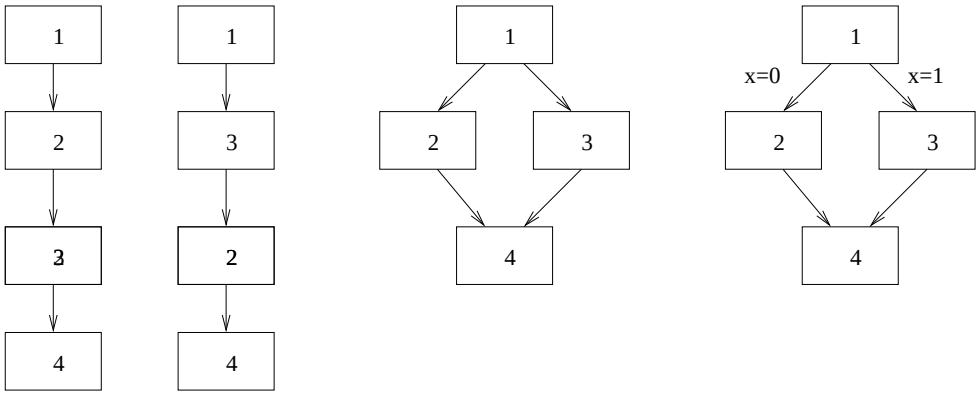


FIG. 3.11 – Modification de l'ordre des activités

Dans ADEPTflex[Rei98], un ensemble minimal d'opérations de modifications de workflow est présenté. Des critères de correction sont définis pour déterminer si un changement peut être appliqué à une certaine instance. Si ces contraintes sont violées, le changement est soit rejeté, soit la correction doit être restaurée de manière explicite avec des techniques de gestion des exceptions. Les changements respectent la consistence du flot de contrôle et du flot de données. Les auteurs traitent aussi une autre modalité de changement, la durée de vie du changement. Ils fournissent un cadre pour supporter des changements temporaires et permanents avec la possibilité de composer les changements et d'annuler les changements temporaires.

Dans [Liu98], un langage pour l'évolution du workflow est présenté, en accentuant les aspects dynamiques ; avec le langage proposé, le concepteur peut spécifier quelles instances doivent migrer

vers quelles versions, dépendant des conditions sur les données du workflow.

Le problème des modifications dynamiques a été abordé aussi dans le domaine de génie logiciel. Le langage SLANG, développé dans le contexte du projet SPADE [Ban93b] est un langage réflexif pour la modélisation des procédés logiciel. La définition d'un modèle SLANG inclut un ensemble de types et de définitions d'activités. Les activités correspondent aux unités logiques de travail et sont définies par des réseaux de Petri de haut niveau ; leur état est donné par le marquage du réseau. Les instances des activités sont appelées des copies actives, et incluent toutes les informations nécessaires pour exécuter l'activité : la définition de l'activité et l'état de la copie active. Les activités sont exécutées par un interpréteur SLANG et une instance nouvelle dédiée de l'interpréteur est créée quand une nouvelle activité doit être exécutée. Le langage, dû à sa nature réflexive, permet la manipulation des définitions des activités et des états des copies actives comme des données ordinaires du procédé ; ceci signifie que des méta-procédés qui agissent sur d'autres (méta-) procédés peuvent être définis, et en particulier il est possible de spécifier des procédés qui gèrent l'évolution des autres procédés.

Cependant, SLANG n'inclut pas de constructions pour l'évolution du modèle : les modifications des définitions des activités (évolution statique) ou des copies actives (évolution dynamique) sont représentées comme des boîtes noires ; SLANG ne fournit pas de langage de modification.

EPOS [Con91] est un environnement de développement conçu pour supporter l'évolution de larges systèmes logiciels. Un procédé est décrit par un réseau d'activités, et des activités composées peuvent être raffinées dans un réseau de sous-activités. Un gestionnaire d'exécution vérifie les pré-conditions de l'activité, exécute le code associé et vérifie les post-conditions. In [Jac93], le support pour l'évolution est présenté. EPOS, comme SPADE, offre des caractéristiques réflexives puisque les instances et les définitions des activités peuvent être accédées comme n'importe quel autre objet dans le système. Les changements d'un procédé sont réalisés à travers des modifications de la définition d'une activité composée. Les instances de l'activité en cours d'exécution sont automatiquement converties pour qu'elles suivent la nouvelle définition. Cependant, elles doivent être redémarrées (en perdant tout le travail déjà fait) si les modifications provoquent un état incohérent de l'instance.

Pour supporter ce type de flexibilité à l'exécution (flexibilité *a posteriori*), en dehors des opérations de modifications dynamiques, un système de workflow doit fournir des opérations d'intervention pour les utilisateurs. Avec ces opérations, les utilisateurs peuvent intervenir de manière active dans l'exécution des activités contrôlées par le système, en changeant le flot de contrôle prédéfini des activités.

Dans [Wes96], un ensemble prédéfini d'interventions humaines (sauter, arrêter une activité et supprimer une instance de procédé) est permis. Dans MOBILE [Jab96], des opérations spécifiques peuvent être définies en utilisant un langage d'opérations. Un système commercial qui fournit des opérations pour retravailler, avorter et compenser une instance est ProMInanD [Kar90]. Des exemples de systèmes commerciaux qui permettent la modification des instances pendant l'exécution sont InConcert et TeamWARE [Swe94].

Conclusion Les approches présentées se basent sur des modèles traditionnels simples et fournissent des primitives pour changer dynamiquement l'instance courante de procédé. Ces primitives modifient le flot de contrôle et de données en transformant le modèle de procédé. Le modèle résultant est une instance du métamodèle qui est simple. Si ces approches permettent de supporter des situations inattendues qui demandent d'ajouter/supprimer une activité, elles ne sont pas suffisantes à cause de la rigidité du métamodèle qui contrôle l'exécution de chaque instance, modifiée ou pas.

3.3.4 Conclusion sur les approches workflows flexibles

La première et la troisième approche considèrent la flexibilité au niveau de l'exécution du procédé. Dans le premier cas, le modèle est un guide pour atteindre un objectif ; dans l'autre cas, le modèle est un chemin pour atteindre l'objectif, mais ce chemin peut être changé pendant l'exécution du procédé. Dans la deuxième approche, c'est le modèle qui évolue pour fournir la flexibilité demandée.

La première approche considère le modèle comme un guide pour atteindre l'objectif et, par conséquence, les activités des participants ne sont pas contrôlées et coordonnées. Dans cette situation, d'autres moyens de synchronisation doivent être fournis pour l'accès aux données de procédés.

La deuxième approche essaie d'inclure la flexibilité dans le modèle ; elle offre des éléments du modèle pour représenter différents patrons de travail. Les procédés coopératifs étant fortement dynamiques et imprévisibles, il est difficile de modéliser à priori toutes les interactions entre les participants.

La troisième approche offre la possibilité de modifier dynamiquement le modèle de procédé avec des primitives pour la modification de l'ordre des activités et l'insertion des nouvelles activités. Le support pour les modifications dynamiques du modèle est un besoin important des applications coopératives. Cependant, contrairement aux modèles présentés, la flexibilité *a posteriori* (par des modifications dynamiques) doit être combinée avec une flexibilité *a priori* (permettant les exécutions coopératives illustrées dans la problématique).

Si toutes ces approches proposent des solutions pour une exécution flexible, l'échange de données flexible et contrôlé n'est pas pris en compte.

Dans la section suivante nous présenterons des modèles de transactions avancées qui adressent les problèmes de la cohérence des données accédées par plusieurs activités, d'habitude de longue durée.

3.4 Modèles de transactions avancées

Les approches de workflow transactionnel proviennent de la communauté de recherche des bases de données et attachent des propriétés transactionnelles aux workflows.

Le terme de workflow transactionnel a été introduit par Sheth et Rusinkiewick [She93] pour souligner l'importance des propriétés transactionnelles pour les workflows. En particulier, la propriété d'atomicité serait utile pour être appliquée à un procédé ou à une partie de procédé. Par exemple, on pourrait avoir besoin de déclarer deux activités comme une unité de travail atomique (payer une facture et soustraire le montant du compte du payeur).

Mais, un procédé ne peut pas être traité comme une seule transaction ACID pour les raisons suivantes :

- Les procédés sont de longue durée. Considérer tout le procédé comme une transaction demanderait de bloquer les ressources pour de longues périodes de temps.
- Les procédés impliquent typiquement plusieurs bases de données et systèmes indépendants. Assurer les propriétés ACID pour tout le procédé demanderait une coordination coûteuse entre ces systèmes.
- Comme les procédés ont souvent des effets externes, il n'est pas possible ou même désirable, de garantir l'atomicité en utilisant les mécanismes traditionnels d'annulation.

Quelques modèles de transactions longues ont été développé pour permettre la définition de propriétés similaires aux propriétés ACID au niveau d'un procédé et pour gérer l'échec des activités (voir [Elm92],[Jaj97]).

L'un des premiers modèles proposés pour gérer les transactions longues a été le modèle des SAGAs [GM87]. Basiquement, une *SAGA* consiste en une séquence de transactions ACID qui peuvent s'intercaler avec les transactions qui composent une autre SAGA. Cependant, la transaction SAGA garantit l'atomicité en compensant toutes les transactions composantes commitées dans l'ordre inverse. Les SAGAs relâchent la propriété d'isolation des transactions et augmentent le parallélisme inter-transactions. Une extension permettant d'emboîter les SAGAs a été proposée dans [GM91].

Tandis que la propriété d'isolation pour une SAGA est relâchée, une transaction composante reste isolée. Ce modèle ne peut pas être utilisé pour des activités longues, imprévisibles, qui ne peuvent pas être découpées à l'avance en transactions ACID.

Le modèle de ConTract [Wac92],[Reu95] partage l'idée de base des SAGAs, mais permet, en utilisant un *script*, la définition explicite du flot de contrôle et des informations entre étapes (transactions) prédéfinies. La figure 3.12 présente le «script» d'une réservation pour un voyage d'affaires. Intuitivement, à partir des données du voyage, il faut consulter les horaires d'avion, faire la réservation du vol, de l'hôtel et d'une voiture pour finir le contrat.

Le langage de script fournit des constructions de langage spéciales pour l'exécution parallèle et séquentielle, pour les branchements et les boucles. Similaire au modèle NT/PV [Kor88], il permet de définir des *invariants*. Les *invariants* sont des contraintes de cohérence définies dans la base de données pour contrôler la concurrence dans les conditions du relâchement de l'isolation. Une caractéristique du modèle est le fait qu'un ConTract est «recouvrable» en avant. Dans le cas d'une panne, un état cohérent d'un ConTract peut être restauré et son exécution peut continuer. Ceci est réalisé en utilisant des bases de données de contexte privées où l'état d'un ConTract est sauvegardé régulièrement. En plus, ConTract relâche l'atomicité, de telle manière qu'un ConTract peut être interrompu et re-instantié.

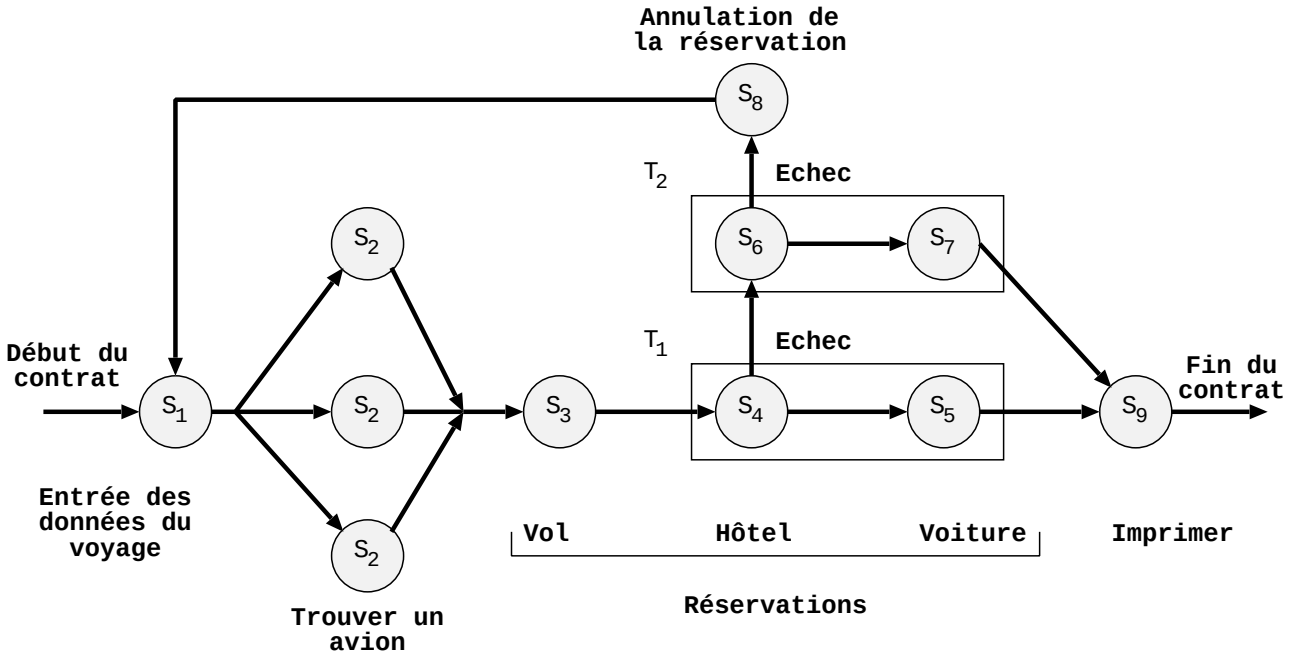


FIG. 3.12 – Exemple de contract

Les *Flex transactions* [Elm90] permettent de relâcher plus tôt les ressources. Une transaction

Flex consiste en un ensemble de tâches, chacune ayant un ensemble de transactions équivalentes du point de vue fonctionnel. Un ensemble de dépendances d'exécution sur les sous-transactions, incluant des dépendances d'échec, de succès, ou des dépendances d'événements externes peut être spécifié. Pour relâcher l'isolation les transactions Flex utilisent la compensation. L'atomicité globale est relâchée en permettant la spécification d'états de terminaison acceptables dans lesquels certaines transactions peuvent être avortées.

Les *ATMs* (Activities/Transaction Model) [Day91] permettent d'intégrer des transactions ACID et des activités qui ne doivent pas être ACID. Une activité de longue durée est modélisée comme un ensemble d'unités d'exécution qui peuvent consister en d'autres activités ou transactions, de manière récursive. Le flot de contrôle et le flot de données peuvent être spécifiés statiquement dans un script ou dynamiquement par des règles événement-condition-action. Le recouvrement est supporté par la spécification de règles de compensation ou de *transactions de contingence* qui représentent des alternatives.

Les modèles transactionnels qui ont été inventés par la suite (par exemple WAMO [Ede95],[Ede98], WIDE [Gre97] , CREW [Kam98]) ont repris les mêmes idées de base :

- attacher des propriétés transactionnelles à certaines parties des procédés,
- définir des activités de compensation et des alternatives,
- déclarer certaines activités comme critiques (ou vitales),
- définir des points dans le procédé jusqu'auxquels l'annulation doit être faite en cas d'échec.

L'exécution des dépendances inter-activités dans les workflows transactionnels est discutée dans [Att93]. Les activités sont modélisées en fournissant leurs états visibles et les événements correspondant aux transitions d'états (start, commit, abort, etc.). Les dépendances inter-activités peuvent être classées comme des dépendances qui peuvent être activées, rejetées ou retardées. Les dépendances entre les activités comme la dépendance d'ordre ou d'existence entre des événements significatifs sont spécifiées formellement en utilisant la logique computationelle d'arbre. Le système METEOR [Kri95] est aussi basé sur le concept de dépendance entre activités.

Le TSME (Transaction Specification and Execution Environnement) [Geo94] est un système programmable qui supporte la spécification des workflows transactionnels et la configuration des gestionnaires de transactions pour implanter les modèles désirés. Il fournit un langage de spécification pour exprimer les différents types de dépendances intra et inter-transactions et les critères de correction que la transaction doit satisfaire.

Similaire à TSME, *transaction Toolkits* permet l'implantation par programmation des modèles de transactions avancées en fournissant un ensemble de primitives de transactions. En utilisant une interface de programmation C++, ASSET [bil94] permet à un concepteur de transaction de compiler des spécifications de transactions avancées dans un code exécutable. ASSET se base sur des primitives de transactions dérivées du formalisme ACTA[10] [Chr90].

Le projet Transcoop [Faa97] définit dans un seul langage des propriétés transactionnelles et des propriétés organisationnelles pour un procédé. En contraste avec les autres approches, une activité n'est pas une transaction. Mais la correction des échanges de données est assurée en définissant des règles d'exécution qui spécifient des contraintes sur l'ordre des opérations sur les données tant dans l'espace d'un utilisateur, que dans les espaces privés (à travers des opérations d'exportation et d'importation)

[10] ACTA est un métamodèle de transactions qui permet la spécification des modèles de transactions avancées

3.4.1 Conclusion

Le fait que les modèles de transactions avancées soient centrés bases de données fournit un bon cadre théorique, mais limite leurs applications pratiques et leurs possibilités [Alo96]. Les modèles de workflow ont une sémantique plus riche et adressent des problèmes qui ne sont pas pris en compte par les modèles de transactions avancées : les aspects organisationnels, les activités humaines, les interfaces utilisateurs, la surveillance, la distribution, l'hétérogénéité, la simulation. Les modèles de workflow transactionnel qui combinent les workflows et les modèles de transactions, restent centrés données. Le développement d'un système de workflow basé sur un modèle particulier de transactions conduit à des restrictions majeures qui limitent son application et utilité comme outil de workflow.

Pour démontrer que les modèles de workflow sont un sur-ensemble des modèles de transactions avancées, le projet Exotica ([Alo94], [Alo96]) a décrit des méthodes et des outils pour implanter des modèles de transactions avancées au dessus de Flowmark (le système de workflow qui est le prédécesseur de IBM MQ Series). L'idée de base a été de fournir un modèle de workflow étendu qui intègre des concepts de transactions avancées. En particulier, les auteurs ont montré comment les SAGAs et les transactions flexibles pourraient être implantées en Flowmark. L'utilisateur peut définir une activité de compensation pour chaque activité du procédé. Un preprocesseur transforme ces spécifications en FDL (Flowmark Definition Language) en insérant des chemins de compensation après une activité ou un groupe d'activités, qui sont exécutés de manière conditionnelle après l'échec d'une activité.

Ces approches de workflow transactionnel se focalisent sur la coordination, c.a.d. l'exécution coordonnée d'un ensemble d'activités associées, exécutées par des humains ou automatiques. Les contraintes d'exécution sont généralement induites à partir des dépendances de données ou des contraintes organisationnelles.

La coopération est limitée au passage des résultats entre activités (transactionnelles) d'une manière programmée, prédéfinie. L'échange flexible des données entre activités n'est pas considéré par les workflows transactionnels. Ces modèles relâchent l'atomicité d'un procédé en le divisant en sous-transactions. Cependant, celles-ci gardent la propriété d'atomicité.

Ces modèles nécessitent une spécification complète des procédés : découpage en activités, échange de données entre activités et même, parfois, des actions de compensations. Pour ces raisons, ils sont difficile à adapter aux procédures d'entreprise[11], et encore plus aux procédés coopératifs qui sont dynamiques, imprévisibles et peu structurés.

3.5 Collecticiels

Le domaine du CSCW[12] a pour objectif d'utiliser les technologies informatiques pour faciliter la collaboration entre individus. Il comprend une gamme large de technologies : messagerie électronique, vidéo-conférences, éditeurs partagés, etc. Les logiciels de ce domaine sont appelés « collecticiels » ou « synergiciels ». Nous présentons des exemples de collecticiels qui gèrent le partage et l'échange de données. Ensuite, nous citons quelques systèmes plus récents qui intègrent des fonctions pour la coordination.

[11] business process
[12] Computer Supported Cooperative Work

3.5.1 Collecticiels traditionnels

La plupart des collecticels synchronisent l'accès coopératif aux données partagées d'une manière plus ou moins ad-hoc, sans utiliser des notions de correction formelles bien définies. Le contrôle de la concurrence dans les collecticiels asynchrones se base généralement sur des mécanismes comme le verrouillage explicite, contrôlé par les utilisateurs des objets, des modes de verrouillage différents, sémantique de verrouillage étendue et notifications. Certains systèmes utilisent des protocoles de passage de jeton, ou des mécanismes de tour de rôle pour la synchronisation des opérations concurrentes sur des données partagées, limitant en conséquence sévèrement la disponibilité des données. D'autres systèmes ne fournissent pas de contrôle de concurrence et se basent seulement sur des protocoles sociaux, fournissant seulement du feedback visuel pour indiquer les objets couramment utilisés.

Les systèmes d'édition coopérative hypermedia les plus connus appartenant à cette catégorie de collecticiels asynchrones sont Notecards [Hal88], CoAuthor [Jah89], KMS [Aks88], Intermedia [Yan88], EHTS [Wii91], HyperBase[Wii93] et HyperForm [Wii92]. Quilt [Lel88] et PREP [Neu92, Ell91] sont des exemples d'éditeurs de groupe qui utilisent le passage de versions intermédiaires et d'annotations pour permettre la coopération entre coauteurs.

L'inconvénient principal de ces systèmes est qu'ils n'offrent pas un mécanisme générique pour réaliser une gamme large d'applications coopératives. La plupart des systèmes utilisent des mécanismes *ad hoc* pour partager les données, ils sont souvent très restrictifs et ils manquent d'une notion formelle pour la cohérence des données. Certains systèmes offrent des mécanismes préprogrammés pour la coordination.

La plupart des collecticiels mentionnés ci dessus sont des prototypes de recherche et ils sont conçus pour un domaine d'application spécifique, par exemple l'édition coopérative. Les dernières années quelques plateformes commerciales de collecticel sont apparues qui peuvent être utilisées pour développer des applications coopératives (classifiées aussi comme méta-collecticiels par les chercheurs du domaine de CSCW).

3.5.2 Meta-collecticiels

Deux exemples représentatifs sont Lotus Domino [Lot97] (connu précédemment comme Lotus Notes) et des versions récentes de Microsoft Exchange [Mic98]. Ces plate-formes de collecticiel fournissent du support de haut niveau pour la communication, le routage des messages, des référentiels pour les documents et des outils pour développer des applications qui utilisent ces fonctionnalités de base. En offrant des langages de script, des outils pour le développement des formulaires, ils permettent à un concepteur d'application de créer une instance d'un collecticiel.

Lotus Notes est une plate-forme de développement client-serveur qui intègre une base de données et une infrastructure basée sur des messages. L'environnement de développement d'application (Notes Application Development Environment ADE) permet le développement des applications qui stockent et font circuler des objets d'informations en utilisant les services de base de données et de messagerie.

La *base de données de documents* Notes contient les documents. Un document Notes est défini comme un objet contenant du texte, des graphiques, des objets vidéo ou audio ou n'importe quel autre type de données formatées.

L'*infrastructure de messagerie* fonctionne pratiquement sur n'importe quel système d'exploitation. L'information n'est pas seulement stockée et retrouvée dans les bases de données, mais peut aussi circuler entre utilisateurs ou même bases de données.

Serveur Microsoft Exchange/ Client Microsoft Outlook Le client Microsoft Outlook, combiné avec Microsoft Exchange, fournit de la messagerie, un agenda de groupe, la gestion des informations personnelles et un environnement de développement de formulaires.

Outlook permet la création d'applications de workflow et de collecticiels basées sur des formulaires. Les formulaires Outlook peuvent être aussi inclus dans un message électronique (émail) qui peut être envoyé sur l'Internet. Des applications de collecticiel plus compliquées peuvent être développées en utilisant le système de programmation Visual Basic, les scripts et les contrôles ActiveX.

Conclusion Ces plate-formes permettent le développement des applications de workflow basées sur la circulation des documents. Par exemple, Domino fournit des fonctionnalités de routage pour permettre la construction d'applications de workflow avec des architectures basées sur une base de données ou sur une messagerie. Comme les systèmes de workflow, les deux collecticiels manquent de fonctionnalités qui permettent le partage flexible des documents d'une manière coopérative.

3.5.3 Intégration de la coordination

Dans cette section nous présentons des environnements de coopération étendus avec des fonctions de coordination.

Dans Chips (Coopérative Hypermedia for Process Support) [Haa99], les auteurs utilisent comme base un système coopératif hypermedia avec beaucoup de facilités pour la communication et la coopération et intègrent des capacités pour supporter les procédés (support pour une coordination plus formelle) dans le système. Les utilisateurs peuvent passer facilement de la définition à l'exécution du procédé du fait de l'interprétation duale d'une structure hypermédia comme structure du procédé et de l'information manipulée.

Le système prototype SCOPE [Mia98] intègre aussi des technologies pour le support des procédés dans un environnement de partage d'information. Le concept de base est celui de session, qui définit le fait que, dans une période de temps, les membres d'une équipe exécutent des activités dans un espace partagé. Un modèle de procédé est décrit comme un document hypermedia et il spécifie les sessions du procédé et les relations entre les sessions. Le système se base sur les facilités de l'outil COAST pour maintenir un espace partagé (mécanismes de contrôle au niveau des données, contrôle d'accès, contrôle de la concurrence). En permettant en plus la définition des protocoles de collaboration (diagramme de transition d'états d'un objet) par les utilisateurs, le système permet la coopération synchrone et asynchrone entre les membres d'une équipe et entre équipes.

L'intégration d'un système de workflow avec des outils de collaboration est étudié dans le projet Orchestra [Ant95], [Gui97]. Les outils de collaboration sont utilisés comme des mécanismes pour résoudre les exceptions qui résultent de la coordination. Le système de workflow doit identifier les situations où il n'existe pas de solutions formalisées. Une fois identifiées et classées comme des problèmes qui peuvent être résolus avec une interaction informelle, une technique est sélectionnée et l'outil qui la supporte est démarré. Les outils intégrés sont des outils d'aide à la négociation et à la décision du groupe. Quand le problème est résolu, le contrôle est retourné au moteur de workflow.

3.5.4 Conclusion

Si les technologies CSCW sont utiles pour résoudre des problèmes de communication et de collaboration liés à la distribution des participants dans le temps et dans l'espace, ils manquent de capacités de guidage et de mécanismes d'automatisation pour les procédés structurés. Si l'utilisation de ces outils pour résoudre des tâches non-structurées miment certains environnements réels de travail, ils ne garantissent pas la cohérence des résultats, ne maintiennent pas une procédure de travail standardisée et ne conduisent pas à l'optimisation et à l'apprentissage du procédé de travail. Il n'existe pas de modèle explicite de travail autre que l'utilisation *ad hoc* des outils disponibles. Ces systèmes ne fournissent pas de moyens pour mesurer l'accomplissement, le progrès d'une activité, ni des mécanismes pour décrire ce qui doit être fait d'autre que la capture des relations de communication et de collaboration entre participants. L'ajout d'un modèle de travail à un outil de collecticiel conduit souvent à un travail supplémentaire du participant sans bénéfice très concret pour le guidage ou l'automatisation. La synchronisation entre le travail proposé et le travail effectif est souvent faite à la main. Cette surcharge peut conduire à la demotivation des participants.

3.6 Synthèse

Dans ce travail nous nous sommes fixé pour objectif d'offrir un support informatique à la coordination et la coopération des membres d'une équipe virtuelle. Pour améliorer l'efficacité de la coopération, les activités de l'équipe doivent être coordonnées. L'outil doit permettre de définir, exécuter et faire évoluer les procédés que l'équipe virtuelle doit suivre pour atteindre son objectif.

En analysant les diverses approches pour la coordination des équipe virtuelles, il nous semble que les solutions actuelles ne répondent pas aux besoins d'exécution flexible, d'échange de données flexibles et de dynamique des procédés.

Les systèmes de développement de logiciel centrés procédé se focalisent sur la gestion des données partagées et offrent des fonctions limitées pour la gestion des procédés. Ils sont conçus pour gérer les problèmes spécifiques à la conception de logiciel et sont souvent limités à cette classe d'applications.

Les deux principaux domaines de recherche qui offrent des solutions générales pour le travail coopératif sont présentés dans la figure 3.13 : les workflows et les collecticiels. Les workflows se focalisent sur la coordination des participants dans des procédés très structurés, pendant que les collecticiels se focalisent sur le support à la coopération des groupes participant dans des procédés plus «fluides», moins structurés (généralement, à travers des espaces partagés, permettant l'échange d'informations).

Les environnements qui permettent le partage d'information aident les participants au procédé à se réunir, à accumuler leurs connaissances et à travailler effectivement, indépendamment de leur distribution dans le temps et dans l'espace. Cependant, ces systèmes fournissent moins de support aux utilisateurs pour organiser et coordonner leurs activités afin d'atteindre un objectif dans un contexte organisationnel. La coordination se base fortement sur l'expérience des individus et des équipes et sur la qualité de leurs interactions.

Les systèmes de workflow actuels offrent un bon support à la coordination, mais ne sont pas adaptés à la coopération à travers les objets partagés du procédé. Dans ces systèmes il existe une séparation entre la coordination (en utilisant le modèle de procédé) et la coopération sur les informations ou les documents partagés (qui sont, en général, maintenus en dehors du système

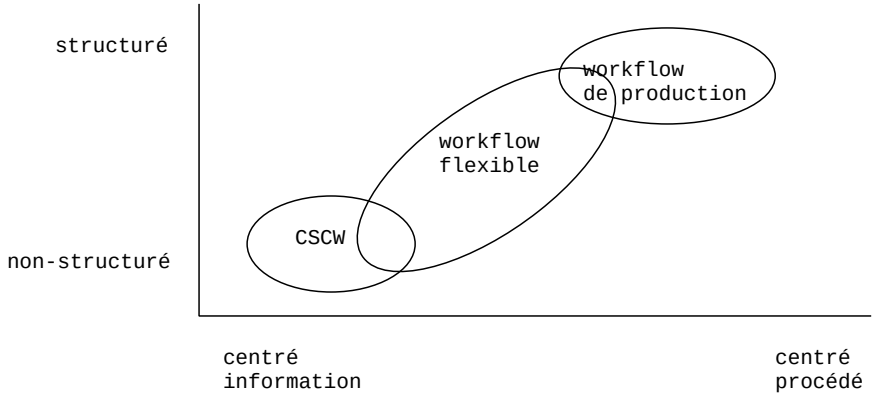


FIG. 3.13 – Le spectre du workflow

de workflow). De plus, les modèles de coordination manquent de flexibilité pour les applications coopératives.

Les workflows transactionnels ont comme objectif d'assurer la cohérence des données échangés. Ces modèles ne peuvent pas être appliqués aux procédés coopératifs parce qu'il nécessitent une modélisation complète du procédé, de ses activités et de l'échange de données entre activités.

Afin de répondre aux besoins des applications coopératives qui ne sont couverts ni par les workflows de production, ni par les collectiels, quelques projets de recherche proposent des solutions augmentant la flexibilité des modèles de workflow. Ces approches offrent des solutions pour une exécution plus flexible, pour modéliser des situations de travail spécifiques, pour supporter les modifications dynamiques du procédé. En examinant ces modèles, nous avons pu constater que leurs modèles font souvent appel à des constructions complexes et qu'ils ne gèrent pas l'échange de données flexible de façon satisfaisante.

En conclusion, les différentes approches présentées fournissent une gamme large du support, mais généralement accentuent soit la coordination, soit la coopération. Pour permettre la flexibilité nécessaire aux procédés coopératifs, une approche qui intègre les deux aspects, est nécessaire.

Notre proposition pour un workflow flexible vise à offrir un support qui :

- permet la coordination à l'aide d'un modèle formel simple et la coopération grâce à un échange de données flexible, mais contrôlé,
- permet l'exécution des procédés comprenant des parties structurées et moins structurées (voir figure 3.13),
- permet différents degrés de flexibilité dans l'exécution d'une séquence d'activités.

Chapitre 4

Modèle de workflow coopératif

4.1 Introduction

La principale contribution de cette thèse est la définition d'un système de workflow coopératif comme résultat de l'intégration de deux composants : un système de workflow et un gestionnaire de transactions. Le système de workflow doit permettre la description des procédés et leur exécution de façon suffisamment flexible pour répondre aux demandes de parallélisme, d'anticipation et d'échange flexible de données introduites précédemment.

Partant de l'observation que les procédés compétitifs et coopératifs peuvent être décrits de la même manière, avec le même formalisme, nous utilisons un modèle de description de workflow traditionnel. Ce modèle est décrit dans la première partie du chapitre. L'exécution coopérative d'un procédé est réalisée en interprétant d'une manière flexible ce modèle de description. En particulier, le modèle d'exécution permet le démarrage anticipé des activités ainsi que l'échange de données intermédiaires, en plusieurs versions entre ces activités.

Le composant workflow gère le flot de contrôle et la circulation des données de procédé (données qui participent aux choix des chemins d'exécution dans l'instance courante du procédé). Les procédés coopératifs nécessitent des mécanismes supplémentaires pour l'échange et la gestion des leurs données. A cet effet, nous avons introduit un modèle de gestion de transactions coopératives. L'intégration de ces deux composants offre un support pour la modélisation et l'exécution des procédés coopératifs avec différents degrés de flexibilité, en fonction des besoins spécifiques du procédé.

Le chapitre est organisé comme suit :

- Une partie introductive qui décrit le modèle de définition de procédés, qui est donc le modèle traditionnel défini pour les procédés administratifs ou de production [Ley00].
- Une deuxième partie présente l'interprétation coopérative de ce modèle. Le modèle d'exécution traditionnel est modifié pour permettre l'anticipation des activités et la circulation versionnée des données entre activités. Les problèmes de cohérence de l'exécution induits par ces modifications sont analysés.
- La troisième partie décrit le composant « gestionnaire de transactions coopératives ».
- La dernière partie analyse l'intégration des deux composants. Plus précisément, les aspects suivants sont abordés : le dialogue entre les deux composants, les fonctionnalités supplémentaires qui doivent être assurées par chaque composant, et le rapport entre les dépendances entre activités générées par l'échange de données et les dépendances définies dans le flot de contrôle.

4.2 Modèle de définition d'un procédé

Un modèle de procédé est la représentation d'un procédé dans une forme qui permet sa manipulation automatique (modélisation, exécution) par un système de gestion de workflow. Un modèle de procédé est constitué d'un réseau d'activités et des dépendances entre elles, des critères pour spécifier le démarrage et la terminaison d'un procédé et des informations sur les activités individuelles (participants, applications, données informatiques associées, ···) [Coa95].

Ce modèle de procédé est interprété par le moteur de workflow, et sert à créer des instances et à contrôler leur exécution.

Plusieurs méthodes ont été développées pour permettre la description des modèles de procédés. Une des plus utilisées est basée sur les graphes d'activités. Les raisons pour lesquelles elle est une des plus utilisées sont sa simplicité d'un côté, et sa puissance d'expression de l'autre côté.

Les concepts utilisés pour décrire un modèle de procédé et la description précise des propriétés et du comportement des instances du modèle s'appellent *métamodèle*.

Les concepts fournies par le métamodèle constituent sa *syntaxe*. La description précise des propriétés et du comportement des instances du modèle s'appelle la *sémantique* du mètamodèle.

Nous nous limitons à présenter les concepts fondamentaux nécessaires pour décrire notre approche. Pour chaque concept, nous donnons une description textuelle (une formulation mathématique est donnée dans l'annexe A). Les notations utilisées ont été principalement prises dans [Ley00].

La syntaxe et la sémantique de notre métamodèle de workflow se base sur la théorie de graphes. Soit $\mathcal{P}$ l'ensemble de tous les modèles de procédés. Un modèle particulier de procédé $P \in \mathcal{P}$ est représenté par un graphe orienté G. L'ensemble des noeuds N de G comprend toutes les activités du modèle P associé. L'ensemble E des arcs de G décrit toutes les séquences d'activités possibles dans P. Un ensemble de conditions $\mathcal{C}$ (règles de procédé ou prédicats) détermine l'enchaînement réel des activités pour une instance particulière de P. Ces conditions sont formulées dans les termes du contexte d'un modèle P, c.a.d. des données manipulées par ses activités.

4.2.1 Données de procédé

A chaque modèle correspond un ensemble de données qui décrivent toutes les informations nécessaires pour son exécution.

Ces données incluent : des informations requises en entrée des activités, des informations requises pour l'évaluation des conditions qui déterminent l'enchaînement réel des activités, des données qui doivent être échangées entre activités.

L'ensemble de ces données nécessaires à un procédé particulier pour son exécution correcte s'appellent *données pertinentes pour le procédé*, ou *données de procédé*.

Les données de procédé sont rassemblées dans l'ensemble V, qui est associé à un modèle de procédé P. Un membre $v \in V$ a un nom et un type et il s'appelle *élément de données*.

4.2.2 Conteneurs de données

Les activités et les procédés ont des données en entrée et en sortie. Les données d'entrée et de sortie sont représentées comme des ensembles d'éléments de données, appelés *conteneurs*.

L'application i associe à chaque activité, modèle de procédé et condition, son conteneur d'entrée. L'application o associe à chaque activité et modèle de procédé son conteneur de sortie. (Les conditions ont seulement des conteneurs d'entrée, parce que leur sortie est toujours une valeur de vérité.)

Une collection d'instances des données de $i(X)$ ou $o(X)$ est appelée une instance conteneur ou une instance de conteneur.

Ainsi, les conteneurs sont des groupements explicites de données d'entrée et de sortie. Le conteneur d'entrée et de sortie sont des ensembles de données associées avec les entrées et, respectivement les sorties possibles de X.

4.2.3 Activités

Un procédé définit comment un groupe de travail réalise un objectif particulier, comme, par exemple, éditer un document ou corriger une erreur dans un programme. Il définit chaque unité de travail (activité) qui doit être réalisée pour atteindre cet objectif.

La définition d'une activité inclut la spécification des données d'entrée et de sortie, l'outil utilisé pour réaliser l'unité de travail associée, la personne qui a la qualification pour la réaliser et la méthode pour déterminer si le travail est terminé ou pas.

L'ensemble des activités d'un procédé définit l'ensemble des noeuds N du graphe du modèle de procédé associé.

Une activité définit le type de ses entrées et sorties comme une spécification abstraite au niveau du modèle, et est donc, perçue comme un opérateur. L'outil qui implante l'activité et qui sera ultérieurement invoqué par le système de gestion de workflow va recevoir une instance du type d'entrée et va produire une instance du type de sortie.

L'application Ψ associe à chaque activité son implantation. L'implantation d'une activité $\Psi(A)$ peut être un programme, ou un modèle de procédé. Dans le premier cas, elle s'appelle *activité programme*, dans le deuxième cas, *activité procédé* ; un procédé qui implante une *activité procédé* est aussi appelé *sous-procédé*.

Une implantation d'activité est une application qui associe à une instance du conteneur d'entrée, une instance du conteneur de sortie. Quand une activité A est démarrée, le système de gestion de workflow construit une instance du conteneur d'entrée de A (matérialisation du conteneur). Ensuite, l'implantation de l'activité $\Psi(A)$ est invoquée et reçoit cette instance comme entrée. Quand l'implantation de l'activité se termine, une instance du conteneur de sortie est retourné au système de gestion de workflow. Le système de gestion de workflow rend persistante l'instance retournée (dématérialisation du conteneur de sortie).

Attribution des activités Le travail est réalisé par des acteurs. Un acteur peut être une personne, mais aussi une unité computationelle, comme un démon qui exécute sur une machine particulière une activité de manière automatique.

Le métamodèle associe à chaque activité A une requête $\Omega(A)$ pour déterminer dynamiquement l'ensemble des acteurs qui peuvent réaliser le travail correspondant, appelé requête de personnel. Cette requête calcule les acteurs en se basant sur une base de données organisationnelle qui définit les rôles et les unités organisationnelles.

Puisque cette base peut changer dans le temps, la requête de personnel est dépendante du temps. Le temps est reflété dans le métamodèle par l'ensemble des nombres entiers positifs $\mathbb{N}$. Quand la requête est évaluée au moment i, elle retourne l'ensemble des acteurs $\Omega(A)(i) \subseteq \mathcal{A}$ qui peuvent exécuter l'activité, où A dénote l'ensemble de tous les acteurs.

Condition de sortie Chaque activité A a une condition de sortie associée $\varepsilon(A)$, ce qui permet de suspendre une activité et de la continuer plus tard. Quand l'implantation de l'activité retourne au système de workflow, la condition de sortie est évaluée sur l'instance courante du conteneur de

sortie. Quand la condition de sortie est satisfaite, le travail associé est considéré comme terminé, sinon il est supposé être seulement interrompu et il doit reprendre ultérieurement.

Date limite Pour la plupart des procédés, il est important qu'ils soient réalisés dans une certaine période de temps. Le concepteur du procédé peut définir des dates limites au niveau du procédé ou des activités ; des actions de notification ou des exceptions peuvent être déclenchées pour le cas où la limite de temps est dépassée.

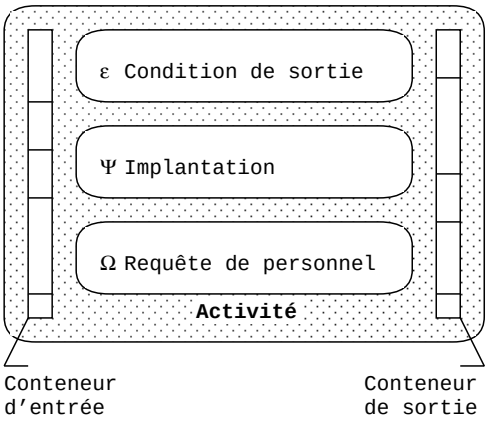


FIG. 4.1 – Structure d'une activité

La figure 4.1 illustre les éléments de définition d'une activité : la requête de personnel, l'implantation, la condition de sortie et les conteneurs de données.

4.2.4 Flot de contrôle

Un modèle de procédé décrit chaque unité de travail, mais aussi la séquence adéquate d'unités de travail pour atteindre un certain objectif.

Quand un modèle de procédé est spécifié, pour chaque activité A, toutes les activités qui peuvent potentiellement la suivre $A_1, A_2, \ldots A_n$ doivent être spécifiées. Graphiquement, A sera connectée avec $A_1, A_2, \ldots A_n$ par des arcs orientés.

Les successeurs de A dans l'instance courante dépendent des règles qui contrôlent chaque transition potentielle $(A, A_1), \ldots (A, A_n)$. Ces règles s'appellent *conditions de transition* et sont des prédicats sur les données de procédé, plus précisément sur les données des conteneurs d'entrée des conditions.

Dans le métamodèle, la dépendance de la transition potentielle de A à B, de sa condition de transition associée p est définie explicitement en spécifiant le flot de contrôle comme un triplet (A, B, p), appelé connecteur de contrôle. Autrement dit, les arcs du graphe orienté G sont pondérés par des prédicats sur V.

L'ensemble de tous les connecteurs de contrôle, noté E de chaque modèle de procédé doit respecter les deux contraintes suivantes :

1. Entre deux activités il y a au plus un connecteur de contrôle. Deux activités soit ne sont pas liées, soit sont liées par un et un seul connecteur de contrôle.
2. Les connecteurs de contrôle ne forment pas de boucles [13].

[13] Les boucles dans un procédé peuvent être spécifiées en utilisant les conditions de sortie pour un sous-procédé ;

Dans la suite, nous utilisons également les notations suivantes :
- C est l'ensemble de toutes les conditions,
- l'ensemble de toutes les conditions de transition de tous les connecteurs de contrôle qui pointent vers l'activité A est noté $C^{\leftarrow}(A)$,
- l'ensemble de toutes les conditions de transition de tous les connecteurs de contrôle qui partent de l'activité A est noté $C^{\rightarrow}(A)$.

Noeuds de branchement et de jointure L'objectif principal pour lequel la technologie workflow est appliquée dans la pratique est d'accélérer l'exécution des procédés. Les trois techniques fondamentales pour diminuer la durée de l'exécution sont l'affectation des activités, la notification et le parallélisme.

Le mécanisme principal fourni au niveau du métamodèle pour accélérer l'exécution est la possibilité de spécifier des flots de contrôle parallèles. Une activité qui a plus d'un connecteur de contrôle sortant est une *activité de branchement*. Les branches peuvent s'exécuter en parallèle. Plus précisément, les activités qui peuvent s'exécuter en parallèle sont celles dont les conditions de transition des connecteurs de contrôle ont été évalués à vrai.

Le travail en parallèle doit être synchronisé. Le métamodèle réalise la synchronisation avec les activités de jointure. Une *activité de jointure* est une activité ayant plus d'un connecteur de contrôle entrant (l'activité B dans la figure 4.2).

Nous notons l'ensemble de toutes les activités de jointures avec N_* et nous définissons $N_\bullet := N - N_*$ l'ensemble des noeuds réguliers (activités avec un seul connecteur de contrôle entrant).

Note : La spécification du flot de contrôle présente beaucoup de ressemblances avec les langages de programmation. A l'aide des noeuds de branchement et de jointure, le métamodèle permet la programmation parallèle. Le support pour les séquences est évident, les branches sont modélisées comme des conditions de transition complémentaires et les boucles sont modélisées comme des conditions de sortie sur les sous-procédés (un sous-procédé est répété tant que la condition de sortie n'est pas satisfaite). Ainsi, les concepts du flot de contrôle permettent la spécification des « programmes » parallèles.

Condition de jointure Pour mieux contrôler la synchronisation du travail, une activité de jointure A a une condition de jointure associée $\Phi(A)$, qui est une expression booléenne sur les conditions de transition des connecteurs de contrôle entrant dans l'activité.

La figure 4.2 illustre une activité de jointure de type rendez-vous (AND-Join).

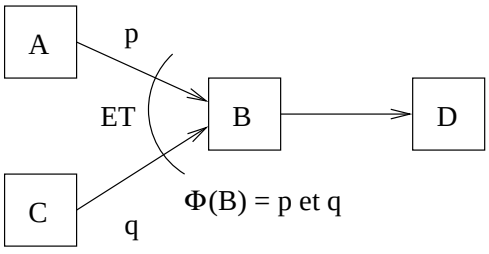


FIG. 4.2 – Activité de jointure

Pour exécuter une activité de jointure, les conditions suivantes doivent être satisfaites :

le sous-procédé est répété tant que sa condition de sortie n'est pas vraie

- Les conditions de transition de tous les connecteurs de contrôle entrant doivent avoir été évaluées avant que la condition de jointure soit calculée, et
- la condition de jointure associée doit être vraie.

Note : Tant qu'une condition de jointure n'est pas évaluée, la condition de jointure a la valeur de vérité «non-connue». Ainsi, nous utilisons une logique avec trois valeurs de vérité dans le cas des conditions de jointure.

Le fait que les activités de jointure doivent attendre que toutes les conditions de transition entrantes soient évaluées augmente la complexité de l'implantation des systèmes de gestion de workflow. Pour ne pas attendre indéfiniment, le système de gestion de workflow doit résoudre le cas ou il devient évident que le travail sur une branche qui entre dans l'activité de jointure dans le modèle, n'arrivera jamais à l'activité de jointure, dans l'instance courante du procédé. Le modèle détecte les activités qui ne sont pas exécutées dans l'instance courante, qui peuvent être soit :

- un noeud régulier dont la condition de transition est évaluée à faux, soit
- une activité de jointure dont la condition de jointure est évaluée à faux.

Chaque fois qu'une telle activité est détectée, une procédure de marquage du chemin mort (chemin qui ne peut pas être atteint dans l'instance courante) calcule la clôture transitive jusqu'aux noeuds de jointure et de fin et met les conditions de transition sur les chemins calculés à la valeur de vérité fausse.

4.2.5 Flot de données

La définition des données du procédé comprend d'une part la spécification des conteneurs d'entrée et de sortie des activités et des conditions et d'autre part la spécification du flot de données entre activités.

Soit A une activité et B une autre activité ou un prédicat. Si B utilise en entrée des données qui sont supposées être produites par A, alors, cette dépendance de données entre A et B est exprimée par un connecteur de données entre A et B. Le métamodèle permet aussi de spécifier quelles données du conteneur d'entrée de B attendent des valeurs de quelles éléments de données du conteneur de sortie de A. Cette définition est réalisée en spécifiant simplement la paire correspondante d'éléments de données (v_1, v_2) avec $v_1 \in o(A)$ et $v_2 \in i(B)$. L'ensemble de toutes ces paires spécifiées pour A et B est noté $\Delta(A, B)$. Une paire (v_1, v_2) ajoutée à $\Delta(A, B)$ aura l'effet suivant : quand le conteneur d'entrée de B sera matérialisé, l'élément de donnée v_2 va recevoir une copie de l'instance courante de l'élément de donnée v_1 du conteneur de sortie $o(A)$. Une restriction appliquée aux connecteurs de données spécifie qu'une activité B peut attendre des données d'une autre activité A si et seulement s'il existe un chemin dans le graphe du procédé de A à B (dans ce cas, on dit que B et atteignable à partir de A).

Une autre condition interdit que deux connecteurs de données aient le même élément de données comme destination. Cette contrainte évite les conflits à la matérialisation d'un conteneur qui pourraient être provoqués si deux connecteurs de données fournissent des données différentes.

Conteneur d'entrée et de sortie d'un procédé Un modèle de procédé $P \in \mathcal{P}$ peut avoir un conteneur d'entrée $i(P) \subseteq V$ et un conteneur de sortie $o(P) \subseteq V$. Ceci permet de traiter un procédé comme une abstraction computationelle qui reçoit des données en entrée, fait quelques calculs et retourne des données en sortie. Ainsi, un procédé se comporte comme une activité et peut être utilisé comme implantation d'activité. Comme effet secondaire, les procédés deviennent des entités réutilisables.

L'application $\vec{\Delta}$ reflète comment les données sont échangées entre les conteneurs du procédé et ses activités.

Pour spécifier que la donnée x du conteneur d'entrée du procédé sera copiée dans la donnée y du conteneur d'entrée de l'activité ou du prédicat X, la paire (x, y) sera ajoutée à $\vec{\Delta}(X)$. De manière similaire, pour spécifier que la donnée v de sortie de l'activité A incluse dans le procédé sera copiée dans l'élément z du conteneur de sortie du procédé, la paire (v, z) est ajoutée à $\vec{\Delta}(A)$.

4.3 Modèle d'exécution d'activités coopératives

Dans cette section nous décrivons comment le système de workflow va interpréter un modèle de procédé, c'est à dire la *sémantique opérationnelle du métamodèle*. Si la syntaxe du modèle présenté dans la section précédente est celle d'un modèle traditionnel, la sémantique est différente. Elle permet une exécution plus flexible du procédé, en relâchant, dans certaines conditions, la condition de démarrage d'une activité.

Quand un procédé est instancié, le système de gestion de workflow assigne des activités aux participants du procédé en interprétant l'information contenue dans le modèle de procédé, défini par le concepteur du procédé. Il est important de décrire comment le modèle est interprété pendant l'exécution pour s'assurer qu'il s'agit du comportement souhaité par le concepteur du procédé.

Dans ce but nous définirons tous les états possibles de chaque élément du métamodèle et les opérations qui transforment ces états. Nous allons mettre en évidence les modifications par rapport au modèle d'exécution classique.

Nous modélisons le temps par l'ensemble des nombres naturels, $\mathbb{N}$. Le métamodèle ne prend pas en compte l'interaction entre les instances différentes de procédés, donc, nous pouvons associer à chaque instance de procédé son axe de coordonnées temps. Sur cet axe, $0 \in \mathbb{N}$ représente le moment où l'instance a été créée. Quand un événement qui est significatif pour le procédé se produit, le paramètre temps courant, $i \in \mathbb{N}$ est incrémenté et l'état de tous les éléments du modèle du procédé, comme les activités, les conteneurs et les prédicats, est modifié. Le parcours d'un modèle pour exécuter le procédé afférent par un moteur d'exécution, est appelé *navigation*. La navigation est décrite dans le métamodèle par un ensemble d'applications paramétrées par le temps.

Le calcul de nouveaux états pendant l'exécution d'un procédé ne dépend pas seulement de l'état courant des éléments du modèle du procédé, mais aussi des données du procédé représentant le contexte actuel du procédé. A chaque moment, exactement un seul événement pertinent pour le procédé, est pris en compte. Par exemple, si plusieurs activités se terminent en même temps, leur terminaison est prise en compte séquentiellement.

Dans la partie suivante, nous décrivons de manière intuitive l'interprétation flexible du flot de contrôle défini dans le modèle de procédé pendant la navigation, en introduisant l'idée d'anticipation. Cette flexibilité est supportée en modifiant le graphe d'états d'une activité. Les états d'une activité seront définis de manière formelle. Ensuite, nous décrivons la gestion et l'attribution des activités et comment une étape de navigation se déroule dans notre modèle. L'anticipation nécessite aussi la modification du flot de données, plus précisément la circulation versionnée des données entre activités. Finalement, nous analysons les problèmes de cohérence qui doivent être résolus.

4.3.1 Anticipation - Définition intuitive

Les systèmes traditionnels de gestion de systèmes de workflow imposent une dépendance de type démarrage-fin entre les activités. Ceci signifie qu'une activité ne peut commencer que si les activités précédentes sont terminées. Cependant, dans les procédés coopératifs qui n'utilisent pas un système informatique comme support pour la coordination, la plupart du temps, les activités se superposent et commencent leur travail avec des résultats intermédiaires (en opposition aux résultats finaux) des activités précédentes, même si toutes les conditions pour leur exécution ne sont pas satisfaites :

- quelques unes des activités précédentes ne sont pas encore terminées (travailler sur des brouillons ou des résultats partiels) ;
- les conditions de démarrage sont actuellement évaluées à faux ou ne peuvent pas être évaluées (un certain visa manque, certains tests n'ont pas été passés) ;
- les données d'entrée ne sont pas complètes (mais les données disponibles sont suffisantes pour commencer à travailler).

L'anticipation est le moyen que nous proposons pour implanter cette façon naturelle de travailler. L'anticipation permet à une activité de commencer son exécution plus tôt par rapport aux dépendances de contrôle définies dans le modèle de procédé. L'activité peut ainsi travailler sur les résultats intermédiaires des activités précédentes. Quand les activités précédentes sont terminées et quand toutes les conditions de démarrage sont satisfaites, l'activité qui a anticipé passe à l'état normal d'exécution. Elle continue son exécution comme si elle n'avait jamais anticipé, comme si elle avait attendu que les activités précédentes se terminent avant de commencer son travail. À ce moment, les valeurs finales de ses paramètres d'entrée sont disponibles. Comme elle a été déjà commencée, l'activité peut probablement terminer son exécution plus tôt.

L'anticipation permet une exécution plus flexible tout en préservant, en même temps, les dépendances de terminaison entre les activités. Le schéma d'exécution dans la figure 4.3 est un exemple d'exécution sans (1) et avec (2) anticipation. L'activité « Edition » fournit une version intermédiaire du document édité à l'activité « Revue ». Dans ce cas ci, les activités « Revue » et « Modification » peuvent commencer plus tôt ; le procédé entier peut ainsi être terminé plus tôt.

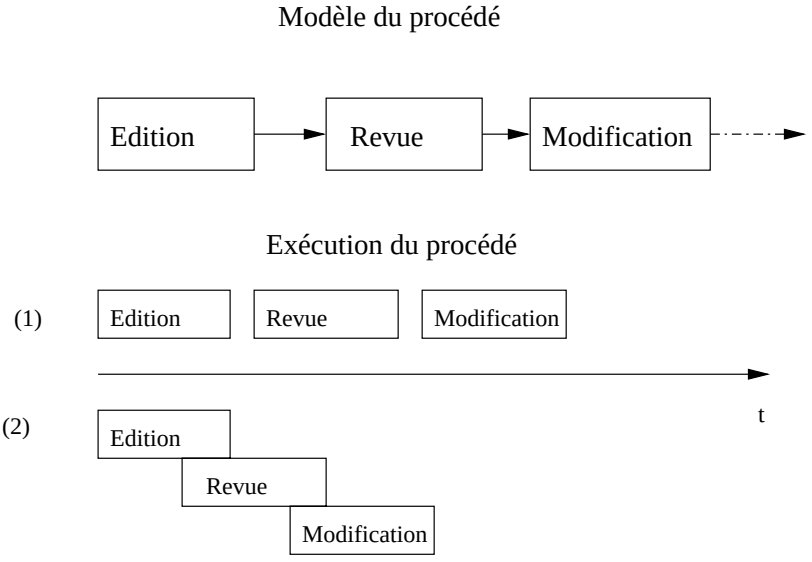


FIG. 4.3 – Exécution d'un procédé (1) sans et (2) avec anticipation

Cycle de vie d'une activité

Pour permettre l'anticipation, le diagramme d'états d'une activité est modifié, en introduisant deux nouveaux états : « prêt à anticiper » et « anticipant » (cf. figure 4.4).

Quand un procédé est créé, toutes les activités sont dans l'état « initial » (sauf les activités de démarrage, qui sont dans l'état « exécutable »). Une activité passe à l'état « exécutable » dès que les précédentes sont finies et si les conditions de transition sont vraies. Dès qu'une activité devient « exécutable », elle est assignée à tous les agents qui satisfont sa requête de personnel. Quand elle est choisie par un agent (ou si elle est automatique, quand son implantation est démarrée), l'activité devient « active ». Quand son exécution se finit (la condition de sortie est vérifiée), l'activité passe à l'état « fini ».

Ce qui est caractéristique à la coopération dans le modèle d'une activité coopérative c'est l'anticipation. Dans certaines conditions qui seront discutées par la suite, une activité peut passer de l'état « initial » à l'état « prêt à anticiper ». De même que pour la transition à l'état « exécutable », l'activité est assignée à tous les agents qui satisfont la requête de personnel. Une activité « prêt à anticiper » peut devenir soit « exécutable » si toutes ses conditions sont satisfaites avant qu'un acteur la démarre, soit « anticipante » quand un de ces acteurs choisit de démarrer plus tôt son exécution. Une activité dans l'état « prêt à anticiper » passe à l'état « actif » quand elle est dans une situation où elle aurait la permission de commencer son exécution si elle n'avait pas anticipé. En d'autres termes, quand les activités précédentes sont terminées.

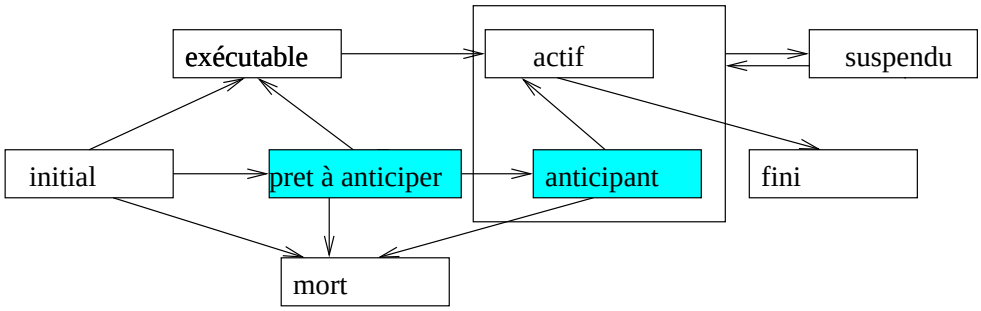


FIG. 4.4 – Diagramme d'états d'activité

Conditions pour une activité pour passer à l'état « prêt à anticiper » En ce qui concerne le moment où une activité dans l'état « initial » peut commencer à anticiper, plusieurs stratégies peuvent être considérées :

1. Anticipation libre - une activité dans l'état « initial » peut anticiper à tout moment (l'état « prêt à anticiper » fusionné avec l'état « initial »). Ceci permet aux activités de commencer leur travail plus tôt. En d'autres termes, les dépendances de flot de contrôle définies dans le modèle de procédé sont interprétées au moment de l'exécution comme des dépendances fin - fin : une activité peut terminer son exécution seulement quand l'activité précédente a terminé. Dans notre exemple, il signifierait que « Modification » et « Revue » pourraient commencer à tout moment. Avec cette stratégie, la convergence vers un état consistent peut être difficile et implique le risque d'avoir effectué un travail inutile. L'anticipation libre devrait être réservée à des cas exceptionnels.
2. Anticipation dictée par le flot de contrôle - une activité peut anticiper quand son prédécesseur direct a commencé à travailler, c.-a.d.. est en état « anticipant » ou « actif ». Pour

une activité de type jonction (OR-join), au moins l'un de ses prédécesseurs doit être dans l'état « anticipant » ou « actif ». Pour une activité de type rendez-vous (AND-Join), toutes les activités précédentes doivent avoir démarré. Dans ce cas ci, la dépendance traditionnelle de démarrage - fin entre les activités est relâchée, étant remplacée par la dépendance démarrer-démarrer ; c.-à.d.. l'activité successeur peut commencer son exécution dès que le prédécesseur la commence. Dans notre exemple, l'acteur en charge de l'activité de modification pourrait commencer dès que l'activité de revue commence. Ce n'est pas la stratégie la plus adéquate dans ce cas-ci, mais elle se base sur la responsabilité des utilisateurs. Par exemple, si l'acteur exécutant « Modification » n'est pas le même que celui exécutant « Édition », il peut commencer à lire le document pour évaluer le travail qu'il devra effectuer, même s'il n'a pas encore reçu les commentaires. Cette stratégie diminue le risque de travail perdu.

3. Anticipation dictée par le flot de contrôle et de données - une activité peut anticiper quand ses prédécesseurs sont en état « anticipant » ou « actif » et pour toutes ses données d'entrées obligatoires, il existe des valeurs disponibles. Ces conditions assurent que l'activité qui anticipe a tous ses éléments d'entrée disponibles et ses prédécesseurs ont déjà commencé à travailler. Cette stratégie ajoute à la stratégie précédente des contraintes supplémentaires concernant la disponibilité des données d'entrée. Dans ce cas-ci, l'activité de modification pourrait commencer seulement avec un brouillon du document et une version incomplète des commentaires. L'acteur en charge de l'activité « Modification » a l'occasion de voir ce qu'il devra faire et de commencer à traiter quelques commentaires. Dans ce cas-ci, une synchronisation devra être faite avec la version finale du document. Cette stratégie garantit la disponibilité des ressources, même si elles ne sont pas dans l'état final.

Ces stratégies permettent de choisir le degré de flexibilité adéquat pour un procédé.

Dans la partie suivante, nous décrivons le calcul des états des activités de manière formelle. Dans un premier temps nous définissons l'application qui donne l'état d'une activité à n'importe quel moment. Dans un deuxième temps, chaque état est défini.

4.3.2 États d'une activité

A un moment donné, chaque activité d'une instance de procédé est dans un état bien défini. L'ensemble de tous les états pertinents pour la navigation est noté S. Naviguer dans un modèle de procédé signifie assigner un état à chaque activité, en concordance avec la sémantique opérationnelle du métamodèle. Les états pertinents pour la navigation inclus dans S, sont les suivants :

- Initial – Quand une instance du procédé est créée, cet état est assigné à chaque activité.
- Prêt à anticiper – Cet état indique que l'activité peut commencer à anticiper.
- Anticipant – L'état dans lequel une activité entre quand un acteur démarre l'implantation associée à une activité qui était dans l'état « prêt a anticiper ».
- Exécutable – Une activité passe dans cet état la première fois que toutes ses conditions pour son exécution deviennent vraies.
- Actif – L'état dans lequel une activité entre quand un acteur démarre l'implantation associée à l'activité.
- Suspendu – Une activité entre dans cet état quand l'acteur qui exécute l'unité de travail associée arrête son exécution et la condition de sortie associée n'est pas encore satisfaite.
- Fini – Une activité entre dans cet état quand sa condition de sortie est satisfaite.
- Mort – Une activité entre dans cet état si elle est atteinte par le processus d'élimination du chemin mort.

A tout moment, l'état d'une activité est donné par l'application définie ci-dessous.

DÉFINITION 4.1 (APPLICATION « ÉTAT D'UN ACTIVITÉ »)
L'application « état d'une activité »

$$\omega : \mathbb{N} \times N \to S$$

associe à chaque moment $i \in \mathbb{N}$ et à chaque activité $A \in N$ son état $\omega(i, A)$. Pour chaque $i \in \mathbb{N}$, $\omega_i(A) := \omega(i, A)$ définit une application $\omega_i : N \to S$.

Les activités finies et mortes sont importantes pour la navigation. Donc, il est utile, pour la description de la navigation de définir des applications spéciales qui fournissent exactement l'ensemble de ces activités.

DÉFINITION 4.2 (APPLICATIONS « ACTIVITÉS FINIES ET MORTES »)
L'application « activités finies »

$$\lambda : \mathbb{N} \to \rho(N), i \mapsto {\omega_i}^{-1}(fini)$$

dérive à chaque moment $i \in \mathbb{N}$ l'ensemble de toutes les activités $\lambda(i) \subseteq N$ qui sont finies. ($\rho(N)$ est l'ensemble de parties de l'ensemble N, c.a.d. l'ensemble de tous les sous-ensembles de N)
La fonction d'activités mortes

$$\delta : \mathbb{N} \to \rho(N), i \mapsto {\omega_i}^{-1}(mort)$$

dérive à chaque moment $i \in \mathbb{N}$ l'ensemble de toutes les activités mortes $\delta(i) \subseteq N$. Nous utilisons les notations suivantes : $\lambda_i := \lambda(i)$ et $\delta_i := \delta(i)$

Nous définissons maintenant chaque état. Pour définir le moment où une activité devient exécutable, nous utilisons l'état du prédicat qui constitue la condition de démarrage de l'activité.

États des prédicats A chaque moment, chaque prédicat dans une instance de procédé a un état bien défini. Un prédicat peut avoir les états suivants : « évalué » ou « non-évalué », qui indique si sa valeur de vérité a été déjà déterminée ou pas encore. Nous étendons l'ensemble S pour inclure ces états. De la même manière que pour les états d'une activité, nous utilisons une application pour définir l'état d'un prédicat.

DÉFINITION 4.3 (APPLICATION D'ÉTAT D'UN PRÉDICAT)
L'application d'état d'un prédicat

$$\xi : \mathbb{N} \times \mathcal{C} \to S$$

associe à chaque moment $i \in \mathbb{N}$ et à chaque prédicat $p \in \mathcal{C}$, son état actuel $\xi(i, p)$.

L'application d'état d'un prédicat satisfait les conditions suivantes, qui représentent les règles pour le changement valide d'état des prédicats :

1. quand le procédé est instancié, tous les prédicats sont dans l'état non-évalué.
2. la condition de transition d'un connecteur de contrôle qui part d'une activité finie est dans l'état évalué.

3. la condition de transition d'un connecteur de contrôle qui part d'une activité morte est dans l'état évalué et la valeur de vérité est fausse.
4. les conditions de sorties des activités suspendues et finies sont dans l'état évalué.
5. une condition de jointure est évaluée si et seulement si tous ses prédicats constituants sont dans l'état évalué.
6. dans toutes les autres situations, un prédicat a l'état « non-évalué » et sa valeur de vérité n'est pas connue.(Nous considérons trois valeurs de vérité : vrai, faux et pas connu).

Calcul des instances des conteneurs Soit X une activité, un modèle de procédé ou un prédicat. A n'importe quel moment, le métamodèle associe des instances au conteneur d'entrée et de sortie de X. Quand une instance du modèle de procédé est créée, à chaque conteneur associé à l'instance, des instances par défaut y sont attribuées. Par la suite, les instances de conteneur sont calculées en exécutant les implantations des activités et en appliquant les connecteurs de données. Les définitions suivantes décrivent de manière plus précise le calcul des instances de conteneurs.

Définition 4.4 (Instance courante)
Soit $P \in \mathcal{P}$ un modèle de procédé, N son ensemble d'activités et $\mathcal{C}$, son ensemble de conditions. Pour $X \in N \cup \mathcal{P} \cup \mathcal{C}$ et une donnée $v \in i(X) \cup o(X)$, nous notons ${}^{i}v$ l'instance attribuée à v au moment i. De maniere similaire, nous notons ${}^{i}i(X)$ et ${}^{i}o(X)$ l'instance du conteneur d'entrée et de sortie du X, respectivement, au moment $i \in \mathbb{N}$.

Quand un modèle de procédé est instancié au moment $i = 0$, pour chaque conteneur, l'instance par défaut est calculée en attribuant des valeurs par défaut à ses données. Ceci est une représentation conceptuelle. Pour économiser l'espace de stockage, un système de workflow crée typiquement les instances des conteneurs à la demande et non à l'avance. Les valeurs par défaut sont utilisées à ce moment pour l'information manquante.

Définition 4.5 (Instance par défaut)
Soit $P \in \mathcal{P}$ un modèle de procédé ; N son ensemble d'activités et $\mathcal{C}$, son ensemble de conditions. Alors :
$\forall X \in N \cup \mathcal{P} \cup \mathcal{C}$
$\forall v \in i(X) \cup o(X) \exists w_{X,v} \in DOM(v) :^{0} v = w_{X,v}$

$DOM(v)$ est le domaine de l'élément de donnée v.
Maintenant, nous pouvons définir quand une activité devient exécutable.

Activités dans l'état « exécutable » Chaque modèle de procédé P a une ou plusieurs activités de démarrage, c.a.d. des activités qui n'ont pas d'arcs entrants. Nous notons l'ensemble de toutes les activités de démarrage N'.

$$A \in N' :\Leftrightarrow \{e \in E | \pi_2(e) = A\} = \phi$$

où π dénote l'application de projection des produits cartésiens et ses indices spécifient les composants sélectionnés du domaine de projection. (Une activité A est une activité de démarrage si dans l'ensemble de connecteurs E, il n'existe pas de connecteurs ayant A comme destination.)
Quand P est instancié, toutes ses activités de démarrage passent à l'état « exécutable ».

DÉFINITION 4.6 (ACTIVITÉS EXÉCUTABLES AU DÉMARRAGE DU PROCÉDÉ)
Une activité A devient exécutable au moment $0 \Leftrightarrow A \in N'$.

Pour les moments $i > 0$, les activités deviennent exécutables comme résultat d'une étape de navigation qui est déclenchée quand une activité passe à l'état « fini ». L'étape de navigation détermine tous les arcs qui sortent de l'activité finie et calcule la valeur de chaque condition de transition associée. Si le point d'arrivée d'un tel arc est un noeud régulier, l'activité correspondante devient exécutable si et seulement si la condition de transition associée est vraie, sinon, la condition de jointure du noeud d'arrivée doit être vraie. Ceci est exprimé dans la définition suivante.

DÉFINITION 4.7 (ACTIVITÉS DANS L'ÉTAT « EXÉCUTABLE »)
Soit $A \in N - N'$ une activité qui n'est pas de démarrage, A devient exécutable au moment $i, i > 0 \Leftrightarrow$

1. $\exists (X, A, p) \in E : \xi(i, p) =$ evalué $\wedge\, \xi(i-1, p) =$ non-evalué
2. Une des deux conditions suivantes est vraie :
   - $A \in N_\bullet \wedge p({}^i i(p)) = 1$ /* noeud régulier */
   - $A \in N_* \wedge \phi(A)({}^i i(q) | q \in C^{\leftarrow}(A)) = 1$ /* noeud de jointure */
3. $\omega(A, i-1) \in$ {initial, prêt à anticiper}

Le seul événement qui change l'état d'une condition de transition de l'état « non-évalué » à l'état « évalué » est la terminaison d'une activité et la procédure d'élimination du chemin mort (qui est déclenchée aussi par la terminaison d'une activité). En conséquence, l'instanciation du procédé et la terminaison des activités sont les deux seuls événements qui font qu'une activité devienne exécutable.

Propriété :
$A \in N$ devient exécutable au moment $i \Rightarrow \omega(i, A) = executable$

Activités dans l'état « prêt à anticiper » Nous considérerons par la suite la stratégie dictée par le flot de contrôle. Une activité passe à l'état « prêt à anticiper » dès que ses prédécesseurs sont dans l'état « exécutable » ou « anticipant ».

DÉFINITION 4.8 (ACTIVITÉS EN ÉTAT « PRÊT À ANTICIPER » (STRATÉGIE FLOT DE CONTRÔLE)
Une activité A passe à l'état « prêt à anticiper » au moment $i \Leftrightarrow$

1. $\omega(A, i-1) = initial$
2. $\forall N \in A^{\leftarrow}, \omega(N, i) \in$ {actif,anticipant} si A est un noeud de jointure de type AND-Join ou un noeud régulier
3. $\exists N \in A^{\leftarrow}, w(N, i) \in$ {actif,anticipant} si A est un noeud de type OR-Join.

Une stratégie encore plus stricte est la stratégie d'anticipation qui demande que le prédécesseur soit dans l'état « actif ».

L'événement qui fait qu'une activité devient prête à anticiper dans le cas de la stratégie dictée par le flot de contrôle est le démarrage d'un prédécesseur.

Nous donnons maintenant la définition de l'état « suspendu » et « fini ».

Activités suspendues Avant qu'une activité puisse passer dans l'état « suspendu », elle doit avoir été dans l'état « actif » ou « anticipant ». Une activité qui ne satisfait pas sa condition de sortie quand son implantation termine, entre dans l'état « suspendu ». L'activité n'est pas terminée avec succès ; elle doit être continuée ultérieurement et la navigation ne peut pas passer à l'activité suivante. La définition suivante décrit formellement les conditions pour cet état.

DÉFINITION 4.9 (ACTIVITÉS SUSPENDUES)
Soit $A \in N$ une activité avec l'implantation $\Psi(A)$. Alors, $\omega(i, A) = suspendu \Leftrightarrow$

1. $\exists j \in \mathbb{N} : j < i \wedge \omega(i, A) \in \{\text{actif, anticipant}\}$
2. $\exists k \in \mathbb{N} : j < k \leq i \wedge \Psi(A)$ est terminé au moment k
3. $\forall k \leq r \leq i : \varepsilon(A)({}^{r}i(\varepsilon(A))) \neq 1$

Quand l'activité passe à l'état « suspendu » en venant de l'état « anticipant », la condition de sortie n'est pas évaluée, donc sa valeur de vérité est « non-connu ».

Activités finies De même que pour les activités suspendues, l'implantation d'une activité finie doit avoir été activée à un moment précédent. Pour passer à l'état « fini », par opposition à l'état « suspendu », la condition de sortie doit être satisfaite.

DÉFINITION 4.10 (ACTIVITÉS FINIES)
Soit $A \in N$ une activité avec l'implantation $\Psi(A)$. Alors, $A \in \lambda_i \Leftrightarrow$

1. $\exists j \in \mathbb{N} : j < i \wedge \omega(i, A) = \text{actif},$
2. $\exists k \in \mathbb{N} : j < k \leq i \wedge \Psi(A)$ est terminé au moment k
3. $\varepsilon(A)({}^{k}i(\varepsilon(A))) = 1$

Nous remarquons que l'état « fini » peut être atteint seulement de l'état « actif ». Même si une activité a anticipé, elle doit passer par l'état « actif » avant de finir.

4.3.3 Navigation - gestion des bons de travail

Dans cette partie nous décrirons l'attribution des activités aux acteurs. Dans un premier temps, le système de workflow doit déterminer les activités qui peuvent être sélectionnées pour l'exécution. Dans un deuxième temps, ces activités doivent être attribuées aux acteurs conformément à la requête de personnel. Seulement un de ces acteurs doit exécuter le travail correspondant.

Sélectionner les activités pour exécution et anticipation Dans la définition 4.7 nous avons défini le moment exact où une activité devient exécutable. La définition suivante nous permet de faire référence à ce moment :

DÉFINITION 4.11 (APPLICATION EXÉCUTABLE)
La fonction

$$\eta : \mathbb{N} \times N \to \{0, 1\}$$

avec
$\eta(i, A) = 1 :\Leftrightarrow$ A devient exécutable au moment i est appelée *application exécutable*. L'ensemble

$$\eta_i := \{A \in N | \eta(i, A) = 1\}$$

est appelé *l'ensemble des activités qui deviennent exécutables au moment i*. La fonction $\tau : N \to \mathbb{N}$ attribue à chaque activité le moment du temps où elle devient exécutable :

$$\tau(A) = i :\Leftrightarrow A \in \eta_i$$

.

En se basant sur cette définition, la fonction d'état doit avoir la propriété suivante pour $i > 0$:

$$A \in \eta_i \Rightarrow \omega_i(A) = executable \wedge \omega_{i-1}(A) \neq executable$$

Une activité devient exécutable à un moment précis, unique de temps. Ceci s'exprime par : $A \in \eta_i \Rightarrow \forall j \neq i : A \notin \eta_j$.

Nous définissons de la même manière, l'ensemble des activités prêtes à anticiper récemment.

Dans la définition 4.8 nous avons défini le moment exact où une activité devient prête à anticiper. La définition suivante nous permet de faire référence à ce moment :

DÉFINITION 4.12 (APPLICATION D'ANTICIPATION)
La fonction

$$\eta^a : \mathbb{N} \times N \to \{0, 1\}$$

avec
$\eta^a(i, A) = 1 :\Leftrightarrow$ A devient prête à anticiper au moment i est appelée *application d'anticipation*. L'ensemble

$$\eta^a_i := \{A \in N | \eta^a(i, A) = 1\}$$

est appelé *l'ensemble des activités qui deviennent prêtes à anticiper au moment i*. La fonction $\tau^a : N \times \mathbb{N}$ attribue à chaque activité le moment du temps où elle devient prête à anticiper :

$$\tau^a(A) = i :\Leftrightarrow A \in \eta^a_i$$

.

En se basant sur cette définition, la fonction d'états doit avoir la propriété suivante pour $i > 0$:

$$A \in \eta^a_i \Rightarrow \omega_i(A) = \text{prêt à anticiper} \wedge \omega_{i-1}(A) \neq \text{prêt à anticiper}$$

Une activité devient prête à anticiper à un moment précis, unique de temps. Ceci s'exprime par :

$A \in \eta^a_i \Rightarrow \forall j \neq i : A \notin \eta^a_j$.

En pratique, cet ensemble est calculé au moment où une activité est démarrée. Considérons qu'au moment i, un acteur démarre l'activité A, qui entre dans l'état « actif » ou « anticipant ».

Nous calculons l'ensemble de ses successeurs qui sont dans l'état «initial». L'ensemble η_i^a des activités qui deviennent prêtes à anticiper au moment i est exactement l'ensemble construit ci-dessus.

Gestion des bons de travail Chaque fois qu'une activité devient exécutable ou prête à anticiper, elle est attribuée aux acteurs qui satisfont la requête de personnel[14] associée à l'activité. Chacun de ces acteurs est qualifié pour exécuter l'activité. La paire, constituée d'une activité et de l'acteur qui a le droit d'exécuter l'activité, est appelée *bon de travail*. A n'importe quel moment, le WFMS doit dériver des bons de travail à partir des activités qui deviennent exécutables et prêtes à anticiper.

La requête de personnel associée à l'activité est notée $\Omega(A)$. Quand elle est évaluée au moment i, la requête de personnel retourne un ensemble d'acteurs $\Omega(A)(i)$.

Nous allons définir l'ensemble de bons de travail programmés pour anticipation et pour exécution.

Définition 4.13 (Bon De Travail programmés)
L'ensemble

$$\gamma_i := \{(A, a) | A \in \eta_i \wedge a \in \Omega(A)(i)\}$$

est appelé *l'ensemble de bons de travail programmés au moment i pour exécution.* Il est constitué de tous les bons de travail obtenus à partir de toutes les activités qui deviennent exécutables au moment i.
De la même manière, l'ensemble

$$\gamma_i^a := \{(A, a) | A \in \eta_i^a \wedge a \in \Omega(A)(i)\}$$

est appelé *l'ensemble de bons de travail programmés au moment i pour anticipation.* Il est constitué de tous les bons de travail obtenus à partir de toutes les activités qui deviennent prêtes à anticiper au moment i.

Les acteurs peuvent exécuter leur travail à n'importe quel moment après la programmation des bons de travail. Quand un certain acteur commence l'activité correspondant au bon de travail, tous les autres bons de travail créés pour la même activité doivent disparaître des listes de tâches des autres acteurs. Autrement, plus d'un agent pourrait exécuter le même travail, ce qui conduirait à une perte de temps et de ressources. Pour refléter ce comportement opérationnel, nous introduisons des constructions supplémentaires dans le metamodele.

[14] la requête de personnel (défnie dans la section 4.2.3) détermine l'ensemble d'acteurs qui peuvent exécuter l'activité

DÉFINITION 4.14 (APPLICATION « ACTEUR »)
L'application partielle

$$\alpha : \mathbb{N} \times N \to \mathcal{A}$$

qui satisfait les contraintes suivantes :

1. α n'est pas définie pour $i < \tau^a(A)$
2. $\tau^a(A) < i \leq \tau(A) \Rightarrow \alpha(i, A) \in \Omega(A)(\tau^a(A))$,
3. $i \geq \tau(A) \Rightarrow \alpha(i, A) \in \Omega(A)(\tau^a(A)) \vee \alpha(i, A) \in \Omega(A)(\tau(A))$

est appelée *l'application « agent »*.

Elle associe à chaque activité l'acteur qui l'a activée, l'activité étant soit dans l'état « prêt à anticiper », soit « exécutable ». Si un acteur a commencé l'unité de travail associé à un bon de travail en état « prêt à anticiper », il va continuer à l'exécuter quand l'activité entre dans l'état « actif ». Si l'activité n'a pas été anticipée, alors sa requête de personnel est évaluée de nouveau quand l'activité devient exécutable.

L'application de l'acteur d'activation doit avoir la propriété suivante :

$$\forall A \in N : \omega(i, A) = \text{active} \vee \omega(i, A) = \text{anticipant} \Leftrightarrow \alpha(i, A)\text{est définie.}$$

La note suivante est une conséquence directe de cette propriété :

$$\forall A \in N \forall i \in \mathbb{N} \exists j < i : \omega(i, A) = \text{actif} \Rightarrow \omega(j, A) = \text{exécutable.}$$

Maintenant, nous pouvons décrire comment les bons de travail sont gérés par le WFMS.

DÉFINITION 4.15 (FAMILLE DE BONS DE TRAVAIL)
L'ensemble de tous les bons de travail (pour l'anticipation ou exécution) constitue une suite d'ensembles, appelée *famille de bons de travail*. Nous notons :
– $(\Gamma_i)_{i \in \mathbb{N}}$ la famille de bons de travail pour exécution,
– $(\Gamma_i^a)_{i \in \mathbb{N}}$ la famille de bons de travail pour anticipation.

1. $\Gamma_0 := \gamma_0$
   $\Gamma_0^a := \gamma_0^a$
2. $\Gamma_i := (\Gamma_{i-1} \cup \gamma_i) - \{(A, a) | a \neq \alpha(i, A)\} - \{(A, a) | a \in \xi_i - \xi_{i-1} - \sigma_i^a\} - \{(A, a) | a \in \lambda_i - \lambda_{i-1}\}$, pour $i > 0$
   $\Gamma_i^a := (\Gamma_{i-1}^a \cup \gamma_i^a) - \{(A, a) | a \neq \alpha(i, A)\} - \{(A, a) | a \in \xi_i - \xi_{i-1}\} - \{(A, a) | a \in \delta_i - \delta_{i-1}\}$, pour $i > 0$

Ainsi, à n'importe quel moment, l'ensemble de tous les bons de travail pour anticipation est constitué de :

- tous les bons de travail qui ont été programmés pour anticipation à ce moment (l'ensemble γ_i^a),
- tous les anciens bons de travail, (l'ensemble Γ_{i-1}^a), mais :
  - quand une activité est démarrée en état « prêt à anticiper » (passe dans l'état « anticipant »), tous les bons de travail qui ont résulté par l'attribution de la même activité aux autres acteurs sont supprimés (l'ensemble $\{(A, a) | a \neq \alpha(i, A)\}$),

- quand une activité devient exécutable, tous les bons de travail sont supprimés ($\{(A,a)|a \in \xi_i - \xi_{i-1}\}$) (la requête du personnel sera réévaluée et les nouveaux bons de travail vont être calculés comme bons de travail pour exécution),
- quand une activité passe à l'état « mort », tous les bons de travail sont supprimés.

De la même manière, à n'importe quel moment, l'ensemble de tous les bons de travail pour exécution est constitué de :

- tous les bons de travail qui ont été programmés pour exécution à ce moment (l'ensemble γ_i)
- tous les anciens bons de travail (l'ensemble Γ^a_{i-1}), mais
  - quand une activité est activée pour exécution, tous les bons de travail qui ont résulté par l'attribution de la même activité aux autres acteurs sont supprimés (l'ensemble $\{(A,a)|a \neq \alpha(i,A)\}$),
  - quand une activité est finie, le bon de travail correspondant est supprimé ($\{(A,a)|a \in \delta_i - \delta_{i-1} -\}$) (Nous remarquons que, conformément au point précédent, après l'activation de l'activité, un seul bon de travail correspondant à cette activité reste).

4.3.4 Navigation - description informelle

La navigation s'effectue chaque fois qu'une activité démarre son exécution ou est finie. Quand une activité est finie (passe à l'état « fini »), l'étape de navigation se déroule comme dans un modèle classique :

- déterminer tous les arcs de contrôle qui partent de l'activité finie ;
- déterminer tous les points d'arrivée de ces connecteurs en calculant les successeurs de l'activité ;
- déterminer les conditions de transition associées à ces connecteurs de contrôle ;
- calculer les instances courantes des conteneurs d'entrée de ces conditions de transition ;
- calculer la valeur de vérité des conditions de transition en se basant sur les instances courantes des conteneurs d'entrée ;
- déterminer parmi les successeurs, tous les noeuds réguliers dont la condition de transition entrante a été évaluée comme fausse ; déterminer tous les noeuds de jointure parmi les successeurs dont la condition de jointure est évaluée comme fausse (l'ensemble de noeuds morts).
  - effectuer la procédure d'élimination du chemin mort (voir section 4.3.2) ;
  - une activité située sur un chemin mort ayant une condition de jointure qui est évaluée à vrai, passe à l'état « exécutable ».
- déterminer parmi les successeurs tous les noeuds réguliers dont la condition de transition a été évaluée comme vraie et les noeuds de jointure dont la condition de jointure est évaluée comme vraie. Toutes les activités de cet ensemble calculé qui étaient dans l'état « prêt à anticiper » passent à l'état « exécutable ». Les activités qui étaient dans l'état « anticipant » passent à l'état « actif ».
- Tous les autres successeurs restent dans leur état courant et doivent attendre une étape de navigation future (provoquée par la terminaison d'une activité) pour un possible changement d'état.

Quand une activité démarre son exécution (passe à l'état « actif » ou « anticipant ») :

- tous les successeurs de cette activité sont calculés ;
- les successeurs qui sont dans l'état « initial » sont déterminés. Toutes les activités de cet ensemble qui sont de type noeud régulier ou de jonction (OR-Join) passent à l'état « prêt à anticiper ». Les activités de type rendez-vous (AND-Join) passent à l'état « prêt a anticiper »

seulement si tous leurs prédécesseurs ont été démarrés.

Pour toutes les activités qui passent dans l'état « exécutable » ou « prêt à anticiper » :

- la requête de personnel associée est évaluée ;
- pour chaque acteur, le bon de travail correspondant est créé.

4.3.5 Modification du flot de données pour supporter les résultats intermédiaires

Dans les sections précédentes nous avons mis en évidence les modifications qui doivent être apportées au modèle d'exécution pour permettre l'anticipation. Pour profiter de tous les avantages de l'anticipation, il est nécessaire de permettre aussi la circulation anticipée des données entre activités : la publication de données intermédiaires dans le conteneur de sortie d'une activité et leur transmission conforme aux connecteurs de données vers les activités qui anticipent.

Comme nous considérons principalement des activités interactives, l'utilisateur peut décider de fournir des données de sortie avant la fin de l'activité et éventuellement avec plusieurs versions successives. Nous avons introduit de nouvelles opérations pour une activité , *Écrire* et *Lire*. Ces opérations peuvent être utilisées par des utilisateurs (ou même des outils spéciaux) pour contrôler la publication des données pendant l'exécution de l'activité. L'opération *Écrire* permet la mise à jour d'une donnée de sortie et la rend disponible aux activités suivantes.

Deux modes de circulation de données entre activités sont possibles : le mode distribution[15] et le mode libre service[16]. Dans le mode distribution, le conteneur d'entrée d'une activité qui anticipe est mis à jour automatiquement chaque fois qu'une nouvelle valeur est produite pour l'élément de donnée qui constitue la source du connecteur de données. En mode libre service, l'activité est seulement notifiée et la mise à jour s'effectue avec l'opération *Lire*.

Le modèle de procédé présenté dans la figure 4.5 contient l'activité C qui est implantée par un autre modèle de procédé, contenant les activités A et B. Supposons que l'activité B ait une dépendance de données avec l'activité A et un connecteur de donnée existe entre A et B $(aout, bin) \in \Delta(A, B)$. De manière similaire, il y a une dépendance de donnée entre C et D dans le procédé principal, avec $(cout, din) \in \Delta(C, D)$. Les données circulent aussi entre les activités incluses en C et le conteneur de sortie de C. Par exemple, la donnée *aout* du conteneur de sortie de A est copiée dans la donnée *cout* du conteneur de sortie de l'activité procédé C.

Supposons que l'activité A appelle l'opération $Ecrire(aout)$ pour écrire la donnée de sortie *aout*. Dans la définition du flot des données, deux arcs existent qui ont *aout* comme origine : un arc $(aout, bin)$ entre l'activité A et B et un autre, $(aout, cout)$ entre l'activité A et le conteneur de sortie du procédé. Si B est en état « anticipant », elle doit être notifiée au sujet de l'existence de nouvelles données (mode libre-service) ou être notifiée de l'arrivée des données dans son conteneur d'entrée (mode distribution). De même pour le deuxième connecteur de données : la donnée sera écrite dans le conteneur de sortie du procédé. Comme un autre connecteur existe entre l'activité C et D $(cout, din)$, alors, de nouveau, si D anticipe, elle sera notifiée ou la donnée sera copiée dans le conteneur d'entrée de D. De cette manière, l'échange de données anticipé se réalise entre les activités d'un procédé et celles du sous procédé.

Calcul des instances des conteneurs en présence de l'anticipation Dans le modèle traditionnel, le conteneur de sortie d'une activité est calculé au moment de la terminaison de l'activité ; il est retourné par l'implantation de l'activité. Le conteneur d'entrée est calculé au moment de l'activation de l'activité en appliquant les connecteurs de données.

[15] push
[16] pull

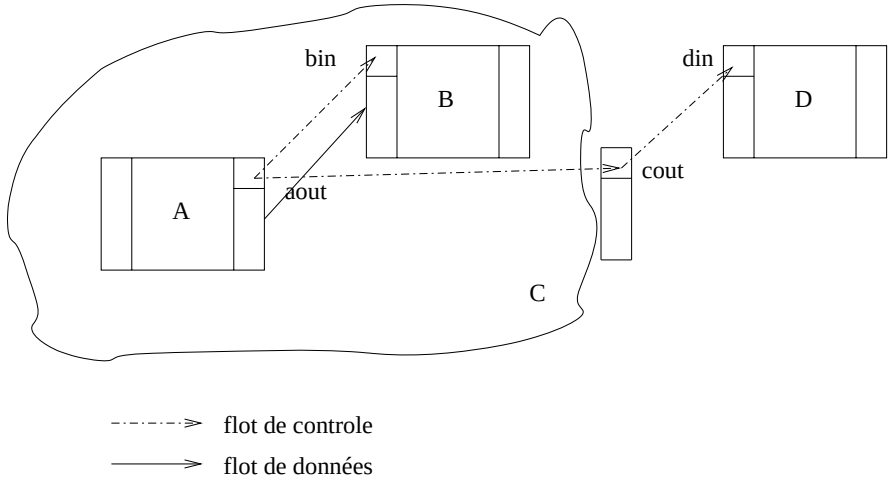


FIG. 4.5 – Opération *Ecrire*(*aout*)

Pour bénéficier de l'anticipation, une activité dans notre modèle peut remplir graduellement son conteneur de sortie et le transmettre au système de workflow.

Nous définissons l'historique du procédé.

DÉFINITION 4.16 (HISTORIQUE DU PROCÉDÉ)
L'historique du procédé est l'ensemble préservant l'ordre des actions (écritures et lectures) sur les données du workflow par l'ensemble de ses activités. Il est noté $H = \{< X, a, o, i >\}$. Chaque étape dans l'historique $< X, a, o, i >$ représentent l'action a (écrire ou lire) par l'activité X sur l'objet o au moment i.

Nous pouvons définir aussi la version d'une donnée de sortie w d'une activité au moment i $V({}^{i}w)$ comme un entier égal au nombre d'opérations d'écriture effectuées par l'activité jusqu'au moment i. Formellement :

DÉFINITION 4.17 (VERSION D'UNE DONNÉE DE SORTIE D'UNE ACTIVITÉ)
Soit w une donnée du conteneur de sortie de l'activité A. Au moment 0, w a la version zéro ($V({}^{0}w) = 0$). Au moment i la version est :

$$V({}^{i}w) = |\{< X_j, \text{écrire}, d_j, j >\in H | j < i \text{ et } d_j = w \text{ et } X_j = A\}|$$

Supposons qu'une activité A soit activée au moment $j \in \mathbb{N}, j > 0$. L'implantation de l'activité $\Psi(A)$ est invoquée, et l'instance courante ${}^{j}i(A) = ({}^{j}v_1, \ldots, {}^{j}v_k)$ du conteneur d'entrée $i(A) = (v_1, \ldots, v_k)$ de A est passée à $\Psi(A)$. Considérons que $\Psi(A)$ termine au moment $i > j$. L'instance du conteneur de sortie de A, retourné par $\Psi(A)$ peut être définie comme $\Psi(A({}^{i-1}v_1, \ldots, {}^{i-1}v_k))$. Dés qu'une activité passe à l'état « actif », son conteneur d'entrée ne change plus ; par contre, il peut changer si l'activité est dans l'état « anticipant ». Si on ne prend en compte que les valeurs finales (activité comme boîte noire, voir figure 4.6), on peut dire que le conteneur de sortie est une fonction du conteneur d'entrée.

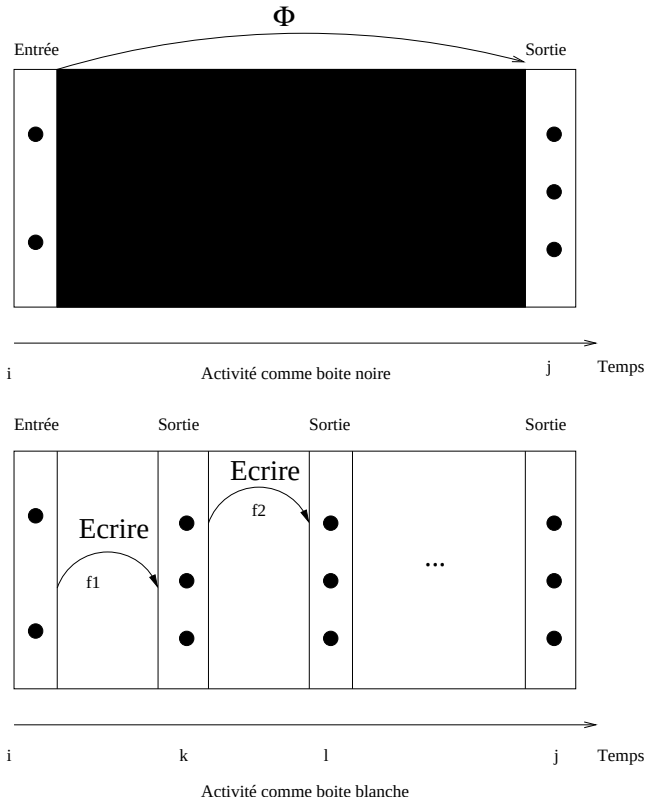


FIG. 4.6 – Activité boîte noire et boîte blanche

DÉFINITION 4.18 (INSTANCE DU CONTENEUR DE SORTIE)
Soit $A \in N$ une activité, $i(A) = (v_1, \ldots, v_k)$ son conteneur d'entrée, $o(A) = (w_1, \ldots, w_r)$ son conteneur de sortie et $\Psi(A)$ son implantation. Alors :
- $\omega(i, A) \in \{\text{terminé, fini}\} \Rightarrow^i o(A) = \Psi(A)(^{i-1}v_1, \ldots, ^{i-1}v_k)$
- $\omega(i, A) \in \{\text{initial, mort, pret à anticiper, executable}\} \Rightarrow^i o(A) = (^0w_1, \ldots, ^0w_r)$
- $\omega(i, A) \in \{\text{actif, anticipant}\} \Rightarrow^i o(A) = (^iw_1, \ldots, ^iw_r)$ où la valeur de la donnée de sortie w_k au moment i est calculée de la manière suivante.
  Si $V(^iw_k) = 0 \Rightarrow^i w_k =^0 w_k$ sinon, soit m le moment de la dernière écriture : $m = \max_j\{< X_j, ecrire, d_j, j > \in H | j \leq i \text{ et } d_j = w_k \text{ et } X_j = A\}$.
  Alors, la valeur du ^iw_k est :

$$^iw_k =^m w_k = f_j(A)((^{m-1}v_1, \ldots, ^{m-1}v_k), ^{m-1}w_k)$$

  où $j = V(^iw_k)$.

(^mw_k a été produit à partir de l'instance du conteneur d'entrée au moment $m-1$: $(^{m-1}v_1, \ldots, ^{m-1}v_k)$) et de la version antérieure de w_k : $^{m-1}w_k$).

Donc, au moment m, l'activité A a effectué une opération « Ecrire » sur w_k. Cette opération peut être formalisée comme une fonction f_j (la j-ième dans une suite de fonctions $f_1(A) \ldots f_m(A)$) qui appliquée au conteneur d'entrée et à la version antérieure de w_k produit une autre version de w_k.

Quand une activité est démarrée, son conteneur d'entrée est calculé en se basant sur les

connecteurs de données, conteneurs de sortie des activités précédentes et les données d'entrée du procédé. Les conteneurs d'entrée des conditions sont calculés de la même manière. Les données du conteneur matérialisé reçoivent des copies des instances courantes des sources des connecteurs de données. Les données qui ne sont pas la destination d'une application de données, gardent leurs valeurs par défaut. Ceci est exprimé dans la définition suivante. Chaque fois qu'une opération de lecture est exécutée (le mode libre service), la donnée reçoit une copie de la donnée source du connecteur de données.

DÉFINITION 4.19 (APPLIQUER LES CONNECTEURS DE DONNÉES)
- $\forall X \in N \cup \mathcal{C}, \forall v \in i(X) : (w, v) \in \Delta(A, X) \cup \vec{\Delta}(X)$
  - mode distribution $\Rightarrow^i v =^{i-1} w$
  - mode libre service
    Soit m le moment de la dernière lecture : $m = \max_j\{< X_j, read, d_j, j > \in H | j \leq i$ et $d_j = v$ et $X_j = X\} \Rightarrow^i v =^m v =^{m-1} w$
- $\forall P \in \mathcal{P} \forall v \in o(P) : (w, v) \in \vec{\Delta}(X) \Rightarrow^i v =^{i-1} w$
- $\forall X \in N \cup \mathcal{C} \cup \{P\} \forall v \in i(X) \cup o(P) : \{(w, v) | A \in N \wedge (w, v) \in \Delta(A, X) \cup \vec{\Delta}(X)\} = \phi \Rightarrow^i v =^0 v$

La première condition décrit l'application d'un connecteur de données qui a comme destination un élément du récipient d'entrée d'une activité pour les deux modes d'échanges de données possibles. Pour le mode distribution, si une activité anticipe et, si pour une de ses données d'entrée, une version est publiée, sa donnée d'entrée sera automatiquement mise à jour. Pour le mode libre service, la valeur correspond à la dernière valeur lue par une opération *Lire*.

La deuxième condition décrit comment est appliqué un connecteur de donnée qui lie un élément de donnée de sortie d'une activité avec un élément de donnée du conteneur de sortie du procédé. Le mode d'échange de données, indépendamment de celui choisi pour l'échange entre activités, est le mode distribution.

La troisième condition spécifie que pour les données qui ne sont pas la destination d'un connecteur de données, les valeurs par défaut sont utilisées.

Nous pouvons définir maintenant les conditions nécessaires pour qu'une activité passe à l'état « prêt à anticiper » pour la stratégie dictée par le flot de données :

DÉFINITION 4.20 (ACTIVITÉS EN ÉTAT « PRÊT À ANTICIPER » (STRATÉGIE FLOT DE DONNÉES))
Une activité A passe à l'état « prêt à anticiper » au moment $i \Leftrightarrow$

1. $\omega(A, i-1) = initial$
2. $\forall v \in i(X) : (w, v) \in \Delta(A, X) \Rightarrow V(^i w) > 0$ et $\exists v \in i(X) : (w, v) \in \Delta(A, X) \Rightarrow V(^{i-1} w) = 0$

L'activité A passe à l'état « prêt à anticiper » quand pour tous ses éléments de données d'entrée il existe des valeurs intermédiaires produites par les activités sources.

Exemple Considérons un exemple pour illustrer la circulation des résultats intermédiaires. L'exemple montre également comment l'anticipation et la circulation des données en avance peuvent conduire à augmenter le parallélisme entre les activités d'un procédé et les activités de son sous-procédé.

Pour illustrer ceci, nous pouvons utiliser l'exemple introduit dans la section 2 et repris dans la figure 4.7 ; il représente un procédé typique pour la gestion de la livraison des produits par une

compagnie de vente au détail. L'activité « Produit » gère la commande courante ; elle peut être implantée comme un sous- procédé qui vérifie si l'élément est disponible, sinon il est commandé ; finalement une facture est produite. Les données de sortie du sous-procédé sont l'entrepôt où le produit est disponible et la date de livraison. Dès que le produit sera en stock ou que la date où il sera reçu du producteur est connue, l'accusé de réception au client peut être préparé. De même, la préparation de l'expédition pourra être lancée dès qu'on connaîtra l'entrepôt qui livrera le produit et la date où il sera disponible.

Pour ce type de procédé, l'anticipation dictée par le flot de contrôle et l'anticipation dictée par le flot de données peuvent être appliquées efficacement. Cependant, l'activité d'expédition ne peut pas être anticipée. C'est une activité typique qui peut être exécutée seulement quand toutes les activités précédentes sont terminées.

Les données sont remplies dans le conteneur de sortie du sous-procédé dès que les activités correspondantes les produisent ; les activités successives dans le procédé supérieur peuvent passer à l'état « prêt à anticiper ».

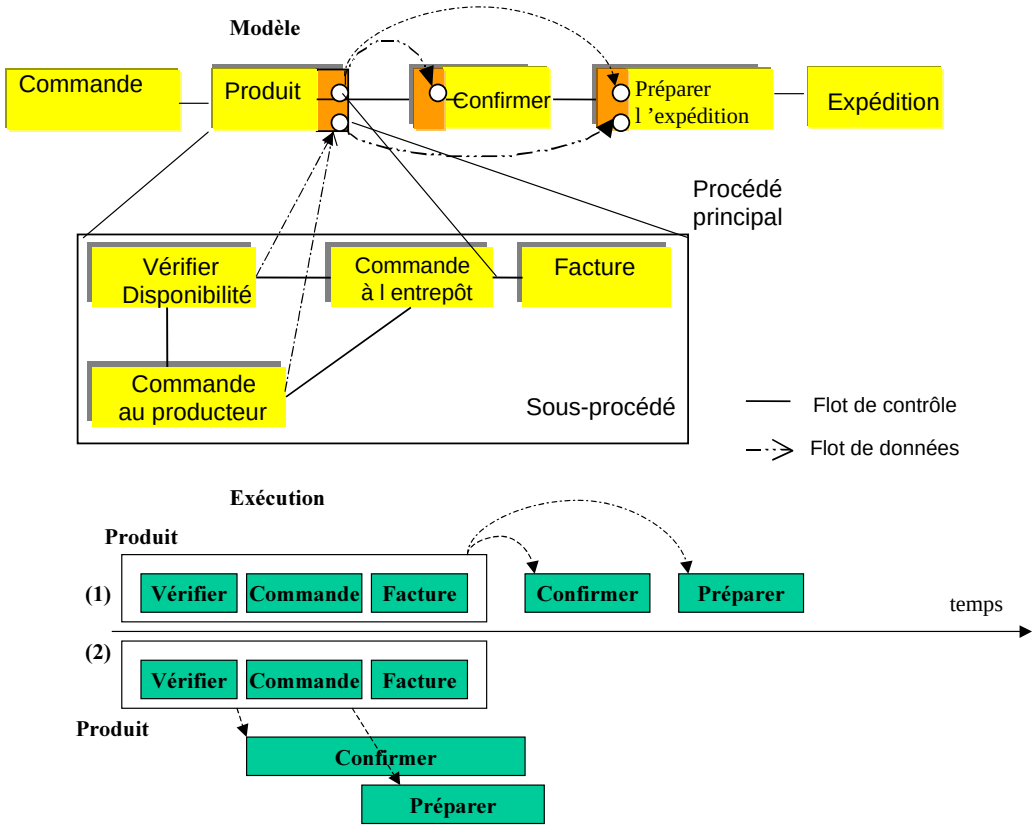


FIG. 4.7 – Echange de données entre procédés

En permettant l'externalisation des résultats intermédiaires, les activités ne sont plus isolées dans des transactions classiques. Ce ne sont plus de boîtes noires. Elles peuvent fournir des résultats intermédiaires qui peuvent être utilisés par les activités suivantes. Si celles-ci sont des activités interactives, les acteurs en charge peuvent considérer de prendre en compte ces nouvelles valeurs ou peuvent choisir d'attendre des valeurs plus stables censées arriver à un moment ulté-

rieur. La rupture de l'isolation des activités est nécessaire afin de tirer bénéfice de l'anticipation, mais pose également des problèmes de maintien de la cohérence. Ces problèmes et le moyen de les résoudre sont décrits dans la section 4.3.6.

Stratégies d'anticipation et condition de démarrage Les stratégies d'anticipation présentées dans la section 4.3.1 prennent en compte l'état des activités précédentes et la disponibilité des données, mais elles ne prennent pas en compte la condition de démarrage d'une activité.

Le fait qu'une activité soit exécutée ou pas dans l'instance courante du procédé dépend de la condition de transition ou de jointure. Cette condition est évaluée dans le modèle classique, quand toutes les activités précédentes sont finies (ou sont devenues mortes) (comme expliqué dans la section 4.3.4).

Nous remarquons que la plupart des activités dans un procédé sont des noeuds réguliers et ont la condition de transition 1 (toujours vraie). Les conditions de transition qui contiennent des prédicats dépendent d'habitude des données produites par l'activité précédente. Donc, si on attend leur évaluation basée sur des résultats finaux, l'anticipation n'est pas possible. Mais, en réalité on retrouve souvent des cas où les décisions sont soit prises en se basant sur des résultats intermédiaires soit sur les connaissances des participants résultant de leur expérience dans des exécutions passées du procédé. Par exemple, dans un procédé de préparation d'un voyage pour une conférence, dans 90% des cas, le chef de l'équipe accepte. Il est donc possible d'anticiper sur cette décision.

Nous discutons des conditions supplémentaires pour l'anticipation, qui peuvent être ajoutées aux stratégies déjà définies et qui prennent en compte l'évaluation de la condition de démarrage.

Nous introduisons un état supplémentaire pour l'état d'un prédicat : « anticipant », en dehors des états définis dans la partie 4.3.2 (« évalué » ou « non-évalué »).

- la condition de transition d'un connecteur qui sort d'une activité qui est dans l'état « actif » ou « anticipant », est « anticipant », si pour tous les paramètres d'entrée il existe des valeurs disponibles ;
- une condition de jointure est dans l'état « anticipant » si tous ses prédicats constituants sont dans l'état « évalué » ou « anticipant », dont au moins un est dans l'état « anticipant ».

Par rapport à la condition de démarrage, on a les possibilités suivantes :

- permettre l'anticipation seulement si la condition de démarrage est 1 (toujours vraie). Cette stratégie élimine le risque de travail perdu.
- permettre l'anticipation aussi pour les activités dont la condition de démarrage n'est pas triviale (1). La condition doit être dans l'état « anticipant » et la valeur intermédiaire de la condition égale à 1.
- permettre l'anticipation dans tous les cas et laisser à l'appréciation de l'utilisateur l'opportunité d'anticiper ou pas. Comme mesure de sécurité, le concepteur du procédé peut marquer plusieurs activités comme activités qui n'ont pas le droit d'anticiper. En plus, des statistiques ou des prédictions pourraient être utilisées pour apprécier la probabilité que l'activité soit exécutée dans l'instance courante du procédé (voir section 7.2.3).
- permettre l'anticipation dans tous les cas et imposer la condition 1 pour la publication des données. Dans ce cas, l'activité ne peut pas publier des données, mais les actions en dehors du système de workflow ne peuvent pas être contrôlées.

4.3.6 Synchronisation et recouvrement

La publication des résultats intermédiaires est importante pour tirer profit de l'anticipation. Cependant, une activité qui a publié des résultats pendant son exécution peut échouer ou pu-

blier un autre résultat, après qu'elle ait édité un résultat intermédiaire. Le problème est alors comment compenser le travail réalisé avec ces résultats. Cette situation peut être comparée aux lectures impropres dans les transactions traditionnelles. Afin d'assurer ceci, les règles suivantes sont appliquées :

Synchronisation Une activité qui a lu un résultat intermédiaire doit lire le résultat final correspondant. Cette règle est assurée dans le cas de mode d'échange « distribution ». Une activité peut passer à l'état « fini » seulement après avoir passé par l'état « actif ». Ainsi, une activité qui anticipe, à un moment donné passe à l'état « actif ». À ce moment, les activités précédentes sont terminées et leurs résultats finaux disponibles. Quand l'activité passe à l'état « actif », son conteneur d'entrée est actualisé avec les versions finales de ses paramètres d'entrée.

En ce qui concerne le mode d'échange « libre service », deux possibilités existent : soit au moment où l'activité devient active, une mise à jour automatique des données d'entrée avec les versions finales des données qui constituent les sources des connecteurs de données correspondants est effectuée ; soit avant de terminer, nous vérifions que l'activité a lu les dernières valeurs pour ses données d'entrée.

DÉFINITION 4.21 (DONNÉES À JOUR)
Soit X une activité.
$\forall v \in i(X) : (w, v) \in \Delta(A, X) \cup \vec{\Delta}(X)$
Soit m le moment de la dernière lecture :

$$m = \max_j \{< X_j, read, d_j, j > \in H | j \leq i \text{ et } d_j = v \text{ et } X_j = X\}$$

soit n le moment de la dernière écriture :

$$n = \max_j \{< X_j, Ecrire, d_j, j > \in H | j \leq i \text{ et } d_j = w \text{ et } X_j = A\}$$

L'activité a lu la dernière valeur pour l'élément de donnée v si $m > n$.

Recouvrement Une activité qui anticipe fait la supposition qu'elle sera exécutée, ce qui n'est pas sûr. Elle peut passer à l'état « mort », soit parce que sa condition d'entrée (la condition de jointure ou la condition de transition de son arc entrant) est évaluée à faux, soit parce que l'activité précédente est annulée. Elle doit être compensée si elle a publié des données ou si elle a eu des effets dans le monde extérieur. La méthode de compensation dépend de la nature de l'activité (atomique, compensable, etc) et du point où elle en est arrivée dans son exécution.

Un deuxième problème est le cas d'une activité en état « anticipant » qui a lu des données d'une activité précédente qui passe à l'état « mort » (celle-ci peut être une activité dans l'état « anticipant » comme dans le cas précédent, ou une activité dans l'état « actif » qui est annulée par l'utilisateur). Son état est réévalué : elle peut passer à l'état « mort » aussi ou rester dans l'état « anticipant ». Si elle passe à l'état « mort », on applique le cas précédent. Si elle reste dans l'état « anticipant » (nous pouvons avoir ce cas, si elle est une activité de type OR-Join), les lectures impropres des données publiées par l'activité morte seront compensées en utilisant les nouvelles valeurs des données produites. Quand une activité passe à l'état « mort », son conteneur de sortie est mis à jour avec les valeurs par défaut de ses données (voir la définition 4.18). Ce mécanisme assure la compensation des données du procédé ; les autres données auxquelles les applications

accèdent et qui ne sont pas connues par le système de workflow doivent être compensées par des activités de compensation à la charge des agents d'exécution.

Discussion sur l'applicabilité de l'anticipation L'anticipation n'est pas appropriée dans toute situation et n'est pas applicable pour tout type d'activité. Par exemple, elle ne peut pas être appliquée pour une activité automatique ayant des effets qui ne peuvent pas être compensés. Cependant, elle peut être appliquée pour certains activités automatiques qui n'ont pas d'effets de bord, ou qui ont des effets qui ne doivent pas être compensés (l'exécution avec les résultats finaux est suffisante). Par exemple, une activité qui recherche les billets de vol disponibles dans une base de données, peut être automatiquement relancée à chaque modification de ses données d'entrée si la vitesse d'exécution du processus est plus importante pour l'application que la surcharge créée dans la base de données externe. Un exemple similaire est une activité qui compile un programme à chaque modification d'un de ses modules.

Dans le cas des activités manuelles, l'acteur humain en charge de l'activité est responsable de la prise en compte de la compensation.

Comme nous l'avons déjà vu, l'anticipation est particulièrement utile pour les activités de conception, dans lesquelles plusieurs versions d'un objet sont produites. En permettant l'exécution en parallèle de certaines activités qui sont définies en séquence dans le modèle du procédé, elle fournit un support à l'ingénierie simultanée (des activités dépendantes peuvent être simultanément actives).

Pendant la phase de définition du procédé, il n'est pas possible de savoir à l'avance quand l'anticipation se produira, mais il est possible de savoir quelles activités ne doivent pas anticiper. Ces activités seront marquées en tant que telles et suivront un modèle plus classique d'exécution. La stratégie générale pour l'exécution du procédé doit être décidée quand le processus est instancié. Selon le niveau de flexibilité exigé, l'anticipation libre, l'anticipation dictée par le flot de contrôle ou de données seront utilisées pendant tout le procédé.

4.3.7 Synthèse

Nous avons montré comment un modèle de procédé classique est interprété par le moteur de workflow pour permettre une exécution flexible. Par rapport à un modèle traditionnel, l'anticipation des activités et la circulation des données de procédé entre activités avec plusieurs versions sont permises. Trois stratégies d'anticipation ont été identifiées qui permettent différents degrés de flexibilité et de contrôle.

Pour synthétiser le modèle d'exécution proposé, considérons l'exemple de procédé d'édition d'un document illustré dans la partie 4.3.1. La figure 4.8-(2) montre les états des activités dans l'exécution coopérative, dans le cas de la stratégie d'anticipation dictée par le flot de contrôle.

Dès que l'activité « Edition » est démarrée, l'activité « Revue » passe à l'état « prêt à anticiper ». Dans cet état, la requête de personnel correspondante est calculée et un bon de travail est attribué à tous les acteurs qui la satisfont. L'état de l'activité est marqué comme « prêt à anticiper ». Supposons que l'activité d'édition produit une première version du document. Un des acteurs en charge de la revue peut démarrer l'activité correspondante qui va passer à l'état « anticipant ». Les bons de travail correspondants seront supprimés de la liste des autres acteurs.

Si l'activité « Édition » publie une nouvelle version du document, l'activité de revue recevra dans son conteneur d'entrée la nouvelle version.

Quand la première activité se termine, la deuxième activité passe à l'état « actif ». Dans cet état, une version définitive du document produit par l'activité d'édition est disponible. Après avoir pris en compte la nouvelle version, l'activité de revue peut terminer. Conformément au

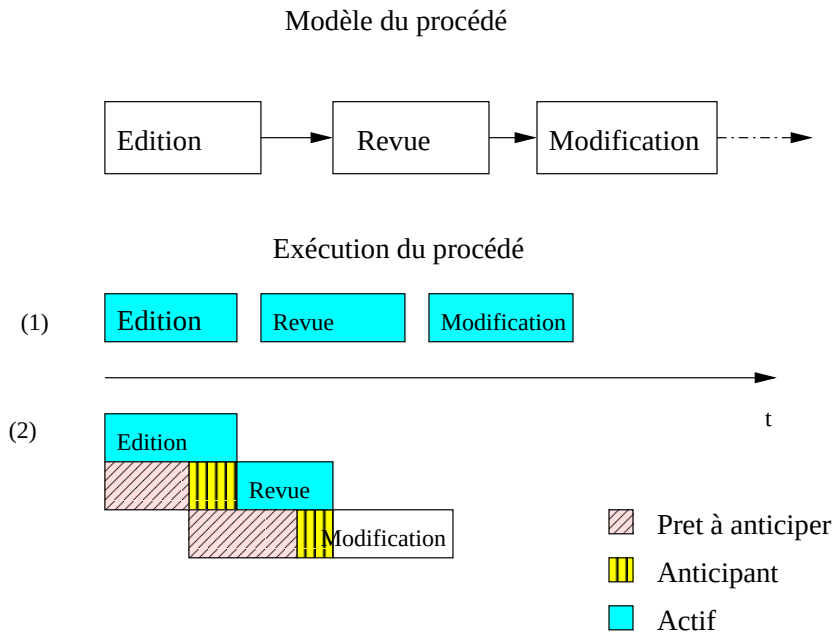


FIG. 4.8 – Les états des activités dans l'exécution d'un procédé sans (1) et avec (2) anticipation

diagramme de changements d'état, une activité peut terminer seulement à partir de l'état « actif ». Une activité passe à cet état seulement si la condition de transition de l'arc entrant est satisfaite. A ce moment, les données de son conteneur d'entrée sont synchronisées, elles vont contenir les version définitives des données produites par les activités précédentes. Le contrôle global est assuré et le procédé s'exécute comme si l'activité n'avait pas anticipé.

Nous avons montré quelles sont les modifications du modèle d'exécution nécessaires pour permettre ce type d'exécution coopérative.

- Deux nouveaux états ont été introduits dans le modèle de l'activité : « anticipant » et « prêt à anticiper ».
- Les activités dans l'état « prêt à anticiper » doivent être distribuées pour l'exécution aux acteurs en charge en indiquant leur état.
- La navigation se déroule différemment par rapport à une exécution classique : le démarrage d'une activité ou la publication d'une donnée peuvent aussi changer l'état des activités suivantes. Dans le modèle classique, le seul événement qui provoque une nouvelle étape de navigation est la terminaison d'une activité.
- Nous avons présenté les modifications du flot de données pour permettre l'échange de données anticipé entre deux activités successives.

Une analyse des impacts de ces modifications sur la correction de l'exécution du procédé et de l'applicabilité de l'approche a été présentée.

Une perspective de notre travail est de vérifier l'acceptation du modèle par les utilisateurs potentiels. Nous envisageons d'étudier l'utilisation de ce modèle par un groupe d'architectes dans le domaine de la construction du bâtiment. Dans le chapitre 6 nous présenterons un outil et une méthode pour analyser l'historique de procédé.

4.4 Modifications dynamiques

Le besoin de modifier le modèle de procédé peut apparaître soit parce que la phase de modélisation d'un procédé est complexe et pas assez précise, soit parce que les caractéristiques et les besoins des procédés changent dans le temps à cause de changements dans l'environnement, ou à de nouveaux objectifs.

Permettre aux utilisateurs de dévier par rapport à une séquence d'activités pré-modélisée, capture la liberté normale des participants de travailler dans un procédé. D'autre part, les changements sans restrictions rendent difficile la tâche d'assurer un comportement prévisible et correct du système. La responsabilité de l'action d'éviter des problèmes de cohérence ou des erreurs d'exécution ne doit pas être laissée à l'utilisateur ou au programmeur d'application. Le système devrait garantir que toutes les contraintes de cohérence qui ont été assurées avant un changement dynamique sont également assurées après que l'instance de procédé ait été modifiée. Tout d'abord, ceci exige que tous les types de dépendances structurales entre les tâches (dépendances de contrôle et de données) soient prises en compte. Les changements doivent considérer l'état de l'instance aussi. Par exemple, il n'est pas possible d'effacer une tâche si elle a été déjà terminée ou si elle est en cours d'exécution.

4.4.1 Opérations de modification dynamique

En général, les modifications du modèle sont réalisées en utilisant les opérations de base suivantes :

- insertion d'une activité,
- suppression d'une activité,
- insertion d'un arc,
- suppression d'un arc et
- modifications des propriétés d'un activité (application associée, requête de personnel, etc.).

L'application de ces opérations se fait comme dans [Wes96], [Rei98].

En dehors des opérations de modification dynamique, des opérations d'intervention pour les utilisateurs peuvent être utiles comme : l'opération qui permet de sauter une activité ou une séquence d'activités, d'arrêter l'activité en cours et passer à la suivante, etc. Une opération importante dans le contexte des procédés coopératifs est celle qui permet la répétition d'une séquence d'activités. Cette opération est utile pour gérer le feedback. Si dans un procédé de production, des conditions précises spécifient la répétition d'une séquence d'activités, dans les procédés coopératifs le besoin d'intégrer des commentaires peut apparaître dynamiquement. Nous détaillons cette opération et ses conditions d'application pour illustrer les problèmes de cohérence d'exécution qui doivent être pris en compte pour les opérations de modifications dynamiques.

Repetition d'une activité

Supposons que dans le procédé de la figure 4.9, l'utilisateur travaille sur l'activité l. L'acteur se rend compte que l'activité l ne produit pas le résultat correct et que les activités j et k doivent être ré-exécutées.

La répétition d'une activité ne peut pas être capturée facilement dans le diagramme de transition d'états parce que ce diagramme spécifie les transitions des états pour une activité, et l'opération de répétition d'une activité crée une nouvelle instance de l'activité.

Nous supposons que seules les exécutions séquentielles sont permises ; la répétition d'une séquence d'activités peut démarrer seulement quand la précédente a été finie.

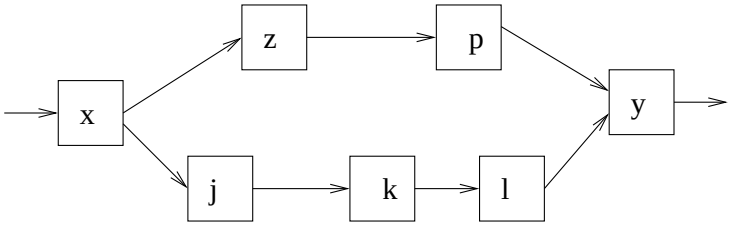


FIG. 4.9 – Fragment de procédé

Les données nécessaires aux activités doivent être fournies par le système. En plus des données initiales, les données produites par les instances précédentes des activités peuvent être utilisées (par exemple des commentaires produits par l'activité l).

Considérons une exécution séquentielle des activités j, k, l montrée dans la figure 4.9. Supposons que après l'exécution de l'activité l, l'utilisateur décide de continuer le procédé en répétant l'activité j. Alors, après la terminaison de l, une autre instance de j est créée. Nous remarquons que l'ordre parait contredire le modèle de procédé, comme j devrait être exécutée avant l, et une instance de j est démarrée après la terminaison de l'instance de l'activité l. Comme nous supposons que les répétitions sont séquentielles, pour chaque répétition les contraintes de flot de contrôle définies dans le modèle de procédé, sont satisfaites.

Supposons que les activités z et l sont actives et l'acteur exécutant l décide de répéter j. Cette opération peut être exécutée parce qu'elle ne contredit pas les contraintes du flot de contrôle. Après la répétition de j, de nouvelles instances des activités k et l sont créées et les deux flots d'exécution concurrents se retrouvent finalement en y.

Il existe des restrictions sur l'opération de répétition d'une activité. Supposons que après la terminaison de l, l'acteur décide de répéter x. Dans ce cas, z est encore active, pendant qu'une instance de x est aussi active. Cette situation contredit les contraintes du flot de contrôle, parce que les activités x et z sont définies comme séquentielles mais elles sont exécutées de manière concurrente.

Pour éviter cette situation, nous supposons qu'après la terminaison d'une instance de l'activité k, une activité j peut être répétée seulement si tous les arcs sortants de j atteignent k. Par exemple, dans la figure 4.9, un saut en arrière de l'activité l à l'activité j respecte cette supposition. L'opération de répétition de x à partir de l, ne la respecte pas et en conséquence, elle est interdite. Par contre, la répétition de x à partir de y est faisable, parce que tous les chemins partant de x atteignent finalement y.

4.4.2 Conclusion

Les opérations de changements dynamiques et les opérations d'intervention pour les utilisateurs sont utiles pour résoudre des situations exceptionnelles et pour donner la liberté aux participants de mieux résoudre leurs tâches. Dans un procédé coopératif, la possibilité de modifier le procédé est un besoin fondamental parce que le procédé ne peut pas être complètement spécifié à l'avance. Le fait que notre modèle soit simple et qu'il n'impose pas beaucoup de restrictions, facilite l'application des changements dynamiques. Nous n'avons pas approfondi le problème des modifications dynamiques dans cette thèse. Mais nous pensons que les approches pour les changements dynamiques existantes dans la littérature peuvent être appliquées et complètent notre proposition pour la flexibilité [Wes96], [Rei98].

4.5 Composant de gestion des données

Dans cette section nous présenterons le système de gestion de transactions coopératives, comme l'un des composants qui pourrait être intégré avec le composant workflow présenté dans la section précédente. Le modèle de transactions coopératives COO [Can96], [Mol96], [God96] [Can98] a été développé dans l'équipe ECOO pour permettre aux activités coopératives de longues durées d'échanger des données en cours d'exécution.

4.5.1 Protocole COO

Le protocole COO relâche la propriété d'isolation du protocole traditionnel des transactions et garantit la Coo-serialisabilité. Relâcher l'isolation est considéré comme un besoin fondamental pour les transactions longues. Naturellement, ceci induit le risque d'incohérence dû aux mises à jour perdues ou aux lectures impropres. Le protocole COO permet d'éviter ce risque : il oblige les utilisateurs à compenser les lectures impropres avant de terminer leur transaction et à se synchroniser en cas de dépendances cycliques. En fait, le protocole impose à une transaction qui a lu un résultat intermédiaire de lire la valeur finale correspondante. La compensation du travail effectué avec le résultat intermédiaire reste sous la responsabilité de l'utilisateur. Le protocole fonctionne de la manière suivante :

- les activités s'exécutent et partagent un espace de travail public. Les données dans cet espace sont versionnées.
- chaque activité a un espace de travail privé dans lequel elle peut lire des données de l'espace public.
- chaque activité peut écrire des données dans l'espace public à tout moment pendant son exécution. Les données écrites pendant son exécution s'appellent résultats intermédiaires. Des données écrites à la fin de son exécution (moment du commit) s'appellent résultats finaux. Dans la figure 4.10, chaque rectangle représente l'exécution d'une activité. Les opérations sont représentées par des flèches étiquetées avec le nom de l'opération (w/r) et le nom de l'objet ; $w(x)$ signifie que l'activité a écrit l'objet x à ce moment de temps.
- si une activité lit un résultat intermédiaire pendant son exécution, elle doit lire le résultat final correspondant avant de terminer. Nous considérons que la lecture du résultat final permet à l'utilisateur de compenser la lecture impropre faite avant. Les utilisateurs sont responsables d'effectuer dynamiquement la compensation.
  Dans la figure 4.10, l'activité « Édition partie B » lit la dernière valeur de x et exécute la compensation appropriée (par exemple, elle relit et met à jour y). Le gestionnaire de transactions maintient une relation de dépendance entre les activités pour mémoriser le fait qu'une activité a lu le résultat intermédiaire d'une autre. Dans notre exemple, quand l'activité « Édition partie B » a lu la valeur intermédiaire de l'objet x produit par l'activité « Édition partie A », une dépendance « Edition partie B » $\rightarrow$ « Edition partie A » est crée. Quand l'activité « Édition partie B » lit la valeur courante de x après la terminaison de l'activité « Édition partie A », la dépendance est supprimée.
- Une transaction ne peut pas terminer si elle est encore dépendante d'une autre transaction. Si une transaction essaie de terminer sans lire la valeur finale d'un objet après avoir lu une valeur intermédiaire de cet objet, l'opération de terminaison est annulée et la transaction reste active.
- Si deux ou plusieurs activités dépendent mutuellement de résultats intermédiaires, elles sont groupées et doivent se synchroniser afin de terminer simultanément. Les utilisateurs doivent négocier afin de convenir de la valeur finale des deux objets en même temps. Une

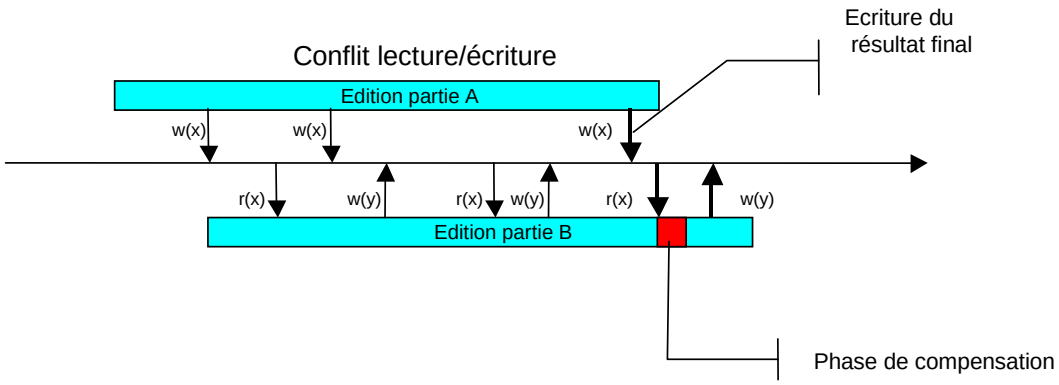


FIG. 4.10 – Échanges de résultats intermédiaires

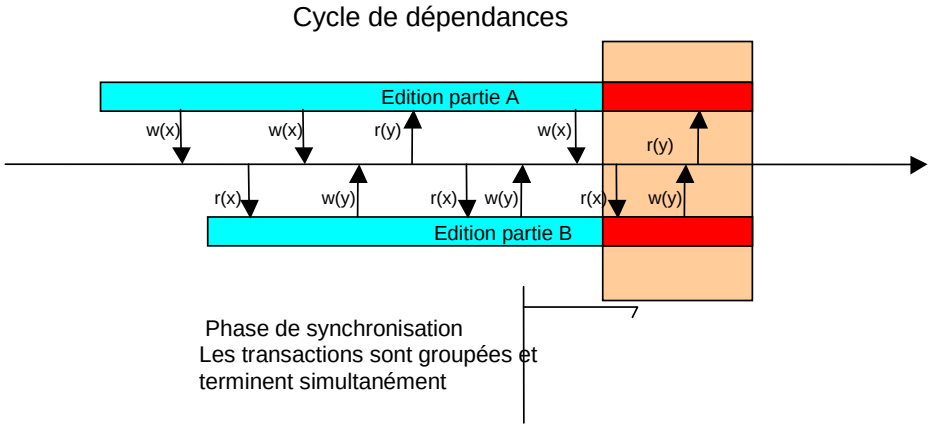


FIG. 4.11 – Cycle de dépendances

transaction groupée peut être considérée comme une transaction dans laquelle deux (ou plusieurs) utilisateurs interagissent. Dans l'exemple de la figure 4.11, « Édition partie A » et « Édition partie B » dépendent mutuellement de x et y. Dans la dernière phase de la transaction, elles se synchronisent afin de terminer ces transactions comme un groupe. De l'extérieur, « Édition partie A » et « Édition partie B » sont considérées comme une seule transaction. Cette phase se déroule de la manière suivante :

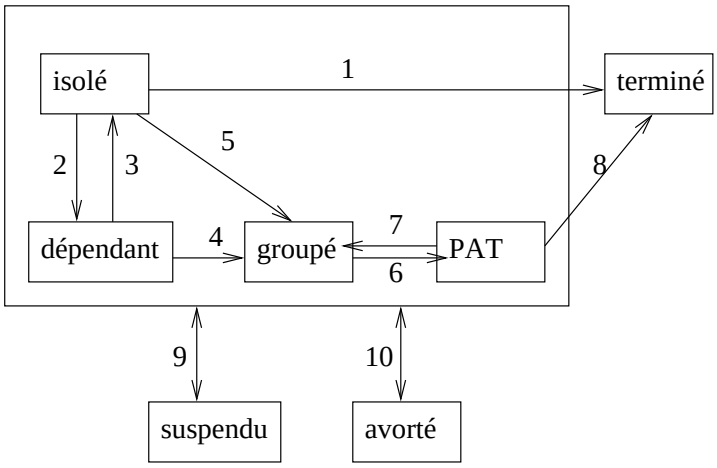
- Une transaction membre du groupe de transactions peut essayer de démarrer la terminaison du groupe en essayant de terminer. L'opération « Terminer », dans ce cas produit un ensemble de résultats potentiels finaux et change l'état de la transaction de l'état « groupé » à l'état « prêt à terminer » (PAT).
- quand un membre du groupe essaie de terminer et tous les autres sont dans l'état PAT, alors toutes les transactions terminent simultanément. Les résultats potentiellement finaux deviennent des résultats finaux.
- quand un membre du groupe essaie de terminer et il existe un autre membre qui est encore actif, alors il produit des résultats potentiellement finaux et passe à l'état PAT.
- si un membre du groupe produit un nouveau résultat intermédiaire pendant la terminaison du groupe, alors cette tentative de terminaison est annulée et tous les membres du groupe passent de nouveau à l'état actif. C'est le moyen pour une activité d'exprimer son désaccord par rapport aux valeurs produites par le groupe et de demander de travailler encore sur l'objet commun.

La figure 4.12 présente le diagramme de transitions d'états d'une transaction. Quand une transaction est créée, elle commence son exécution comme une transaction traditionnelle en passant à l'état « isolé » (nous appelons cet état « isolé » en référence aux transactions ACID traditionnelles où les transactions s'exécutent en isolation). Elle peut terminer normalement ou être avortée.

Une transaction qui a lu un ou plusieurs résultats intermédiaires, mais n'est pas impliquée dans un cycle est dans l'état « dépendant ». Une transaction qui est en état « dépendant » et qui annule sa dernière dépendance revient à l'état « isolé ». Les transactions impliquées dans un graphe cyclique de dépendances forment un groupe de transactions : chaque transaction du groupe passe à l'état « groupé ». Une transaction ne peut pas terminer si elle dépend d'une autre transaction (pas à jour). Si une transaction membre du groupe essaie de terminer (elle est à jour avec les autres transactions), et il existe des transactions dans le groupe dans l'état groupé, elle entre dans l'état « prêt à terminer » (PAT). Si une transaction membre du groupe veut terminer et les autres transactions du groupe sont dans l'état « prêt à terminer » , toutes les transactions du groupe se terminent simultanément et passent à l'état « terminé ». Cependant, si un membre du groupe produit un nouveau résultat intermédiaire pendant la phase de terminaison du groupe, alors cette terminaison est annulée et tous les membres du groupe reviennent à l'état groupé.

Nous rappelons que l'écriture d'un résultat intermédiaire sera toujours sous la responsabilité de l'auteur et que la lecture d'un résultat intermédiaire est également toujours sous la responsabilité du lecteur. Ainsi, la dépendance et le groupement peuvent se produire seulement si les deux parties sont d'accord, et sont ainsi bien informées du risque. De la même manière, le procédé de réconciliation dans le cas du groupement est également sous la responsabilité des utilisateurs.

On peut aussi remarquer qu'une activité est automatiquement ajoutée à un groupe coopératif lorsqu'elle est impliquée dans un cycle. Ceci peut induire le risque du groupement de toutes les transactions dans un grande groupe, qui peut compliquer la convergence du groupe. On verra plus tard comment en exploitant la définition du flot de contrôle, ce risque est diminué. En fait, seules les activités pour lesquelles il n'y a pas une relation d'ordre pourront se retrouver groupées.



PAT: pret à terminer

1: terminaison d'une transaction isolée
2: lecture d'un résultat intermédiare
3: la dernière dépendance a été enlevée
4: création d'un cycle
5: lecture d'un résultat intermédiare qui ferme un cycle de dépéndances
6: à jour
7: une transaction du groupe a publié un nouveau résultat intermédiare
8: la dernière transaction du groupe termine
9:décision de l'agent
10: décision de l'agent

FIG. 4.12 – Diagramme d'états de transaction

4.6 Intégration du composant workflow et du composant gestionnaire des transactions

Dans cette section nous présentons comment, en intégrant le système de gestion de workflow présenté dans la section 4.3 avec un gestionnaire de transactions (les COO-transactions), on peut supporter un échange de données flexible entre les activités. Nous analysons les fonctionnalités qui doivent être fournies par les deux composants afin de développer un système de workflow coopératif « simplement » en intégrant le système de workflow avec le gestionnaire de transactions. Finalement nous décrivons les problèmes et les contraintes induits par cette intégration par rapport à la correction des échanges de données. Plus précisément, le rapport entre la flexibilité des échanges de données et le contrôle qui est exprimé dans le modèle de procédé doit être analysé.

4.6.1 Cycle de vie d'une activité et d'une transaction

Une activité sera encapsulée dans une COO transaction et le cycle de vie de la transaction sera lié au cycle de vie de l'activité.

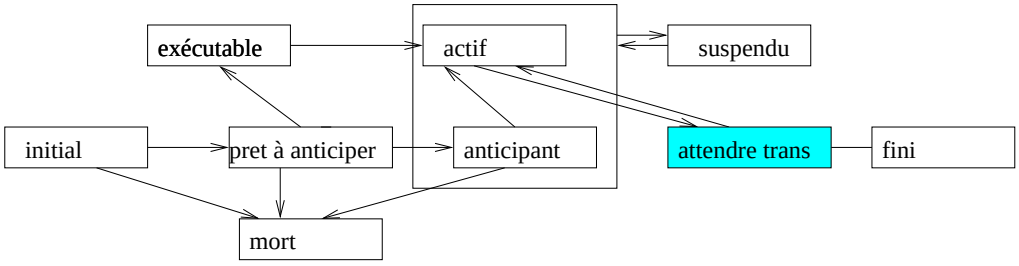


FIG. 4.13 – Diagramme d'états d'activité

Quand une activité est démarrée par un acteur (l'état « actif » ou « anticipant ») une COO transaction est créée. De la même manière, quand l'activité est finie, la transaction doit terminer aussi. Mais du fait de la coopération, la demande de fin de transaction liée à la fin de l'activité peut ne pas aboutir. Le protocole COO peut annuler la demande de terminaison d'une transaction si elle ne respecte pas son critère de correction ; il peut lui imposer d'attendre la fin d'une autre transaction ou de se synchroniser avec les autres transactions, si elle fait partie d'un cycle de dépendances.

Pour gérer cette situation, une solution est d'introduire un nouvel état, appelé « Attendre transaction ». Cet état est différent de l'état suspendu. Une activité est dans l'état suspendu quand elle a arrêté temporairement son exécution ; la condition de sortie de l'activité n'est pas encore satisfaite. Une activité se trouve dans l'état « Attendre transaction » quand elle veut terminer (sa condition de fin est satisfaite) et demander que la transaction attachée finisse également.

Si la transaction peut terminer, le WfMS en est informé et l'activité correspondante peut passer à l'état « fini ». Sinon, l'activité retourne à l'état « actif » pour se synchroniser avec de nouvelles valeurs de résultats intermédiaires ou finaux.

Cet algorithme de terminaison, qui implique un dialogue entre le système de workflow et le gestionnaire de transactions est décrit ci-dessous :

Terminaison d'une activité et COO transaction

1. Le système de workflow demande la fin de la transaction (l'interaction 1 dans la figure 4.14 entre le système de workflow et le gestionnaire de transactions) ; l'activité passe de l'état « actif » à l'état « attendre transaction ».
2. Le gestionnaire envoie la réponse. On peut avoir les cas suivants :
   - la transaction correspondante ne fait pas partie d'un groupe de transactions
     - si elle est dépendante d'une autre transaction, elle doit attendre la fin de celle-ci pour lire le résultat final (réponse = dep)
     - sinon, la transaction est commitée et la confirmation est envoyée au système de workflow (réponse =OK)
   - si elle fait partie d'un groupe de transactions
     - si toutes les autres sont dans l'état PAT, la transaction va déclencher la terminaison de toutes les autres transactions du groupe (réponse =groupeOk)
     - s'il existe encore des transactions dans le groupe dans l'état actif, la transaction passe à l'état PAT (réponse = groupe)
3. Le WFMS change l'état de l'activité en fonction de la réponse reçue : si la réponse est « OK », l'activité passe à l'état « fini », si la réponse est « dep », elle passe à l'état « actif », si la réponse est « groupe », elle reste dans l'état « attendre transaction ». Si l'activité reste dans l'état « attendre transaction », elle attend la fin du groupe, auquel cas elle sera notifiée (étape 3 dans le dessin) et pourra finir. La dernière transaction du groupe va déclencher la terminaison du groupe d'activités. Toutes les activités appartenant au groupe vont terminer en même temps. Si un nouveau résultat intermédiaire est produit par une transaction membre du groupe, la transaction qui était dans l'état « attendre transaction » revient à l'état « actif ».

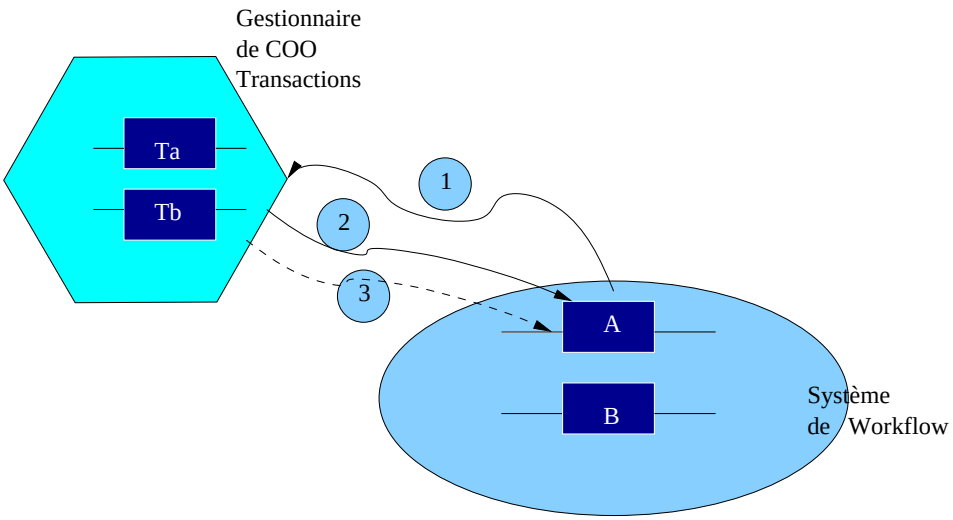


FIG. 4.14 – Terminaison d'une activité et de la transaction attachée

4.6.2 Données partagées d'un procédé

Dans le modèle classique de workflow présenté dans la section 4.2, les données de procédé sont transmises directement entre activités. Ce type d'échange de données reflète la nature des

modèles de type workflow basée sur le flot de contrôle. Pour supporter des environnements plus coopératifs, des types supplémentaires d'échanges de données doivent être permis. Les données doivent pouvoir être accédées à l'initiative de l'utilisateur à partir d'un espace partagé qui doit en contrôler l'accès (principe du libre service). Les activités s'échangent des résultats intermédiaires. Le travail concurrent sur les mêmes données doit être considéré.

Pour que ces types d'échanges soient possibles, il est nécessaire de réaliser l'intégration du système de workflow avec le gestionnaire de transactions introduit précédemment. En particulier, le système de workflow doit connaître les objets partagés lus et écrits par ses activités pour des raisons de traçabilité et recouvrement.

Les données de l'espace partagé qui sont utilisées par les activités coopératives, mais ne le sont pas par des données de procédé sont appelées des données partagées. Le flot des données de procédé et le flot de données partagées ont des caractéristiques différentes :

- L'échange de données partagées ne peut pas être complètement spécifié à l'avance, en raison de la nature imprévisible des activités coopératives.
- Pour les données partagées, le travail des activités en parallèle sur les mêmes données doit être possible pour supporter les styles de travail opportunistes. Pour les données de procédés, leur traitement est connu d'avance et l'enchaînement des activités qui peuvent les modifier est strictement spécifié. (voir la restriction (2) de la définition A.9 du flot de données qui permet un échange de données entre deux activités seulement si une relation de précedence existe entre elles).

Nous définissons ci-dessous le concept de données partagées d'un procédé et les mécanismes d'accès associés dans l'objectif d'assurer des mécanismes unifiés d'accès aux données de procédé et aux données partagées.

Les opérations « Lire » et « Écrire » peuvent s'appliquer aussi aux données partagées.

Un modèle de procédé $P \in \mathcal{P}$ peut avoir un espace public $g(P) \subseteq V$ qui constitue l'ensemble de ses objets partagés auxquels toutes ses activités ont accès.

Les données partagées qui sont connues d'avance comme nécessaires pour une activité peuvent être spécifiées en connectant ces données partagées avec les données d'entrée de l'activité. La signification d'un tel connecteur (le connecteur 3 de la figure 4.15) est qu'au démarrage de l'activité, la valeur actuelle de la donnée partagée est copiée dans le conteneur d'entrée de l'activité. De la même manière, les données partagées modifiées par une activité peuvent être spécifiées en les connectant avec des éléments de sortie de l'activité (connecteur 2 de la figure 4.15).

La définition suivante reflète comment les données sont échangées entre l'espace public et ses activités.

Définition 4.22 (Application de connecteurs de données partagées)
Soit $P \in \mathcal{P}$ un modèle de procédé et soit N l'ensemble de toutes ses activités. L'application

$$\Delta_g : N \to \bigcup_{A \in N} (\rho(g(P) \times i(A)) \cup \rho(o(A) \times g(P)))$$

est appelée application de connecteurs de données partagées et satisfait les conditions suivantes :

1. $\forall A \in N : \Delta_g(A) \in (\rho(g(P) \times i(A)) \cup \rho(o(A) \times g(P))$
2. $\forall B \in N : (x, z), (y, z) \in (\rho(g(P) \times i(B)) \cup \bigcup_{A \in N} \Delta(A, B) \Rightarrow x = y$

Un élément (v_1, v_2) est appelé un connecteur de données.

La première condition assure qu'une application de connecteurs de données partagées qui a

comme origine ou destination l'activité A spécifie seulement des applications de données qui ont A comme origine, ou respectivement destination.

Pour éviter des opérations conflictuelles de copie à l'exécution, la condition (2) interdit d'avoir plus d'une application de donnée ayant comme destination le même élément de donnée dans le conteneur d'entrée d'une activité. Elle combine des applications de données ayant comme origine une donnée partagée avec des applications de données entre activités.

Cette application de connecteurs de données partagées peut être modifiée dynamiquement : une activité peut ajouter dynamiquement un connecteur de donnée entre un objet partagé et un de ses paramètres d'entrée ou de sortie. Les paramètres d'entrée et de sortie peuvent être ajoutés dynamiquement aussi.

Nous décrivons maintenant comment s'appliquent les connecteurs de données partagées pendant la navigation. Les connecteurs ayant comme destination une donnée partagée s'appliquent chaque fois qu'une opération d'écriture est effectuée pour la donnée de sortie d'une activité qui est l'origine du connecteur de donnée et lors de la terminaison de l'activité.

Un connecteur de donnée ayant comme origine une donnée partagée s'applique au démarrage de l'activité, pour construire l'espace privé de l'activité et pour chaque opération de lecture sur la donnée correspondante.

Définition 4.23 (Appliquer les connecteurs de données)
- $\forall X \in N \cup \mathcal{C}, \forall v \in i(X) : (w, v) \in \Delta_g(X)$
  - mode distribution $\Rightarrow^i v =^{i-1} w$
  - mode libre service
    Soit m le moment de la dernière lecture : $m = \max_j\{< X_j, lire, d_j, j > \in H | j \leq i$ et $d_j = v$ et $X_j = X\} \Rightarrow^i v =^m v =^{m-1} w$
  ou $i < j$, ou j est le moment de terminaison de l'activité ; Pour $i > j \Rightarrow^i v =^{i-1} v$.
- $\forall P \in \mathcal{P}\, \forall u \in g(P) : (t, u) \in \Delta_g(X), si < X, write, t, i > \in H \Rightarrow^i u =^{i-1} t$

La première condition décrit l'application d'un connecteur de données qui a comme destination un élément du conteneur d'entrée d'une activité pour les deux modes d'échanges de données possibles (connecteur 3 de la figure 4.15). Pour le mode distribution, si une activité anticipe et, pour une de ses données d'entrée une version est publiée, sa donnée d'entrée sera automatiquement mise à jour. Pour le mode libre service, la valeur correspond à la dernière valeur lue par une opération Lire. Après la terminaison de l'activité, le connecteur de donnée n'est plus appliqué ; la donnée du conteneur d'entrée garde sa dernière valeur.

La deuxième condition décrit l'application d'un connecteur de données qui lie un élément de donnée de sortie d'une activité avec un élément de donnée partagé du procédé (connecteur 2 de la figure 4.15). La valeur de l'objet partagé u sera la valeur copiée par la dernière opération d'écriture effectuée par une des activités qui a un connecteur de données vers l'objet u.

On note que plusieurs connecteurs peuvent exister ayant comme destination l'objet u.

4.6.3 Problèmes et contraintes

Appliquée au modèle traditionnel de workflow, l'encapsulation des activités dans des transactions coopératives permet la rupture de l'isolation des activités qui s'exécutent en parallèle. Cependant, deux activités successives ne peuvent pas partager des résultats intermédiaires. L'anticipation permet la relaxation de l'isolation entre activités successives. La combinaison des deux approches permet potentiellement à n'importe quelle activité de partager des résultats intermédiaires avec n'importe quelles autres activités. Ceci est illustré dans la figure 4.16 où l'activité B

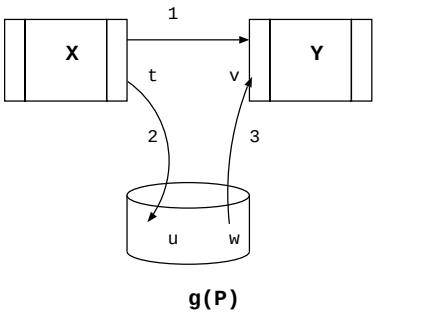


FIG. 4.15 – Flot de données

peut anticiper son exécution grâce aux résultats intermédiaires fournis par A tandis qu'elle peut lire des résultats intermédiaires de l'activité C s'exécutant en parallèle.

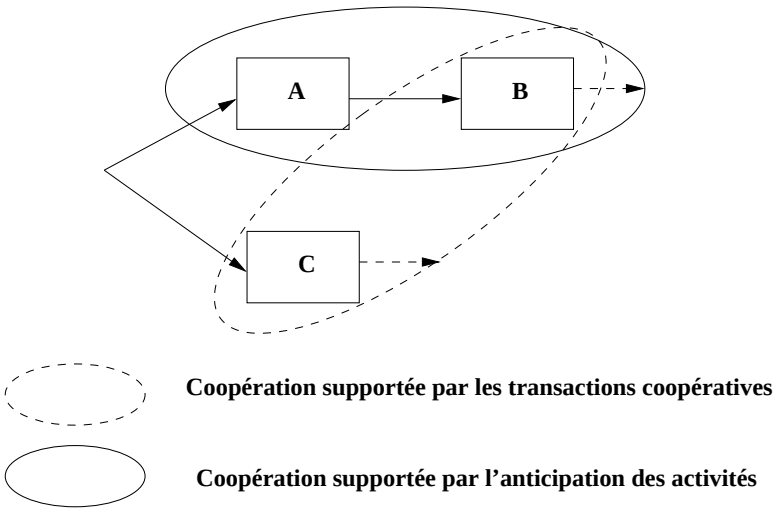


FIG. 4.16 – Coopération supportée par anticipation et transactions coopératives

L'anticipation permet aux activités successives de s'exécuter partiellement en parallèle, en augmentant ainsi le degré de parallélisme. Ces activités peuvent échanger des données, générant des dépendances dynamiques entre elles. Si la transaction A lit un résultat intermédiaire de B, alors A devient dépendante de B et doit finir après B ($A \rightarrow B$). Ces dépendances s'ajoutent aux dépendances statiques définies dans le modèle de procédé. Une dépendance dans le modèle de procédé est exprimée par un connecteur de contrôle entre deux activités. Si un connecteur va de l'activité E à l'activité F, alors F doit finir après E ($F \rightarrow E$).

Le graphe de dépendances initial qui résulte du modèle de procédé est augmenté pendant l'exécution avec les dépendances dynamiques.

L'anticipation et l'échange de données partagées peuvent conduire à des cycles dans le graphe global de dépendances. La figure 4.17 illustre un tel cycle. L'activité A lit un résultat intermédiaire de B, devenant dépendante de B. L'activité B lit un résultat intermédiaire de C, en devenant dépendant. Par transitivité, A devient dépendante de C. Mais, la relation d'ordre définie dans le

flot de contrôle spécifie que C doit finir après A, donc C est dépendante de A.

Deux possibilités existent pour gérer les conflits entre les dépendances dynamiques et celles statiques :

- autoriser la terminaison simultanée d'actives successives. Si une dépendance dynamique ferme un cycle dans le graphe de dépendances (statiques plus dynamiques), toutes les activités impliquées dans le cycle doivent terminer en même temps, suivant le protocole COO pour la terminaison d'un groupe.
  Cette possibilité permet l'échange de données en sens inverse du flot de contrôle. Par exemple, cela permet l'envoi du «feedback» d'une activité vers l'activité qui la précède, mais les activités seront obligées de finir en même temps.
  L'inconvénient est le risque d'avoir un grand nombre d'activités groupées, ce qui peut compliquer la convergence. En fait, si ce phénomène ne peut pas être exclu théoriquement, il est limité en pratique par le flot naturel des actions qui assurent globalement que l'application avance vers son but.
- imposer un ordre stricte de terminaison entre activités successives. La formation des cycles est évitée en empêchant les lectures qui pourraient fermer des cycles. Cette stratégie assure que les dépendances exprimées dans le flot de contrôle sont interprétées comme des dépendances fin-fin.

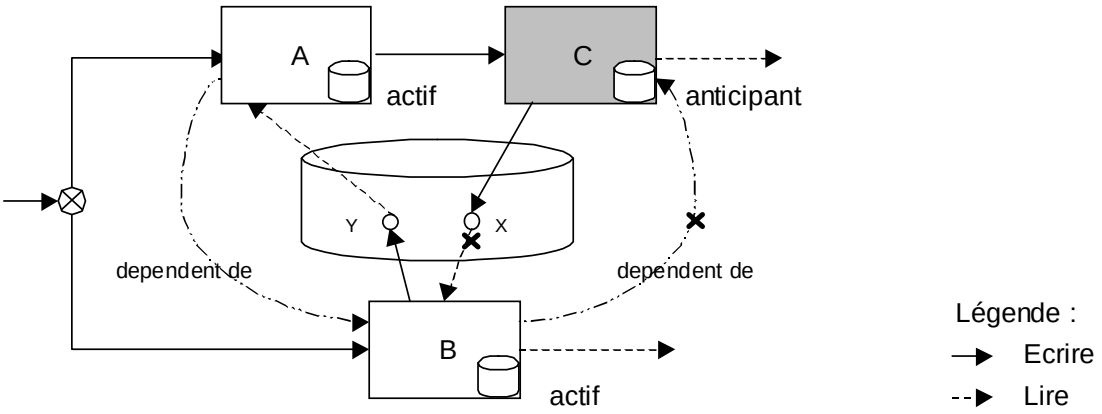


FIG. 4.17 – Interdiction des cycles de dépendances

La contrainte suivante est ajoutée :

RÈGLE 4.1
Une activité ne peut pas lire un résultat intermédiaire d'une autre activité si cette lecture crée un cycle qui implique deux activités entre lesquelles il existe une dépendance de flot de contrôle.

Pour appliquer cette contrainte, l'opérateur de lecture doit être surchargé afin qu'une première étape vérifie si la lecture crée un cycle. Pour cette vérification, on doit connaître les dépendances entre les transactions et la relation d'ordre définie dans le modèle de procédé. Un dialogue entre le système de workflow et le gestionnaire de transactions est nécessaire car le gestionnaire de transactions gère les cycles entre les transactions et le système de workflow connaît la relation d'ordre entre activités.
Le gestionnaire de transactions fournit la transaction qui a écrit la dernière valeur et

l'ensemble de transactions desquelles elle est dépendante et également l'ensemble de transactions dépendantes de la transaction courante. Dans l'exemple de la figure, la transaction qui a effectué la dernière écriture est B, il n'y pas de transactions dépendantes de B et la transaction courante C est dépendante de A.
Ensuite, le système de workflow peut vérifier si dans l'ensemble de dépendances qui seraient créées par cette lecture, il y a une dépendance dynamique contredisant une relation d'ordre entre les activités afférentes exprimée dans le modèle du procédé. Si c'est le cas, la lecture est interdite, autrement elle est exécutée en appelant l'opérateur du gestionnaire de transactions COO.
L'inconvénient de cette stratégie est le fait qu'elle ne permet pas la transmission des données (en particulier des commentaires) dans le sens inverse du flot de contrôle à travers l'espace partagé. Ceci conduirait à la création d'un cycle entre les deux activités. Ces données peuvent être transmises en dehors du contrôle de système de workflow. Dans ce cas il peut être nécessaire de réitérer une partie du procédé pour gérer le cas où une activité a été finie et l'activité suivante a encore des commentaires à fournir. L'opération utilisateur qui permet de répéter une activité précédente a été présentée dans la section 4.4.

4.6.4 Conclusion

Dans cette partie nous avons étudié l'intégration d'un système de workflow avec un gestionnaire de transactions coopératives pour permettre un échange de données flexibles entre activités.

Les activités sont encapsulées dans des transactions coopératives et peuvent échanger des résultats intermédiaires en cours d'exécution. Au démarrage de l'activité, la transaction associée est créée et enregistrée dans le gestionnaire de transactions.

Pour la synchronisation de la terminaison de l'activité et de la transaction attachée, un nouvel état de l'activité doit être introduit. Lors de la phase de terminaison, un dialogue entre le système de workflow et le gestionnaire de transactions est nécessaire.

L'échange spontané de données entre activités simultanées peut générer des dépendances dynamiques entre activités. Nous avons analysé le rapport entre ces dépendances dynamiques et les dépendances statiques représentées dans le flot de contrôle et nous avons indiqué comment résoudre les problèmes générés.

L'intégration du système de workflow et du gestionnaire de transactions permet à des activités d'échanger des données en cours d'exécution, d'accéder de manière spontanée et sans être bloquées par d'autres activités aux données partagées du procédé.

4.7 Workflow coopératif : synthèse

Dans ce chapitre nous avons décrit un modèle de workflow coopératif qui fournit la flexibilité nécessaire aux applications coopératives et qui garde la simplicité du modèle classique de workflow (administratif, de production).

Le concept d'anticipation a été introduit pour affaiblir la dépendance stricte de précédence entre activités successives. Ce concept permet aux activités de démarrer plus tôt leur exécution, en se superposant avec les activités précédentes et en consommant plusieurs versions des données produites par celles-ci.

L'échange flexible de données est assuré en encapsulant des activités en transactions coopératives. Ceci permet aux activités de travailler de manière coopérative sur les données communes aux procédés : s'échanger des versions intermédiaires, travailler en même temps sur les mêmes données.

L'anticipation et l'échange flexible de données peuvent être adaptés en fonction des besoins du procédé et des règles de travail de l'équipe virtuelle. A l'extrême, un procédé compétitif (ou traditionnel) est simplement un procédé coopératif où l'anticipation est interdite dans le composant de gestion d'activités et la visibilité des résultats intermédiaires (la transition 2 du schéma 4.12) est interdite dans le composant de gestion de transactions.

Nous reprenons l'exemple du procédé d'édition d'un document pour illustrer les différents types d'exécutions possibles. L'exécution la plus stricte est l'exécution dite « compétitive », présentée dans la figure 4.18. Au démarrage du procédé, le document est « chargé » depuis l'espace partagé et il est passé d'une activité à l'autre, les activités s'exécutant de manière strictement séquentielle.

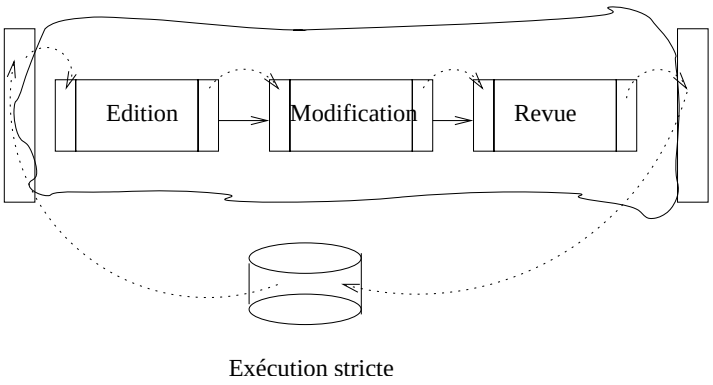


FIG. 4.18 – Différents degrés de flexibilité

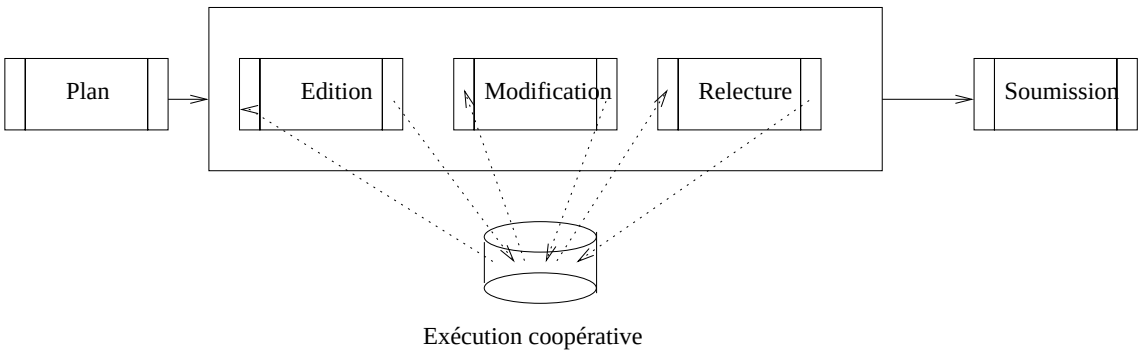


FIG. 4.19 – Différents degrés de flexibilité - Coopération *ad hoc*

L'exécution la plus flexible peut être obtenue en définissant des fragments de procédés, ou même des activités isolées et en omettant la définition du flot de données. Dans ce cas, les activités lisent des données à partir de l'espace partagé et la correction des échanges de données est assurée par le gestionnaire de transactions. Le rôle du système de workflow est limité à assurer l'affectation des activités aux participants et le flot de contrôle défini dans les fragments de procédé. Les parties non-structurées peuvent être incluses dans un procédé plus structuré, comme dans l'exemple de la figure 4.19.

Entre les deux extrêmes, l'exécution stricte et la coopération *ad hoc*, différentes stratégies d'anticipation et différents modes d'échanges de données peuvent être définis pour répondre aux

besoins des applications coopératives.

Chapitre 5

Mise en œuvre

Les propositions décrites dans cette thèse ont fait l'objet de plusieurs mises en œuvre dans différents contextes. Elles permettent de montrer la validité de l'approche dans le contexte plus large de la construction d'une plate-forme de coopération. La forme la plus aboutie de ces mises en œuvre est celle qui réalise l'intégration du moteur de workflow flexible dans le prototype MOTU. Ce prototype intègre en effet l'ensemble des services de coopération qui font partie des objectifs du projet ECOO. C'est cette plate-forme que nous décrirons en premier. La seconde mise en œuvre a pu être réalisée au cours d'une coopération avec la compagnie HITACHI dans le projet Corvette. L'objectif du projet a été l'intégration d'un système de Workflow commercial et du service de transactions coopératives implanté sur la plate-forme MOTU. Cette mise en œuvre montre l'intérêt de l'intégration d'un système transactionnel coopératif générique de gestion des données avec un système de workflow traditionnel, les adaptations qu'il a fallu faire au protocole mais également ses limites.

5.1 Le prototype MOTU

L'objectif du prototype MOTU est de montrer l'ensemble des fonctionnalités d'un environnement de travail coopératif intégrées au sein d'un même environnement. C'est une implantation en java sur un modèle client-serveur réalisée au sein de l'équipe ECOO.

5.1.1 Description général de MOTU

MOTU est une implantation client-serveur d'un ensemble de services destinés à faciliter la coopération d'une équipe virtuelle. MOTU intègre les fonctions de base nécessaires au travail en groupe et à la coordination implicite :

- Un service d'événements qui va servir à l'intégration des différents services et à la notification des clients.
- Un service de gestion de versions. Le service de gestion de versions constitue la base pour permettre le travail concurrent sur des objets partagés.
- Un service d'authentification et de gestion des utilisateurs qui remplit les fonctions classiques d'identification et de gestion des droits d'accès.
- Un service de gestion d'espace de travail. Le service de gestion des espaces de travail permet aux utilisateurs de définir l'ensemble des objets sur lesquels ils travaillent grâce à un système de vue et assure la synchronisation de ces objets entre la base partagée et une copie de ces objets sur l'ordinateur utilisé par l'utilisateur. Ceci permet aux utilisateurs de disposer

d'une copie des fichiers en permanence et de pouvoir les modifier avec les outils appropriés. Les mises à jours des fichiers sont notifiées immédiatement à tous les utilisateurs. Un mécanisme de publication et de synchronisation (checkin/checkout) permet à l'utilisateur de maîtriser l'état de son espace de travail privé par rapport à la base partagée.

Différents systèmes de visualisation ont été développés pour fournir aux utilisateurs une conscience permanente de l'état de leur espace de travail par rapport à l'espace partagé et aux espaces de travail des autres utilisateurs. Ainsi, les conflits potentiels peuvent être détectés plus rapidement. Contrairement aux autres environnements du même type, un utilisateur est informé des modifications faites à un objet dans l'espace privé d'autres utilisateurs. Un conflit potentiel entre deux utilisateurs modifiant ou ayant modifié le même objet peut être détecté avant son écriture dans la base partagée. La figure 5.1 donne un aperçu de l'environnement de l'utilisateur. On y trouve les différents outils permettant d'avoir des vues différentes de l'espace de travail de l'utilisateur. Les couleurs associées aux objets sont des indicateurs de leur état par rapport à la base publique et aux bases privées des autres utilisateurs. Dans cette configuration, il n'y a pas de support explicite à la coordination des utilisateurs. Ceux-ci sont conscients des objets modifiés par leurs collègues mais pas des activités qu'ils mènent effectivement. Les outils de communication intégrés dans la plate-forme (matérialisés entre autre ici par un système de messagerie instantanée) leur permettent de se coordonner de manière informelle. Ainsi, lorsqu'un conflit potentiel est détecté, une communication peut facilement être établie entre les "participants" à ce conflit pour leur permettre d'en comprendre les causes et d'en coordonner la résolution. En utilisant la plate-forme, les participants à un projet ont une conscience des objets modifiés par les autres utilisateurs mais pas de ce qu'ils font.

5.1.2 L'implantation des workflows flexibles

Le prototype MOTU a été étendu pour intégrer un service de coordination qui implante un moteur d'éxécution de workflow flexible comme celui décrit dans le chapitre précédent. Cette extension fournit une interface permettant la définition dynamique d'un procédé et son exécution. L'interface de définition permet également de visualiser l'état d'avancement de l'exécution du procédé. La notification est gérée par le système de messages qui fait partie du noyau de MOTU. La figure 5.2 montre les différentes vues dont dispose un utilisateur pour connaître son contexte par rapport à un procédé : son espace de travail courant, sa liste de projets, de tâches en cours et ses bons de travail et l'état courant du procédé.

Les utilisateurs peuvent ainsi connaître en permanence les projets auxquels ils participent, les tâches qui peuvent être démarrées et les tâches qu'ils ont commencées à exécuter. Un système de couleurs permet de différencier l'état des différentes tâches (prêt à anticiper, exécutable, exécution anticipée ou non). L'utilisateur peut également connaître en permanence l'état d'avancement du procédé grâce à une visualisation sous forme de graphe. Un éditeur permet de modifier le procédé au cours de son exécution. Les activités qui ne sont pas en cours d'exécution peuvent être modifiées. De nouvelles activités peuvent être ajoutées. L'affectation des tâches peut également être remise en cause.

Ce moteur de workflow a été défini de façon autonome au sein de la plate-forme. Il réutilise uniquement le service de d'événements pour assurer la synchronisation des vues des utilisateurs et le service de gestion des utilisateurs et de la sécurité. Il faut noter qu'outre la flexibilité du modèle, la présence d'outils permettant de visualiser l'état du procédé et le contexte de l'utilisateur pour l'ensemble des projets auxquels il participe, est particulièrement importante pour un environnement coopératif. L'utilisateur d'un environnement de travail coopératif doit disposer du maximum possible d'informations lui permettant de se situer dans les différents

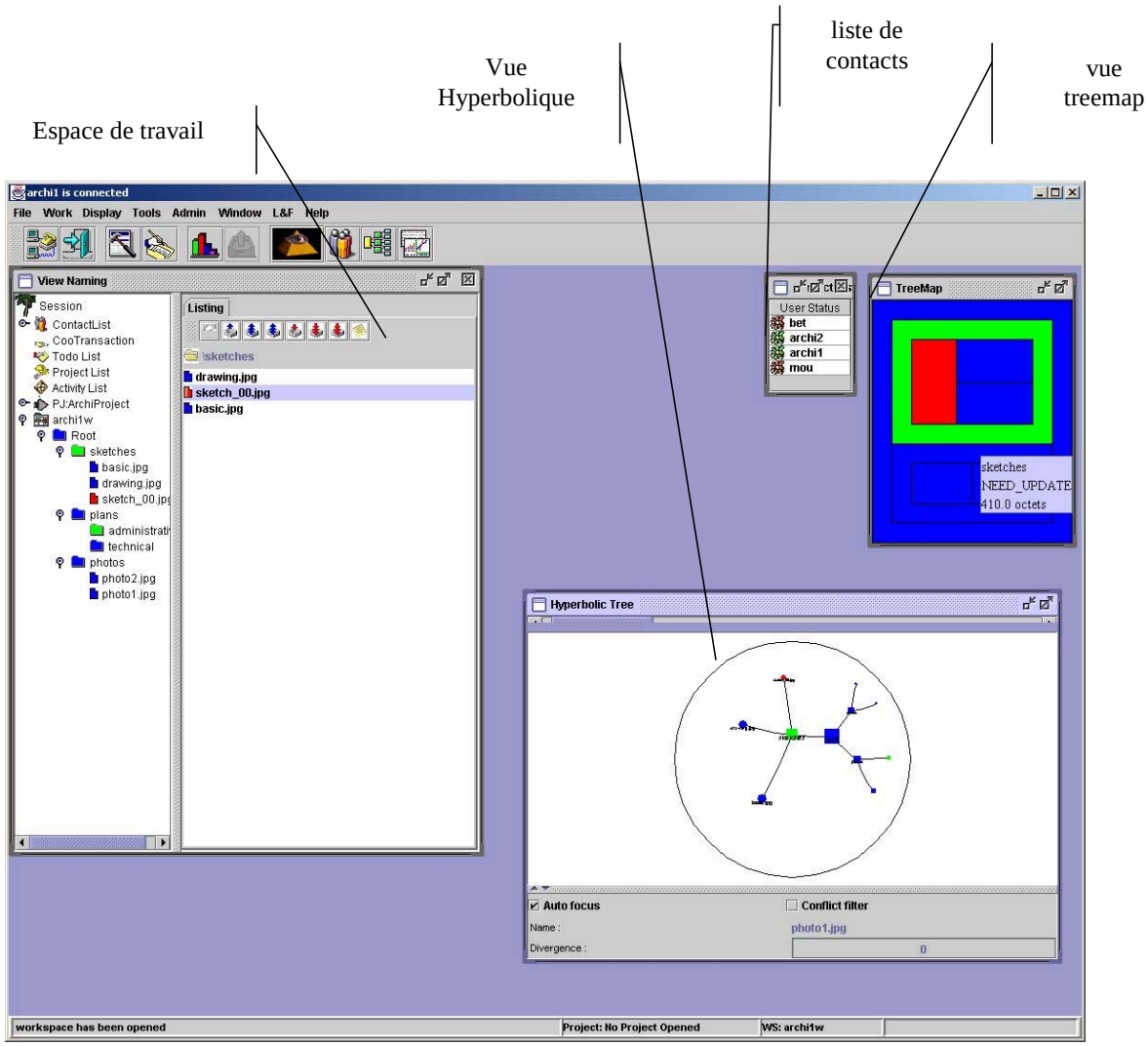


FIG. 5.1 – Interface utilisateur de MOTU

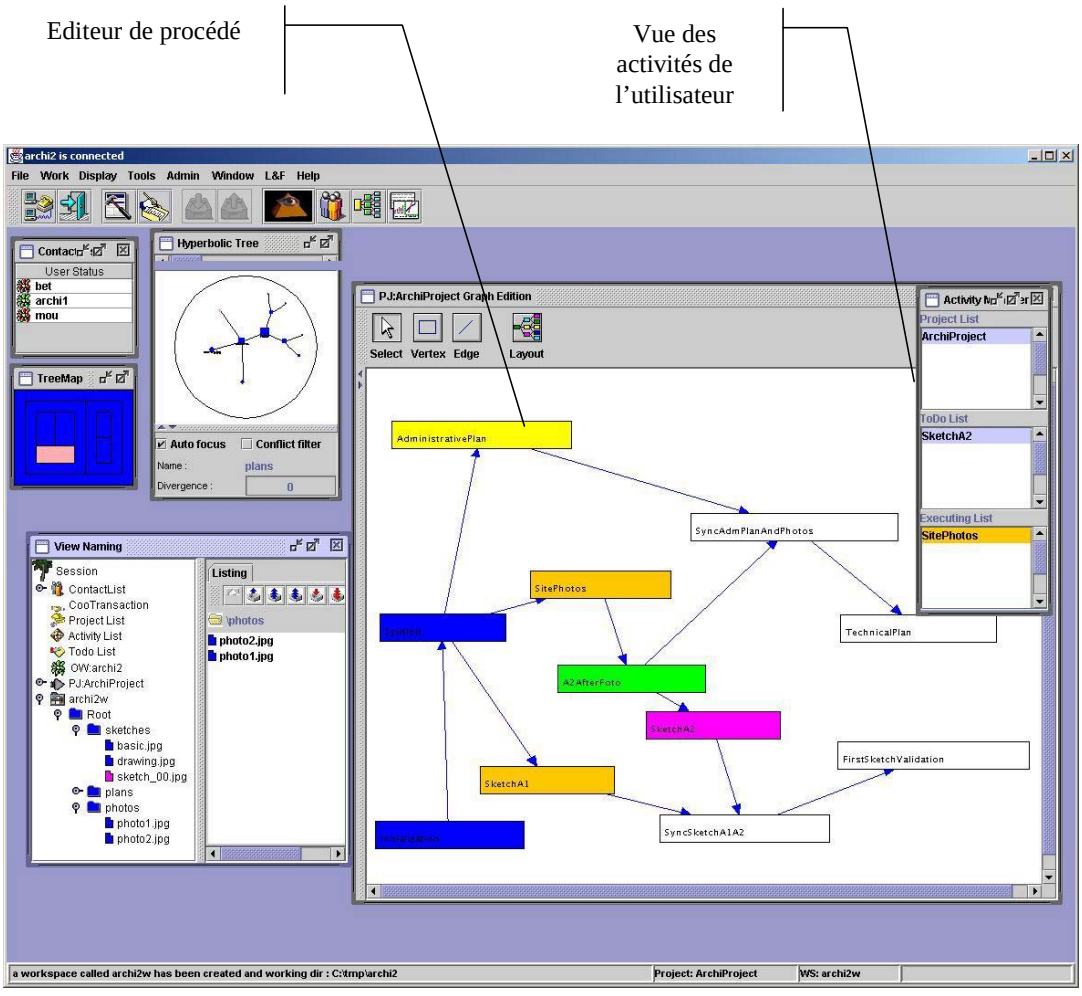


FIG. 5.2 – Interface incluant les outils de workflow

procédés auxquels il participe.

5.1.3 L'intégration des services de coordination

Le moteur de workflow a été développé sur la plate-forme MOTU indépendamment des services de gestion des données. Son intégration a été réalisée ensuite en ajoutant un type particulier d'activité associée à un espace de travail. Le cycle de vie des activités de ce type doit être maintenu de façon synchrone avec celui de leur espace de travail privé. La mise en œuvre faite ici reproduit en partie ce qui a été décrit dans le chapitre précédent concernant l'intégration du moteur de workflow flexible et les transactions coopératives. Ici, seules les données partagées sont prises en compte.

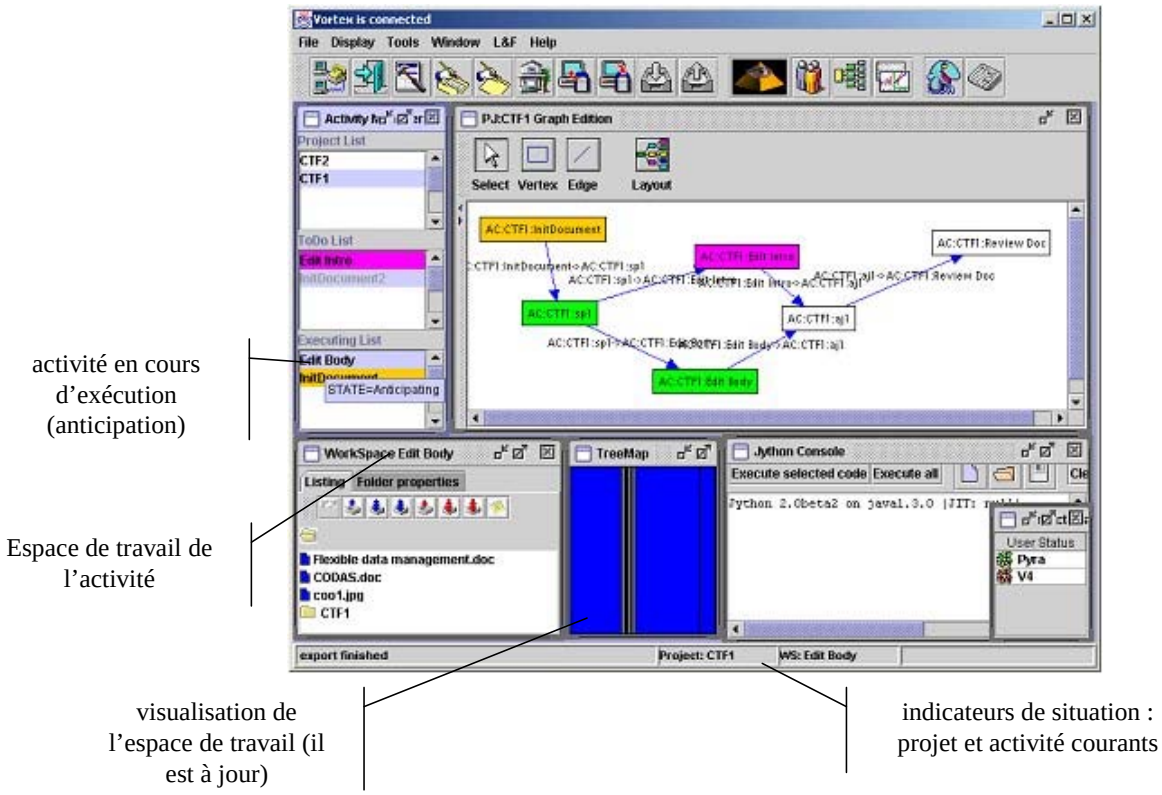


FIG. 5.3 – Intégration des services

Au démarrage d'une activité de ce type, un espace de travail est créé et initialisé avec le contenu de la base pour le projet. Ceci permet à l'utilisateur de pouvoir commencer son activité dans un environnement lui fournissant une base privée contenant tous les objets du projet. Cela implique que toutes les données partagées sont considérées comme lues. Cela peut engendrer plus facilement des conflits mais cela simplifie la tâche de l'utilisateur. A la fin de l'activité, les données modifiées durant son exécution sont recopiées dans la base commune du projet si elles sont à jour. Si elles ne sont pas à jour, l'activité ne peut pas se terminer. L'utilisateur doit se mettre à jour par rapport à la base avant de terminer. Les différents mécanismes permettant de connaître l'état des objets sont particulièrement utiles ici. L'utilisateur connaît son niveau

de synchronisation avec la base et avec les autres utilisateurs participant au même projet. La figure 5.3 illustre ces différents aspects. Elle montre une vue possible de l'utilisateur incluant :

- sa vue des différents projets, des tâches à faire et des tâches en cours ainsi que leur état,
- la vue du procédé sur lequel il travaille avec les tâches en cours d'exécution et celles qui peuvent être démarrées,
- deux vues de l'espace de travail, une correspondant au système de fichier et l'autre donnant une vue globale de l'état des objets. Ici, l'utilisateur dispose d'une version à jour de tous les fichiers.

L'intégration des deux services est utile de deux points de vue. Elle permet le contrôle de l'exécution des procédés et des échanges de données. Elle fournit une assistance à l'utilisateur permettant la sélection automatique de l'espace de travail correspondant à l'activité qu'il exécute et en lui fournissant en permanence l'information de contexte sur ce qu'il a à faire, l'état du projet et l'état des objets produits ou manipulés par les activités. En outre, la simplicité de l'interface, la flexibilité du contrôle et la possibilité de faire évoluer le procédé lui évitent de se sentir trop contraint.

5.1.4 Conclusion

Cette mise en œuvre partielle de la proposition dans un environnement incluant un ensemble de services intégrés de coopération nous a permis d'avoir une vision globale de la place des mécanismes de coordination et d'en mesurer l'intérêt et les limites. La possibilité de définir dynamiquement et de façon simple le procédé est apparue de façon évidente. De même, le fait pour les utilisateurs d'avoir une connaissance non seulement de l'état de leurs tâches mais de l'état d'avancement du procédé est fondamentale. En effet dans un contexte coopératif, la coordination explicite ne peut pas être seulement un système de contrôle. Elle doit être surtout un support au travail de groupe. Ces aspects ne peuvent être capturés que lors de la mise en œuvre et de tests mêmes limités de l'environnement. La possibilité d'anticiper l'exécution d'activité est apparue également comme importante. Elle permet effectivement d'augmenter le parallélisme entre les activités au cours du procédé. En outre, les utilisateurs ont conscience du fait qu'une activité n'est pas dans un état d'exécution normal. Les états anticipables ou en cours d'anticipation sont clairement dénotés dans l'interface. L'utilisateur a ainsi conscience du statut de l'exécution qu'il entreprend et des données qu'il va utiliser et de la priorité qu'il faut lui donner. Cette liberté d'analyser la situation et de disposer d'un nombre plus important d'alternatives apparaît de façon claire lors de l'exécution de scénarios sur ce prototype. Le "carcan" du contrôle de procédé est moins lourd.

Le manque de fiabilité, la complexité et l'architecture primitive du système ne permettaient pas de tests plus larges permettant de valider plus avant ces premières constatations. Dans la section suivante, nous décrivons succinctement une autre expérience qui reprend en partie les résultats décrits dans cette thèse dans le cadre d'un projet industriel : le projet Corvette.

5.2 Le projet Corvette

Le projet Corvette est une coopération avec Hitachi. Le but de ce projet était de voir comment WorkCoordinator, le système de gestion des workflow développé par Hitachi, pouvait être adapté pour exécuter des procédés coopératifs en utilisant les résultats produits dans l'équipe ECOO et en particulier ceux développés dans cette thèse.

5.2.1 WorkCoordinator

WorkCoordinator est le système de workflow commercialisé par Hitachi. Il fournit les outils de définition de procédé et un moteur d'exécution. Les données du procédé sont stockées dans une base de données Oracle. WorkCoordinator fournit une interface Corba permettant son intégration dans l'environnement d'exécution.

Le modèle de WorkCoordinator est différent de celui que nous avons développé. Il est composé de trois niveaux : les procédés, les activités et les tâches (work items). Les deux premiers niveaux sont classiques : un procédé est défini par un ensemble d'activités et par le flot de contrôle entre ces activités. On retrouve les mêmes opérateurs permettant la synchronisation des activités. Une activité peut être réalisée par une ou plusieurs tâches. Les tâches sont affectées aux utilisateurs ou peuvent être réalisées par des outils. Une activité est terminée quand toutes ses tâches sont terminées et que ses conditions de sortie sont vérifiées. La décomposition des activités en tâches est un aspect très particulier de WorkCoordinator. Pour des raisons de simplicité, nous n'avons pas pris en compte cet aspect dans la conception du prototype. Nous avons imposé qu'il n'y ait qu'une tâche par activité pour nous ramener à un modèle plus traditionnel.

Les autres fonctions de WorkCoordinator sont classiques : gestion des listes de tâches et contrôle de l'exécution du procédé. En revanche, WorkCoordinator ne fournit pas de mécanismes permettant une visualisation de l'évolution du procédé mais une simple interface d'administration et un outil de définition. La définition des procédés est compilée et les procédés sont instanciés à partir de cette définition. En cas d'erreur ou d'exception, c'est à l'administrateur de traiter les problèmes manuellement.

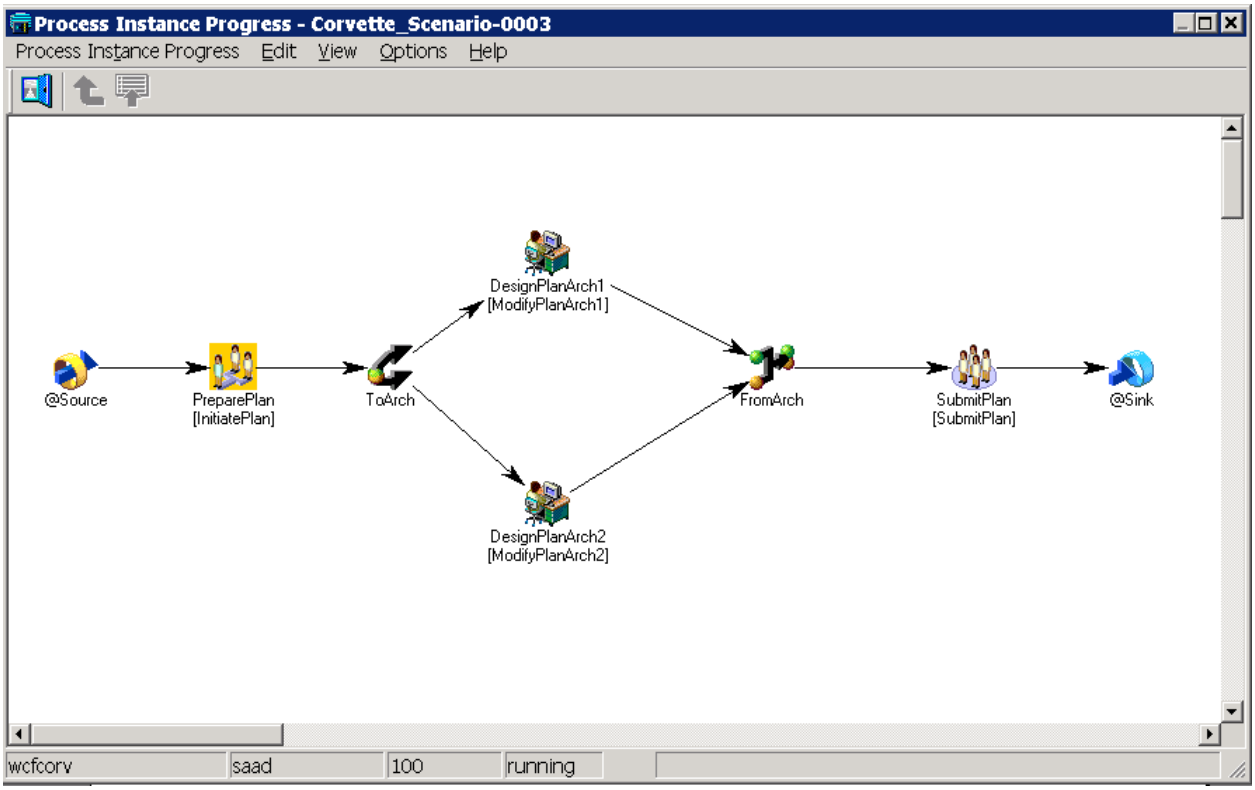


FIG. 5.4 – Définition d'un procédé WorkCoordinator

WorkCoordinator est un bon exemple de ce que l'on peut trouver dans le monde des systèmes de gestion des procédés de production. Les aspects mis en avant sont la fiabilité grâce à l'utilisation de technologies de base éprouvée, l'efficacité, pour permettre l'exécution d'un nombre de procédés important et les capacités d'intégration pour permettre l'adaptation du système à un système d'information existant.

5.2.2 Le prototype Corvette

Le prototype Corvette intègre l'implantation MOTU et Work Coordinator. C'est une intégration ad hoc en raison de l'impossibilité de modifier le système de workflow. Le but de ce prototype était d'intégrer le système de gestion des activités et les transactions coopératives Coo.

Les procédés sont décrits en utilisant l'interface de WorkCoordinator (cf figure 5.4). L'exécution du procédé est contrôlée par le système de workflow. Un procédé est associé à un espace de travail partagé entre les différents intervenants du procédé. Chaque fois qu'une activité est démarrée, une nouvelle transaction coopérative est créée. Le problème consiste ensuite à assurer la terminaison simultanée de l'activité et de la Coo transaction en respectant à la fois les contraintes du procédé et le protocole de transactions. Ce protocole a été adapté pour garantir la synchronisation entre la terminaison d'un groupe d'activité et des Coo transactions correspondantes dans le cas d'une terminaison de groupe. Cette adaptation particulière est due principalement au fait qu'il n'était pas possible d'intervenir sur WorkCoordinator. Le protocole mis en œuvre n'est d'ailleurs pas sûr. Il existe un cas où la synchronisation entre le cycle de vie des activités et des Coo Transactions ne peut pas être garantie. Le protocole de terminaison d'un groupe doit garantir que toutes les activités et toutes les Coo Transaction correspondantes sont prêtes à terminer pour terminer l'ensemble de façon atomique. Or, il n'est pas possible de vérifier que les activités peuvent terminer sans les terminer effectivement. Si la terminaison d'une activité échoue, il n'est donc pas possible de revenir à l'état initial où toutes les activités du groupe sont actives. Pour la terminaison d'un groupe, nous pouvons seulement garantir que si toutes les activités terminent, alors toutes les Coo-Transactions terminent aussi mais pas l'inverse.

La mise en œuvre est illustrée par la figure 5.5. Le démarrage d'une activité dans WorkCoordinator est notifié au serveur MOTU par un trigger Oracle. Cela déclenche la création d'une Coo Transaction. Une Coo Transaction dans MOTU est un espace de travail dont le cycle de vie est contrôlé en respectant le protocole Coo. La terminaison de l'activité est contrôlée par le client par qui doivent passer les commandes de l'utilisateur. C'est l'utilisateur qui décide de la terminaison d'une activité. Une condition supplémentaire de terminaison de l'activité est le respect du protocole par la Coo Transaction correspondante. Cette terminaison échoue si ce n'est pas le cas.

Les interfaces développées pour ce prototype ne sont pas aussi sophistiquées que pour MOTU, ni aussi intégrées en raison des contraintes particulières du projet. L'interface permettant de visualiser l'état des différentes activités en cours d'exécution est celle de WorkCoordinator (cf. figure 5.6).

L'outil permettant de contrôler l'exécution des Coo-Transactions (figure 5.7) a été adapté pour assurer la synchronisation entre celles-ci et les activités de Work Coordinator.

La création des Coo-Transactions est synchronisée avec le démarrage des activités. La terminaison d'une activité est décidée par l'utilisateur. C'est à ce moment que sont effectuées les vérifications du protocole Coo. Le succès ou l'échec de la terminaison de l'activité dépend de l'état des différentes Coo-Transactions.

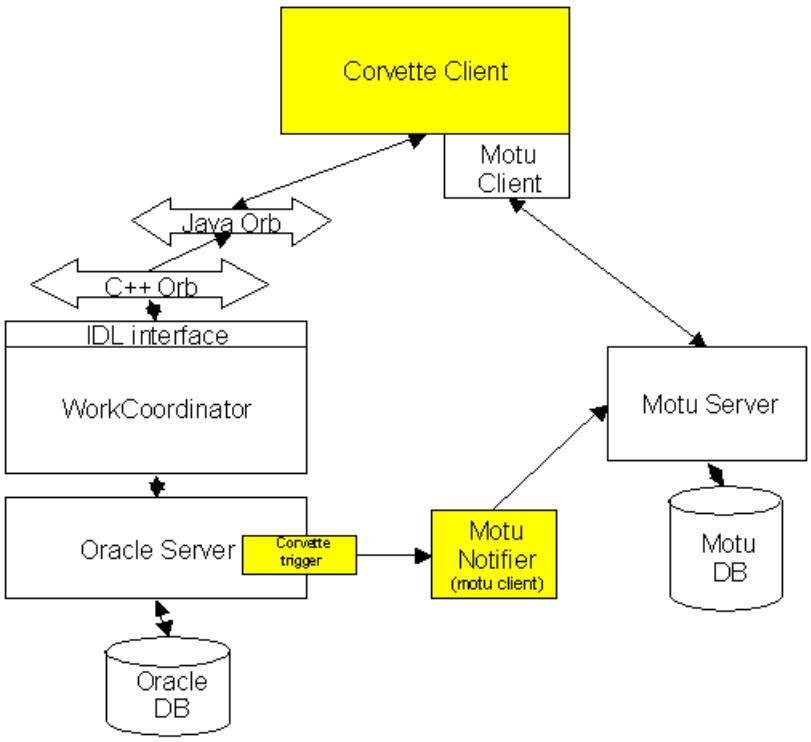


FIG. 5.5 – Architecture de Corvette

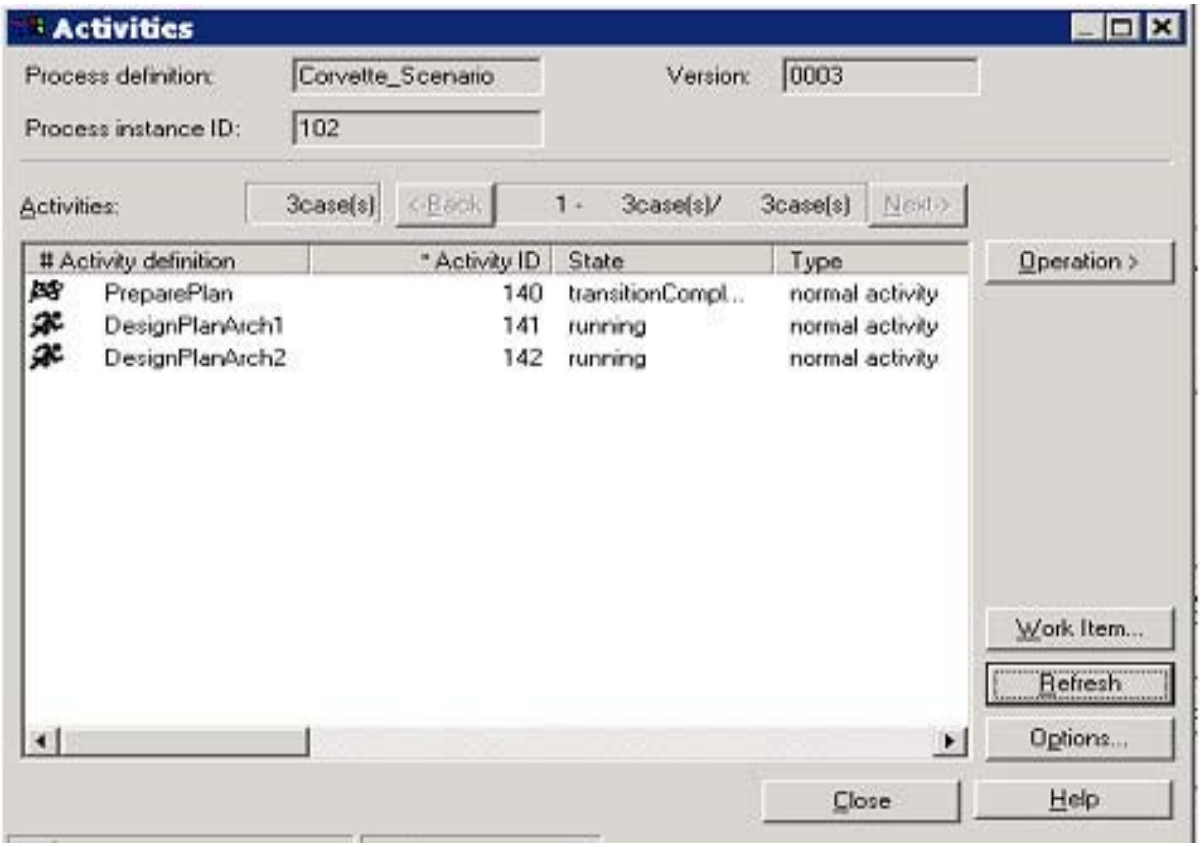


FIG. 5.6 – Interface d'administration d'un procédé de WorkCoordinator

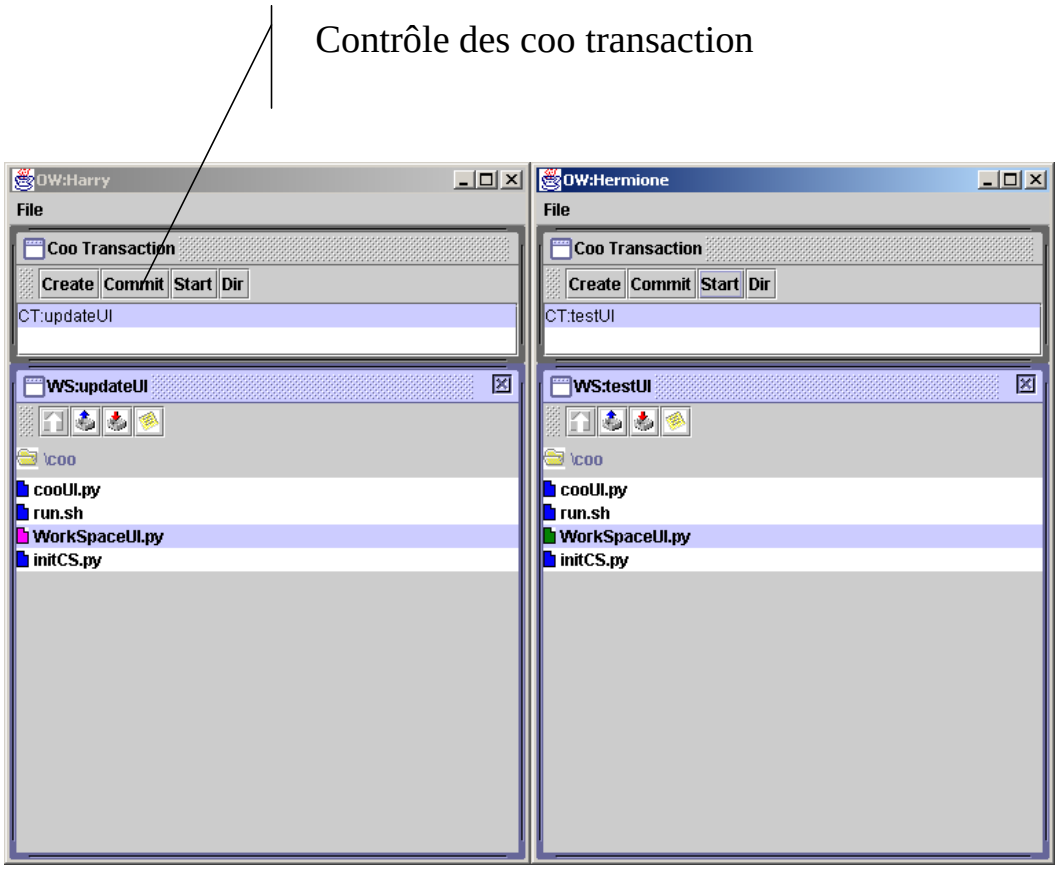


FIG. 5.7 – Vue des coo transactions

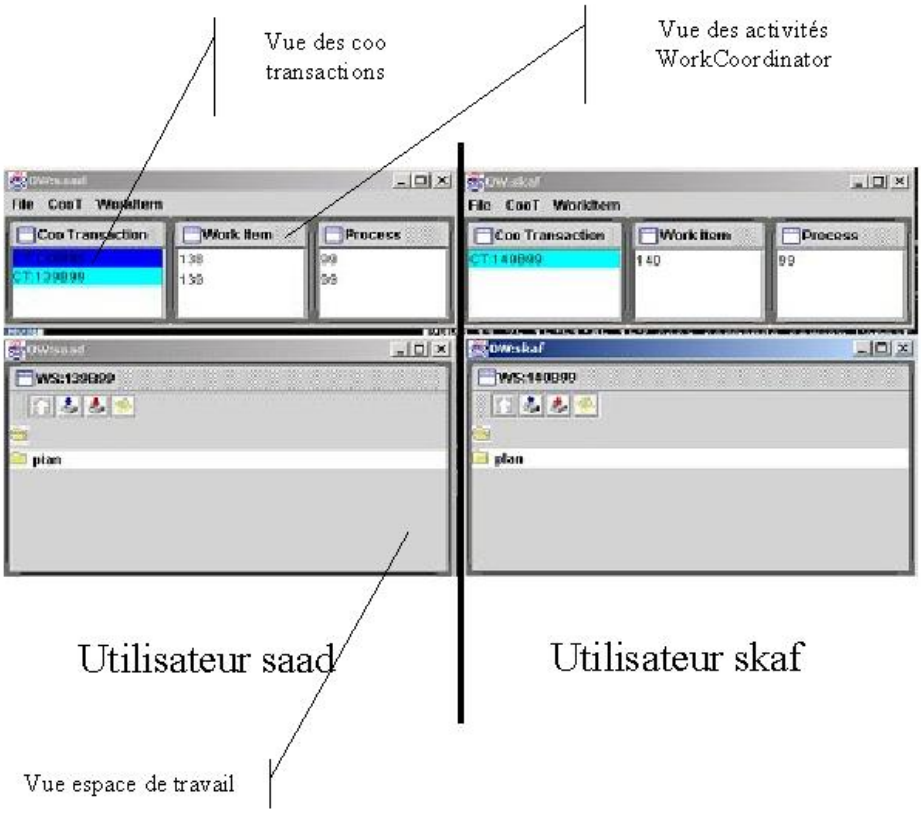


FIG. 5.8 – Vue intégrée de Corvette

5.2.3 Conclusion

L'intégration d'un système de workflow et d'un système de transaction coopérative est intéressante parce qu'elle fournit directement aux utilisateurs un environnement générique de gestion de leur données. Le système est utilisable directement et de façon intégrée. Les utilisateurs n'ont qu'à définir le procédé. Il n'ont pas besoin dans ce cas de réaliser le travail d'intégration classique avec d'autres systèmes d'information comme cela peut être le cas pour des Workflows de production. C'est une approche envisageable pour des environnements de production de logiciels ou de documents. Dans l'environnement mis en œuvre, les utilisateurs sont toujours sûrs de disposer des objets dans la version correspondante à l'activité qu'ils sont en train de réaliser. La correspondance entre activité et espace de travail privé, sous le contrôle du protocole Coo leur garantit la validité des données qu'ils manipulent. Il faut cependant noter que WorkCoordinator est trop rigide pour répondre aux besoins d'un procédé coopératif. Le procédé ne peut pas être changé simplement au cours de son exécution. L'enchaînement des activités doit suivre strictement la définition originale du procédé. Cette implantation montre clairement qu'il ne suffit pas de pouvoir échanger des données de façon flexible. Le modèle d'exécution rigide de procédé est une limite importante du système.

5.3 Conclusion

Les différentes mises en œuvre nous ont permis d'apporter un début de validation expérimentale des propositions faites dans cette thèse concernant le problème de la coordination d'activités coopératives. Il nous est apparu clairement que la définition d'un modèle est une étape qui doit être complétée par la mise en œuvre d'outils adaptés aux pratiques des utilisateurs. Imposer un système de coordination formel à un groupe d'utilisateurs impliqués dans un projet coopératif n'est pas simple. L'environnement de travail doit également apparaître comme un support à l'organisation et à la gestion des données. En outre, tous les moyens appropriés doivent être mis en œuvre pour permettre aux participants à un projet de se situer au cours de l'exécution de celui-ci. Les outils de coordination en sont un important.

La validation expérimentale d'un environnement de coopération est un processus complexe. Nous envisageons de faire une analyse d'usage et d'étudier le role des outils de coordination dans le contexte de la platforme coopérative.

Chapitre 6

Composant d'analyse et de prédiction des exceptions

6.1 Introduction

Dans cette section nous présentons un ensemble d'outils qui permet d'analyser, prédire et prévenir les exceptions. Nous définissons une exception comme une déviation par rapport à l'exécution optimale ou acceptable du procédé. Ceci est une définition de l'exception de haut niveau, orientée utilisateur qui permet aux concepteurs ou administrateurs des procédés de définir ce qu'ils considèrent comme « exception » et ainsi, le problème qu'ils veulent analyser ou éviter. L'analyse des exceptions peut en effet aider l'analyste du procédé à déterminer les causes d'une exception. Par exemple, l'analyse peut prouver que les retards dans le procédé de développement d'un logiciel se produisent chaque fois qu'une sous-équipe particulière spécifique est impliquée, ou chaque fois que, pour des composants complexes, des participants de plus de trois sites sont impliqués.

La compréhension des causes des exceptions peut aussi aider l'équipe et le responsable à identifier les changements nécessaires pour éviter la répétition de ces exceptions. Par exemple, le responsable peut décider d'arrêter la collaboration avec une sous-equipe donnée.

Des outils adaptés peuvent également permettre de prédire l'occurrence d'exceptions au moment de l'instanciation du procédé, et raffiner progressivement la prédiction au fur et à mesure que l'exécution du procédé avance et que plus d'information devient disponible. Par exemple, ils peuvent prédire qu'une certaine instance du procédé est susceptible de ne pas respecter sa date limite. La prédiction des exceptions est utile pour formuler les attentes correctes au sujet de la qualité de l'exécution du procédé. En outre, elle peut permettre d'agir afin d'éviter l'occurrence d'exceptions. Par exemple, quand l'outil prédit qu'une instance de procédé a une probabilité très élevée de manquer sa date-limite, la priorité de l'instance du procédé (selon l'importance du procédé et les dommages potentiels provoqués en manquant la date-limite) peut être augmentée automatiquement afin d'informer les acteurs impliqués que les activités de cette instance devraient être exécutées en priorité.

Notre approche est basée sur des techniques de fouille de données[17] et d'entrepôt de données[18] appliquées aux données de l'historique des procédés. En fait, les systèmes de gestion de Workflow enregistrent tous les événements importants qui se produisent pendant l'exécution des procédés, y compris la date de démarrage et de terminaison de chaque activité, les données d'entrée et

[17] data mining
[18] data warehouse

de sortie, la ressource qui l'a exécutée, et n'importe quelle panne qui s'est produite pendant l'exécution de l'activité ou du procédé. En agrégeant les données de l'historique dans un *data warehouse* et en les analysant avec des techniques de fouille de données, nous pouvons extraire les connaissances liées aux circonstances dans lesquelles une exception s'est produite dans le passé, et utiliser ces informations pour expliquer les causes de son occurrence aussi bien que pour prédire des occurrences futures dans des instances de procédés courants ou à venir.

Les autres problèmes intéressants qui peuvent être considérés en utilisant la même méthodologie sont : la découverte de la définition de procédés, l'attribution intelligente des activités aux ressources et l'identification automatique des besoins systèmes basée sur la prévision de la charge de travail. Dans la suite de ce chapitre, nous nous référerons à cette thématique de recherche en tant qu'*intelligence de procédé*, ou IP. Les questions principales à traiter sont la caractérisation du problème, l'identification des technologies qui peuvent supporter notre effort, et l'utilisation et la composition de ces technologies dans une architecture globale

Notre but est de concevoir et développer la suite d'outils d'intelligence de procédé comme solution de « prêt-à-utiliser ». En tant que telle, elle doit être applicable dans différentes conditions et doit être capable de répondre à un ensemble large de besoins.

Le problème spécifique de l'analyse, de la prédiction, et de la prévention des exceptions nécessite de relever les difficultés supplémentaires suivantes :

- définir ce qui caractérise une exécution de procédé « normale » par rapport à une exécution « exceptionnelle ». Cette définition change selon les besoins particuliers des utilisateurs et nous devons pouvoir analyser et prévoir un large éventail de situations ;
- déterminer comment exactement les exceptions peuvent être analysées et prédites. En particulier, nous devons identifier quelles techniques d'analyse/fouille des données peuvent être appliquées, et quels attributs du procédé (caractéristiques) doivent être fournis comme entrée aux algorithmes d'analyse de données.
- le problème de la prédiction des exceptions est particulièrement complexe, puisque, idéalement nous voulons faire les meilleures prédictions à chaque moment pendant l'exécution du procédé . Par conséquent, nous devons établir les modèles prédictifs conçus pour différentes étapes dans l'exécution du procédé (avant et après l'exécution de chaque activité).

Dans cette section nous détaillons ces difficultés, nous discutons comment les aborder et nous décrivons des techniques et des outils pour l'analyse, la prédiction, et la prévention des exceptions. Nous commençons par la présentation des travaux similaires. Dans la section 6.3 nous décrivons brièvement l'historique maintenu par un système de gestion de workflow. Nous présentons ensuite l'approche globale IP et l'architecture du système. Nous proposons et discutons une approche pour l'analyse des exceptions, puis présentons notre mise en oeuvre. Ensuite, nous nous concentrons sur le problème de la prévention et prédiction des exceptions. Dans la section 6.7, nous illustrons notre approche par des résultats expérimentaux. Dans la partie conclusion, nous traçons les grandes lignes des problèmes non résolus.

6.2 Approches similaires

A notre connaissance, il n'existe pas d'approche d'analyse et de prédiction des exceptions basées sur des techniques de fouille de données et de *data warehouse*.

Des travaux antérieurs existent dans le domaine de la prédiction des exceptions, mais qui sont toutefois limités à l'estimation des risques de dépassement de date-limite et ils sont basées sur des techniques statistiques simples. Nous récapitulons ces contributions, et par la suite nous soulignons les différences principales avec l'approche que nous proposons.

Une des premières contributions pour la gestion du temps du procédé de workflow est fourni dans [Pan97a], et se base sur des travaux dans le domaine des systèmes en temps réel. Les auteurs proposent de prédire dès que possible quand une instance de procédé est susceptible de dépasser sa date-limite, afin de prendre les mesures appropriées. Dans le modèle de procédé proposé, chaque activité dans le procédé a une durée maximale, assignée par le concepteur du procédé basée sur l'estimation de la durée de l'activité et sur la nécessité de respecter la date-limite du procédé global. Quand la durée maximale est dépassée, une exception est générée qui suspend le procédé. Quand une activité s'exécute plus rapidement que sa durée maximale, un temps de réserve devient disponible. Il peut être utilisé pour ajuster dynamiquement la durée maximale des activités suivantes. Une activité peut prendre toute la réserve disponible ou une partie proportionnelle à son temps d'exécution estimé ou au coût associé à l'impact en cas d'expiration de sa propre date-limite.

L'approche présentée dans [Pan97b] vise également à estimer la durée d'un procédé et des expirations de date-limite. Les estimations sont obtenues en tenant compte de l'état de l'instance du procédé et des statistiques recueillies pendant des exécutions passées. En particulier, le temps de terminaison de l'activité est estimé en se basant sur la charge actuelle des ressources du système et la durée d'exécution moyenne de l'activité, calculée en fonction des exécutions passées. Les temps d'exécution des activités estimés sont alors utilisés pour estimer la durée de l'instance de procédé.

Dans [Ede99] les auteurs proposent une technique différente pour la surveillance et la gestion des dates limites. Dans l'approche proposée, la définition d'un procédé inclut la spécification de la durée prévue pour chaque activité. Cette durée peut être définie par le concepteur du modèle ou être déterminée à partir des exécutions passées. En plus, le concepteur peut définir des dates limites pour des activités ou pour le procédé entier. La date limite indique le temps *au plus tard* pour les activités et pour le procédé, défini comme le temps écoulé depuis le démarrage de l'instance de procédé. Les procédés sont traduits en diagramme PERT qui montre, pour chaque activité, basée sur les durées prévues des activités et sur les dates limites définies, le temps *au plus tôt* de l'activité (c'est à dire la date à laquelle elle peut terminer) et la date *au plus tard* à laquelle elle doit terminer pour satisfaire les contraintes de date limite. Pendant l'exécution d'une instance de procédé, en connaissant : l'instant de temps actuel, la durée prévue d'une activité, et la date *au plus tard* de terminaison, l'avancement de l'instance de procédé peut être évaluée par rapport à sa date limite. Cette information peut être utilisée pour alerter les administrateurs de procédé au sujet du risque de retard et pour informer les utilisateurs au sujet de l'urgence de leurs activités.

Notre approche diffère considérablement de celles présentées ci-dessus. En fait, nous visons à prédire n'importe quel type d'exception, plutôt que de nous concentrer sur des expirations de dates-limite. En outre, nous proposons d'établir des modèles de prédiction en utilisant des techniques de *data warehouse* et de fouille de données, qui permettent des prédictions plus précises, basées sur un nombre important de caractéristiques des instances de procédé, tels que les valeurs des données, les ressources, le jour de la semaine ou l'heure du jour où les procédés ou les activités sont démarrées, etc.

En plus, l'approche présentée dans cette section permet également l'analyse des exceptions afin d'aider les utilisateurs à comprendre et à éliminer les causes de l'exception. Une autre différence par rapport aux approches mentionnées ci-dessus est que nous présentons également l'architecture et la mise en place d'une suite d'outils pour l'analyse et la prédiction des exceptions. Cette approche et l'ensemble d'outils seront réutilisés et étendus pour fournir plus de fonctionnalités pour l'optimisation des procédés.

Finalement, nous mentionnons également le travail de Hwang et al. [Hwa99], qui traite égale-

ment le problème des exceptions par l'analyse des traces d'exécutions des procédés. Cependant, le but des auteurs se limite à aider les utilisateurs dans le traitement des exceptions une fois qu'elles se sont produites. Des suggestions pour le traitement des exceptions sont données en fournissant aux utilisateurs des informations sur la manière dont des situations semblables ont été traitées dans les exécutions précédentes du procédé. Notre but est différent, puisque nous visons à analyser les exceptions pour comprendre pourquoi elles se sont produites, pour prédire et prévenir leur occurrence, plutôt que leur traitement.

6.3 Historique dans un système de gestion de Workflow

Cette section fournit une brève introduction à la structure de l'historique d'un système de gestion de workflow. Il existe des centaines de systèmes de gestion de workflow commerciaux disponibles sur le marché et beaucoup de prototypes de recherches. Si chaque système a son propre modèle de procédé et sa propre structure d'historique, la plupart d'entre eux partagent les mêmes concepts de base. Dans cette section nous présenterons la structure de l'historique du système de gestion de workflow HPPM (HP Process Manager), sur lequel nous avons travaillé et entrepris les expériences décrites dans cette section. Cependant, les mêmes concepts et techniques se retrouvent pratiquement dans n'importe quel autre système de gestion de workflow.

Les systèmes de gestion de workflow enregistrent les informations sur l'exécution des procédés, noeuds et activités, dans une base de données d'audit, typiquement enregistrée dans un système de gestion de bases de données relationnelles. La base de données de l'historique inclut généralement les informations suivantes :

- instances de procédés : le moment de démarrage et de terminaison, l'état actuel d'exécution (par exemple, commencé, suspendu, terminé), et le nom de l'utilisateur qui a commencé l'instance de procédé ;
- instances d'activités : les moments de démarrage et de terminaison, l'état actuel d'exécution, et le nom de la ressource qui a exécuté l'activité ;
- instances de noeuds : les moments de démarrage et de terminaison et l'état d'exécution pour l'exécution la plus récente du noeud dans une instance de procédé (les exécutions multiples sont possibles si le procédé contient des boucles) ;
- modifications de données : nouvelle valeur pour chaque donnée élémentaire chaque fois qu'elle est modifiée.

La figure 6.1 montre les tables de l'historique et leurs relations. Les attributs soulignés dénotent les clés primaires, alors que les liens entre les tables dénotent des contraintes principales étrangères. Pour la clarté de la présentation, dans la figure nous avons légèrement simplifié la structure des tables. En plus des tables d'exécution des instances, les utilisateurs autorisés peuvent accéder aux tables de définition, qui décrivent les procédés, les noeuds, les activités, et les ressources définies dans le WfMS. Une description complète et détaillée du schéma de base de données de l'historique de de HPPM est fournie dans [Hew00b].

6.4 Architecture du composant d'intelligence de procédé

Cette section présente l'architecture globale de l'ensemble d'outils d'intelligence de procédé, pour présenter l'environnement dans lequel la solution pour l'analyse, la prédiction, et la prévention d'exceptions a été développée. La suite d'outils d'intelligence de procédé comprend trois parties principales : le *data warehouse* des données de définition et d'exécution des procédés, le moteur d'IP et le gestionnaire de surveillance et d'optimisation, ou GSO (voir la figure 6.2).

FIG. 6.1 – Schéma de la base de donnée de l'historique

6.4.1 Data Warehouse

Les données sont périodiquement extraites à partir de l'historique d'audit de WfMS et chargées dans le *data warehouse* par des scripts d'extraction, de transfert, et de chargement (ETC). Les scripts nettoient les données en supprimant l'information incohérente et corrompue, souvent présente dans les historiques de WfMS. En plus, le *data warehouse* est conçu pour supporter des transactions analytiques (OLAP[19]) très performantes et l'analyse multidimensionnelle des données d'exécution de procédés qui peuvent provenir des sources hétérogènes. Par conséquent, le *data warehouse* des données d'exécution de procédés est un composant très utile en soi, fournissant un éventail de fonctionnalités de rapports (analyse) qui manquent dans les WfMSs commerciaux. Une présentation détaillée du *data warehouse* du composant d'intelligence de procédé peut être trouvée dans [Bon01].

6.4.2 Moteur d'IP

Le moteur d'IP exécute des algorithmes de *data mining* sur les données du *data warehouse* afin de :

- comprendre les causes des comportements spécifiques, tels que l'exécution de certains chemins dans une instance de procédé, l'utilisation d'une ressource, ou l'incapacité de respecter le contrat du niveau de la qualité du procédé ;
- générer des modèles de prédiction, c'est à dire l'information qui peut être utilisée pour prédire le comportement et les performances d'exécution des instances de procédé, des ressources, et du WfMS.

[19] On-Line Analytical Processing

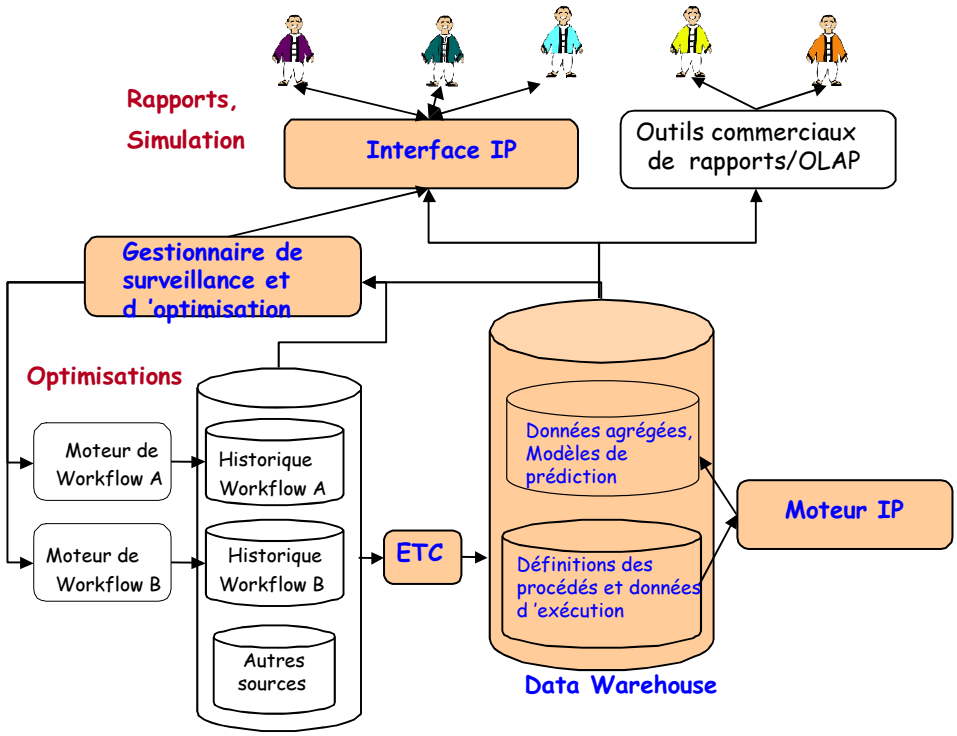


FIG. 6.2 – L'architecture globale du composant IP

Le moteur d'IP enregistre l'information extraite également dans le *data warehouse*, de sorte que l'information puisse être consultée facilement et efficacement par la console d'IP ou par des outils externes d'OLAP ou générateurs de rapports.

6.4.3 Gestionnaire de surveillance et d'optimisation

Le gestionnaire de surveillance et d'optimisation (GSO) utilise aussi bien les informations contenues dans le *data warehouse* que des informations sur les instances de procédés courants enregistrées dans l'historique du WfMS pour faire des prédictions et pour optimiser dynamiquement l'exécution des instances de procédés. Par exemple, le gestionnaire peut être configuré pour augmenter la priorité d'une instance de procédé quand il y a une probabilité élevée pour que l'instance ne respecte pas la date limite. Le GSO peut également alerter les administrateurs du procédé et du système quand il prédit des situations critiques.

6.5 Analyse des exceptions

Après cette brève introduction de l'approche d'intelligence de procédé, nous nous concentrons maintenant sur les exceptions. Cette section décrit notre approche pour aider l'administrateur du procédé et les utilisateurs à comprendre les causes des exceptions.

Nous commençons la section en donnant une définition de la notion d'« exception » et en montrant comment les exceptions sont décrites. Ensuite nous fournissons une vue d'ensemble de notre approche pour l'analyse de ces exceptions, pour présenter ensuite les détails et la description de la mise en oeuvre.

6.5.1 Notion d'exception

L'utilisateur est libre de déclarer qu'un événement ou une situation donnée est « exceptionnelle ». Dans la mise en oeuvre actuelle nous permettons la définition des types d'exceptions suivants :

- une date-limite de procédé ou d'une activité expire,
- la durée d'exécution d'un procédé, d'un noeud dépasse une valeur indiquée,
- la durée d'exécution d'un procédé, d'un noeud dévie de la durée moyenne de n fois l'écart type (n peut être n'importe quel nombre positif ou négatif),
- un noeud ou une activité indiquée est lancé plus que, moins que, ou exactement n fois dans une instance de procédé (n peut être n'importe quel nombre entier non négatif). Ce type d'exécution est particulièrement utile pour déterminer des exécutions du procédé non-optimal, car ces exécutions sont souvent caractérisées par l'activation multiple d'un certain noeud.
- la valeur d'une donnée est supérieure, inférieure, ou égale à une valeur indiquée.

D'autres types d'exceptions peuvent être ajoutés si les utilisateurs veulent surveiller des comportements spécifiques non inclus dans la liste ci-dessus, pourvu bien sûr que leur occurrence puisse être détectée de manière déterministe en analysant l'historique de l'exécution, de sorte qu'il soit possible d'identifier automatiquement des instances de procédés « exceptionnels ». Comme nous le montrerons ci-dessous, les changements qu'on doit apporter à l'outil pour qu'il puisse manipuler un nouveau type d'exception sont minimaux.

Les définitions des exceptions sont enregistrées dans une table relationnelle dans le *data warehouse*, représentée dans la figure 6.3. La figure montre également un enregistrement qui

déclare l'intérêt pour l'analyse des exceptions se produisant quand les instances du procédé d'approbation de dépenses presenté dans la figure 6.5 prennent plus de 8 jours pour se terminer.

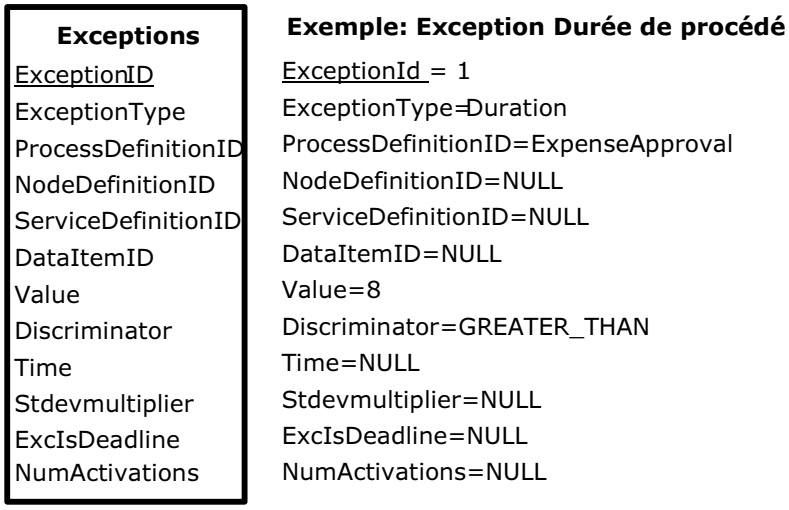


FIG. 6.3 – La table de définition des exceptions et un exemple d'exception

6.5.2 Analyse des exceptions - vue d'ensemble

L'analyse des exceptions est exécutée en appliquant des techniques de *data mining* aux données de définition et d'exécution des procédés, rassemblées dans le *data warehouse*. Spécifiquement, nous traitons ce problème comme un problème de classification. Les applications de classification fonctionnent comme suit : elles prennent comme entrée un ensemble de données d'apprentissage étiquetées (typiquement sous forme d'une table relationnelle) dans lequel chaque ligne (enregistrement) décrit un objet (par exemple, un client dans une application de gestion de client) et la classe à laquelle cet objet appartient (par exemple, client « profitable », « neutre », ou « non profitable ». Le classificateur produit alors un ensemble de règles de classification, qui associent une condition sur les attributs des objets à une classe. La signification d'une telle règle est le fait que les objets dont les attributs satisfont la condition appartiennent à la classe indiquée. Par conséquent, les règles de classification identifient les caractéristiques des objets dans chaque classe, en termes de valeurs des attributs des objets. Par exemple, le classificateur peut découvrir la règle de classification suivante : les clients de Nancy qui ont un salaire annuel de plus de 50,000ff sont « profitables ». Pour chaque règle de classification, le classificateur fournit également des informations sur l'exactitude de la règle, c'est à dire, sur la probabilité que la classification soit correcte.

Le problème d'analyse des exceptions peut être réduit à un problème de classification, où les instances de procédé sont les objets, qui appartiennent à la classe « normale » ou à la classe « exceptionnelle ». Les règles de classification identifient quelles sont les caractéristiques des instances de procédés « exceptionnelles ». Une fois que ces caractéristiques ont été identifiées, l'utilisateur peut avoir une meilleure compréhension des causes de l'exception, et peut alors essayer de traiter ces causes. Afin d'analyser pourquoi les instances d'un certain procédé ont généré une exception spécifique, nous proposons une approche en cinq phases :

- La phase de **préparation de données de procédé** choisit les attributs des instances de procédé qui vont être inclus en tant qu'éléments d'entrée pour le classificateur. Les attributs peuvent inclure par exemple : les valeurs des données de procédé aux différentes étapes pendant l'exécution de l'instance de procédé, le nom des ressources qui ont réalisé les activités dans l'instance de procédé, la durée de chaque activité et le nombre de fois qu'un noeud a été exécuté. Une fois les attributs d'intérêt identifiés, une structure de données (typiquement une table relationnelle) est créée et peuplée avec les données des instances de procédés.
- La phase d'**étiquetage** examine les instances de procédé dans le *data warehouse* et les étiquete comme « normales » ou « exceptionnelles » en se basant sur la définition de l'exception analysée.
- La phase de **préparation de l'analyse des exceptions** fusionne l'information produite par les deux phases précédentes dans un seul corpus qui peut être utilisé pour alimenter l'application de classification.
- La phase de **data mining** applique des algorithmes de classification aux données produites par la phase de préparation. En particulier, suivant une pratique courante pour aborder les problèmes de classification, les données sont divisées en un corpus d'apprentissage et un corpus de validation. Les données dans le corpus d'apprentissage sont utilisées pour générer des règles de classification, et donc pour identifier les caractéristiques des instances normales et, ce qui est le plus important, des instances exceptionnelles. La qualité des résultats peut alors être évaluée en examinant les instances dans le corpus de validation et en calculant les pourcentages des instances qui sont correctement classifiés par les règles.
- Finalement, dans la phase d'**interprétation**, l'analyste du procédé interprète les règles de classification pour comprendre les causes de l'exception, et en particulier pour identifier les problèmes et les inefficacités qui peuvent être traités et enlevés.

Quelques répétitions des phases « data mining → interprétation » peuvent être nécessaires afin d'identifier les règles de classification les plus intéressantes et les plus pertinentes. En particulier, la phase de *datamining* peut produire des règles de classification basées sur les attributs qui ne sont pas intéressantes dans le cas spécifique considéré. Par exemple, les règles de classification identifieront certainement une corrélation entre la durée de l'instance de procédé et une exception correspondant à l'expiration de la date-limite. Cependant, c'est une corrélation évidente et non intéressante. En conséquence, l'analyste peut vouloir répéter la phase de *data mining* et supprimer la durée de l'instance de procédé des attributs considérés pour générer des règles de classification, permettant au classificateur de se concentrer sur des attributs « intéressants ».

6.5.3 Détails et implantation

Cette section détaille notre approche pour l'analyse des exceptions et présente notre mise en oeuvre, développée sur Oracle 8i (la plupart des outils IP est écrite en SQL ou PL/SQL). Tous les modules d'analyse des exceptions font partie du composant moteur d'IP de la figure 6.2.

La première phase, la phase de préparation de données de procédé, pose des problèmes de modélisation de la table correspondante. En fait, les applications de classification exigent typiquement que les données d'entrée résident dans une table relationnelle, où chaque enregistrement décrit un objet spécifique. Par conséquent, pour analyser pourquoi une exception affecte les instances d'un procédé, nous devons préparer une table spécifique à chaque procédé (qu'on va l'appeler table d'analyse de procédé par la suite), qui inclut une ligne par instance de procédé, et où les colonnes correspondent aux attributs de l'instance de procédé. Une colonne supplémentaire, requise pour étiqueter des instances en tant que « normales » ou « exceptionnelles », sera

ajoutée par les phases ultérieures.

Cependant, à la différence des problèmes classiques de classification, les informations sur un objet (instance de procédé) dans le *data warehouse* du composant IP sont dispersées à travers des tables multiples, et chaque table peut contenir des lignes multiples liées à la même instance. Par conséquent, nous sommes confrontés au problème de définir une table appropriée d'analyse de procédé et de la peupler en rassemblant les données des instances de procédé.

En plus, même dans le même procédé, les différents instances peuvent avoir des attributs différents. Le problème est qu'un noeud peut être lancé un nombre de fois différent dans différentes instances. Le nombre de ces exécutions est inconnu *a priori*. Par conséquent, non seulement nous devons identifier quels sont les attributs intéressants de l'exécution d'un noeud à inclure dans la table d'analyse de procédé, mais également combien d'exécutions du noeud et lesquelles doivent être représentées.

Cette question peut être abordée de plusieurs façons : par exemple, nous pourrions décider que si un noeud peut être lancé plusieurs fois, alors nous considérons pour notre analyse seulement une exécution spécifique du noeud (par exemple, la première ou la dernière). Une approche alternative consiste à considérer toutes les exécutions du noeud. Dans ce cas-ci, la table de procédé d'analyse doit avoir, pour chaque noeud, un certain nombre de colonnes proportionnel au nombre maximum des exécutions de ce noeud, déterminées en regardant les données des instances de procédés dans le *data warehouse*. Cependant, malgré le fait que cette technique fournisse plus d'informations à la phase de *data mining*, elle ne donne pas nécessairement de meilleurs résultats. En fait, les tables d'analyse produites de cette façon incluent typiquement beaucoup de valeurs nulles (NULL), particulièrement si le nombre d'exécutions du noeud diffère considérablement d'une instance à l'autre. Les outils de *data mining* de données commerciales ne donnent pas de bons résultats pour les tables rares (avec beaucoup de valeurs nulles). En plus, quand les classifications sont basées sur un grand nombre d'attributs similaires qui ont souvent des valeurs nulles, il est très difficile d'interpréter et de comprendre les résultats. En conclusion, cette approche peut être très complexe et lourde du point de vue du calcul nécessaire si les noeuds sont exécutés beaucoup de fois.

L'approche que nous avons suivie consiste à insérer deux attributs (colonnes) pour chaque noeud qui peut être exécuté plusieurs fois : un pour représenter la première exécution et le second pour représenter la dernière exécution de ce noeud. Ce choix est dû à l'observation de plusieurs expériences que nous avons entreprises sur différents procédés. Les premières et les dernières exécutions d'un noeud dans le procédé ont une corrélation importante avec beaucoup de types d'exceptions de procédé, comme ceux liés au temps d'exécution du procédé et à l'exécution d'un sous-graphe donné dans le procédé.

En conclusion, nous remarquons que le nombre d'attributs intéressants pour nos objectifs peut être assez grand. Par exemple, une exception pourrait être liée au rapport entre les durées de deux noeuds dans le procédé, ou à la somme de deux données numériques du procédé. Dans notre mise en oeuvre nous avons configuré l'outil pour choisir les attributs qui ont montré les corrélations plus élevées aux exceptions dans les essais que nous avons réalisés. En particulier, la table d'analyse de procédé inclut les attributs suivants pour chaque instance de procédé :

- moment de démarrage et de terminaison : ceux-ci sont en fait décomposés en colonnes multiples correspondant à l'heure, au jour de la semaine, au jour du mois, etc., et à l'indicateur de vacances (pour dénoter si le procédé était instancié pendant des vacances) ;
- données du procédé : valeur initiale et finale des données élémentaires de procédé, plus la longueur (en octets) de chaque élément ;
- ressource qui a commencé l'instance de procédé ;
- durée de l'instance de procédé .

En plus, la table d'analyse de procédé inclut des attributs pour chaque noeud dans le procédé (deux ensembles d'attributs sont inclus pour les noeuds qui peuvent avoir des exécutions multiples, comme discuté ci-dessus) :

- moment de temps de démarrage et de terminaison (décomposés de la même manière que pour les instances de procédés),
- données élémentaires : valeur des données de sortie du noeud, plus la longueur (en octets) de chaque élément,
- ressource qui a exécuté le noeud,
- l'état final du noeud (par exemple, terminé, suspendu ou échoué),
- la durée du noeud,
- nombre d'activations du noeud dans l'instance de procédé (cet attribut est inclus seulement une fois par noeud, même si deux ensembles d'attributs sont utilisés pour ce noeud, puisque la valeur serait la même pour les deux).

La table de procédé d'analyse est automatiquement construite par un script PL/SQL, qu'on va appeler *script de préparation de l'analyse de procédé*. Ce script prend le nom du procédé à analyser comme paramètre d'entrée, et recherche l'information de définition de procédé dans le *data warehouse* du composant IP. En particulier, le script identifie les noeuds et les données élémentaires qui font partie du procédé, et définit le schéma de la table d'analyse de procédé (voir la figure 6.4). Une fois que la table a été créée, le script la peuple avec les données des instances de procédés. En plus, les utilisateurs peuvent indiquer un intervalle de date, pour peupler la table de procédé d'analyse seulement avec des instances démarrées dans l'intervalle.

La phase d'étiquetage est implantée par des scripts SQL qui remplissent une table qui ne dépend pas du procédé ou de l'exception analysée (appelée InstancesEtiquetées) indiquant quelles sont les instances de procédé « normales » et lesquelles sont « exceptionnelles » selon une certaine définition d'exception. Un script différent d'étiquetage est nécessaire pour chaque type d'exception, puisque la logique d'étiquetage dépend du type d'exception. Nous avons défini plusieurs scripts de classification pour les types d'exceptions supportées. Cependant, nous observons que les scripts de classification sont les seules parties du code qui dépendent de l'exception, et il est relativement facile de les écrire. Par conséquent, les utilisateurs peuvent étendre le système d'IP pour détecter et analyser d'autres types d'exceptions.

La phase de préparation de l'analyse des exceptions est implantée par un script PL/SQL indépendant du procédé et de l'exception, qui reçoit comme paramètre le nom du procédé et de l'exception à analyser, et produit une vue spécifique au procédé et à l'exception. La vue (représentée dans la figure 6.4) joint les tables d'analyse de procédé et InstancesEtiquetés, pour fournir un corpus de données qui contient les attributs des instances de procédé et les étiquettes. La vue obtenue inclut toute l'information demandée par l'outil de classification pour produire des règles de classification.

La phase de *data mining* peut être exécutée en utilisant différents algorithmes et techniques. Une variété d'applications de *data mining* et de classification de données sont disponibles sur le marché. Par conséquent nous n'avons pas développé nos propres algorithmes *data mining*, mais nous avons utilisé un outil commercial.

En particulier, nous utilisons les arbres de décision pour l'analyse des exceptions. Les arbres de décision [Qui93], [Qui86] sont largement répandus [Pro00] parce qu'ils fonctionnent bien avec des corpus de données très grands, avec un grand nombre de variables, et avec des données de type mixte (par instance, continu, discret, ou booléen). En outre, ils sont relativement faciles à comprendre (même par des utilisateurs qui ne sont pas des experts) et donc simplifient la phase ultérieure d'interprétation. Avec les arbres de décision, les objets sont classés en traversant l'arbre, à partir de la racine et en évaluant les conditions de branchement basées sur la valeur

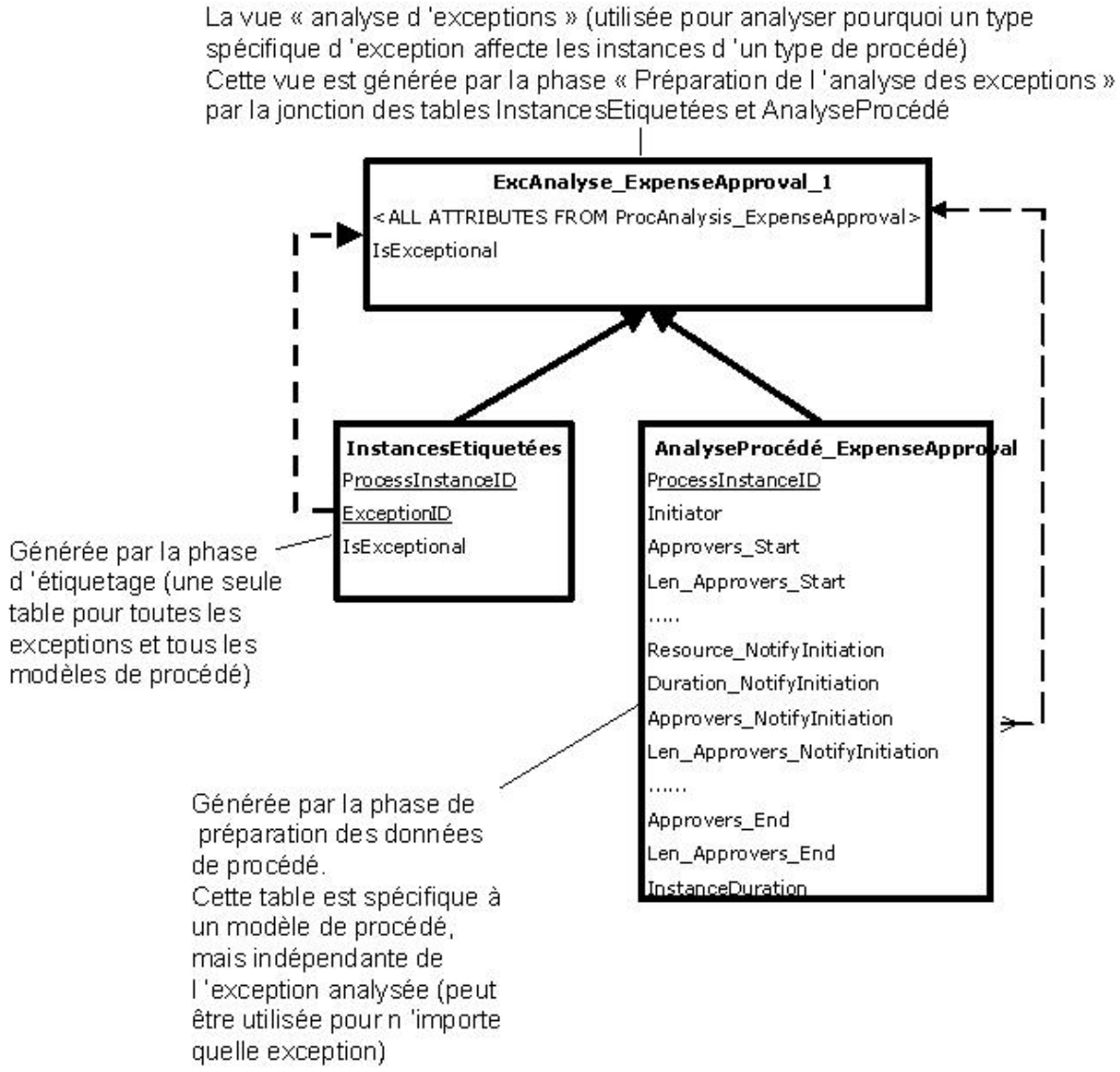


FIG. 6.4 – Tables et vues générées pour analyser l'exception de durée

des attributs des objets, jusqu'à ce qu'une feuille soit atteinte. Toutes les décisions représentent des partitions de l'espace attribut/valeur, de sorte qu'une et seulement une feuille soit atteinte.

Chaque feuille dans un arbre de décision identifie une classe. Par conséquent, un chemin d'accès de la racine à une feuille identifie un ensemble de conditions et une classe correspondante, c'est à dire, il identifie une règle de classification. Les feuilles contiennent également une indication sur la précision de la règle, c'est à dire la probabilité que les objets avec les caractéristiques identifiées appartiennent réellement à cette classe. Les algorithmes de génération des arbres de décision visent en particulier à identifier les feuilles de telle manière que les règles associées de classification soient aussi précises que possible. Une fois qu'un arbre de décision a été produit par l'outil de *data mining*, l'analyste peut alors se concentrer sur les feuilles qui classifient les instances comme exceptionnelles. Puis, l'analyste peut traverser l'arbre vers le haut,à partir de la feuille jusqu'à la racine, pour identifier quel attributs et valeurs d'attributs mènent à cette feuille et identifier donc les caractéristiques des instances « exceptionnelles ».

6.6 Prédiction et prévention des exceptions dans les instances courantes

Les concepts et les outils IP présentés dans la section précédente permettent l'analyse des exceptions. Les résultats sont censés être examinés par des utilisateurs humains pour identifier les causes d'une exception et pour éventuellement les enlever. Un autre but de notre travail est celui de prédire l'occurrence des exceptions. En particulier, nous visons à prédire des exceptions dès que possible dans l'exécution des instances des procédés, de sorte qu'elles puissent être évitées, ou de sorte qu'au moins les attentes adéquates puissent être formulées pour la vitesse et la qualité de l'exécution de procédé.

Suivant la structure de la section précédente, nous fournissons d'abord une vue d'ensemble de l'approche. Puis, nous détaillons l'approche et décrivons notre mise en oeuvre.

Le problème est que les règles de classification produites par l'analyse des exceptions ne donnent pas de bons résultats (et ne peuvent pas même être applicables) pour faire des prédictions pour les instances courantes. En fait, nous voulons classifier des instances de procédé en tant que « normales » ou « exceptionnelles » pendant qu'elles s'exécutent, et probablement à leur démarrage. Par conséquent, la valeur de certains attributs (comme la ressource ou la valeur des données de sortie pour un noeud qui n'a pas encore été exécuté) peut être toujours non définie. Si les règles de classification produites par la phase d'analyse des exceptions incluent de tels attributs, alors ces règles ne peuvent pas être appliquées, et l'instance de procédé ne peut pas être classifiée.

Nous abordons ce problème en modifiant la phase de préparation des données de procédé de sorte qu'elle produise plusieurs tables différentes d'analyse de procédé (qui vont produire probablement par la suite plusieurs ensembles différents de règles de classification), chacune destinée à faire des prédictions à une étape spécifique de l'exécution de l'instance de procédé. Une étape se caractérise par l'ensemble de noeuds exécutés au moins une fois dans l'instance. Par exemple, considérons la table d'analyse de procédé qui vise à dériver des règles de classification applicables au moment de l'instanciation du procédé. Elle est préparée en assumant seulement la connaissance des données suivantes : les données d'entrée de l'instance de procédé, la date de démarrage et le nom de la ressource qui a démarré l'instance. De cette façon, seuls ces attributs apparaîtront dans les règles de classification.

La phase d'étiquetage analyse les instances de procédé pour les étiqueter comme normales ou exceptionnelles, et elle est exécutée comme nous l'avons décrite dans la section précédente.

L'étiquetage doit être fait juste une fois : la même information étiquetante peut être utilisée pour des prédictions à n'importe quelle étape de l'exécution d'instance.

Les phases de préparation de l'analyse des exceptions, de *mining* et d'interprétation sont exécutées également comme nous les avons présentées dans la section précédente, avec la différence qu'elles sont exécutées une fois pour chaque table produite par la phase de préparation de données de procédé.

En plus de cinq phases communes avec l'analyse des exceptions, la prédiction des exceptions inclut également une phase de prédiction et une phase de réaction.

La **phase de prédiction** est la phase où des prédictions sur les instances de procédés courantes sont faites réellement. Durant cette phase, les règles de classification sont appliquées aux données courantes d'exécution de l'instance, pour classer les instances et pour obtenir, pour chaque instance courante et chaque exception d'intérêt, la probabilité que l'instance sera affectée par l'exception.

Dans la **phase de réaction**, les utilisateurs ou les systèmes sont alertés au sujet du risque de l'exception, et prennent les mesures appropriées pour réduire les « dommages » provoqués par l'exception, ou pour empêcher eventuellement son occurrence.

6.6.1 Détails et implantation

Cette section détaille les phases de préparation de données, de prédiction, et de réaction. Les autres phases ne sont pas discutées puisqu'elles sont exécutées et mises en application comme décrit dans la section précédente.

La phase de préparation de données est exécutée en déterminant d'abord les différentes étapes possibles de l'instance du procédé , c'est à dire les différentes combinaisons possibles des états d'exécution de noeud (exécutés ou non exécutés). Puis, pour chaque étape, la table de procédé d'analyse est construite comme décrit dans la section 6.5.3.

La première étape est toujours celle où on n'a exécuté aucun noeud, et elle est utilisée pour faire des prédictions au moment de l'instanciation du procédé. Pour cette étape, la table de procédé d'analyse contient seulement des informations sur le moment d'instanciation, la valeur initiale des données élémentaires de procédé, et le nom de la ressource qui a démarré l'instance. Les tables d'analyse de procédé produites pour les autres étapes incluent, pour chaque noeud exécuté, les mêmes attributs de noeud énumérés dans la section d'analyse des exceptions. Dans la mise en oeuvre actuelle, pour la simplicité, nous considérons seulement la première exécution du noeud, de sorte qu'au plus un ensemble d'attributs pour chaque noeud est inclus.

Cette phase est implantée par un script PL/SQL qui prend le nom du procédé comme paramètre d'entrée et produit toutes les tables de procédé d'analyse pour ce procédé.

La phase de prédiction est exécutée par le moniteur d'exceptions (ME). Il fait partie du composant GSO de la figure 6.2, et accède le *data warehouse* et l'historique du WfMS pour faire des prédictions.

L'accès à l'historique est nécessaire puisque le *data warehouse* du composant IP ne contient pas des données des instances courantes, mais il est mis à jour périodiquement (typiquement une fois par jour ou une fois par mois), selon les besoins. Par conséquent, alors que les règles de classification peuvent être obtenues « off-line », en analysant des données d'entrepôt, les prédictions réelles doivent être faites sur les données courantes que le WfMS écrit dans son historique.

L'accès à l'entrepôt d'IP est nécessaire pour rechercher les règles de classification, produites à l'avance. En effet, notre approche suppose que la phase de *data mining* enregistre sa sortie dans la base de données, de sorte que des règles puissent non seulement être interprétées par des humains, mais également utilisées par des applications telles que le moniteur d'exceptions.

Le moniteur d'exceptions fonctionne en accédant périodiquement à l'historique d'audit de WfMS et en copiant les tables contenant des informations sur les exécutions des instances de procédés. Cette opération est tout à fait simple et elle est exécutée sur une base de données relativement petite (puisque des données sont périodiquement purgées du l'historique d'audit et archivées dans l'entrepôt). Par conséquent, elle a un effet négligeable sur l'exécution du système opérationnel. Une fois que les données ont été copiées, le moniteur d'exceptions examine les instances de procédés à surveiller.

En particulier, pour chaque instance, le moniteur d'exceptions détermine d'abord l'étape de l'exécution de l'instance de procédé, en vérifiant quels noeuds ont été exécutés. Ensuite, il accède à l'entrepôt d'IP pour rechercher les règles de classification (qui dans notre cas ont la forme d'un arbre de décision) à appliquer, correspondant à l'étape de l'exécution de l'instance de procédé. Une fois que l'arbre approprié de décision a été identifié, le moniteur d'exceptions parcourt l'arbre et évalue chaque condition de branchement basée sur la valeur des attributs de l'instance de procédé, jusqu'à ce qu'elle atteigne une feuille. La feuille contiendra une indication de la probabilité que l'instance examinée soit exceptionnelle. Si cette probabilité est au-dessus d'un seuil, alors un nouveau enregistrement est inséré dans une table d'avertissement, détaillant l'identificateur de l'instance de procédé, l'identificateur de l'exception, l'étape de procédé d'exécution, et la probabilité de l'occurrence de l'exception.

La phase de réaction est exécutée par le gestionnaire de prévention des exceptions, qui fait partie également du GSO. Le gestionnaire de prévention des exceptions surveille la table d'avertissement. Quand une nouvelle exception est prédite pour un instance de procédé, le gestionnaire de prévention des exceptions alerte l'utilisateur enregistré en tant que responsable pour le procédé. Les utilisateurs peuvent alors exécuter des actions dans le WfMS ou dans l'organisation pour essayer d'éviter l'exception ou de réduire son impact. En plus, le gestionnaire de prévention des exceptions peut être configuré afin qu'il interagisse de manière autonome et pro-active avec le WfMS pour essayer d'éviter l'exception. Actuellement, la seule forme permise d'intervention automatique consiste à augmenter la priorité de l'instance de procédé pour les instances qui sont susceptibles d'être en retard. L'administrateur de procédé peut indiquer le niveau auquel la priorité peut être augmentée selon la probabilité que l'instance de procédé soit en retard.

Le moniteur d'exceptions et le gestionnaire de prévention d'exceptions incluent des modules en Java et en SQL.

6.7 Résultats expérimentaux

6.7.1 Analyse des exceptions

Nous avons appliqué l'approche et l'outil IP pour analyser plusieurs procédés administratifs au sein de HP, tels que des remboursements des employés et des demandes d'achat. Ces procédés sont implantés sur le système de gestion de procédé HP Process Manager, et sont consultés par des centaines d'employés chaque jour. Comme exemple représentatif, nous discutons les résultats obtenus en analysant le procédé d'approbation de dépenses décrit dans la figure 6.5, et spécifiquement en identifiant les caractéristiques des instances qui prennent plus de 8 jours pour se terminer (le temps d'exécution moyen était d'environ 5 jours). Nous avons eu accès aux données de procédé d'exécution pour une période de cinq mois, correspondant à approximativement 50000 instances de procédé. Environ 15% des instances ont été affectées par cette « exception ».

Après l'importation des données de procédé dans l'entrepôt d'IP, et après avoir défini l'exception comme représentée dans la figure 6.3, nous avons exécuté les scripts décrits dans la section précédente pour étiqueter les instances et pour produire la table d'analyse d'exception, qui est

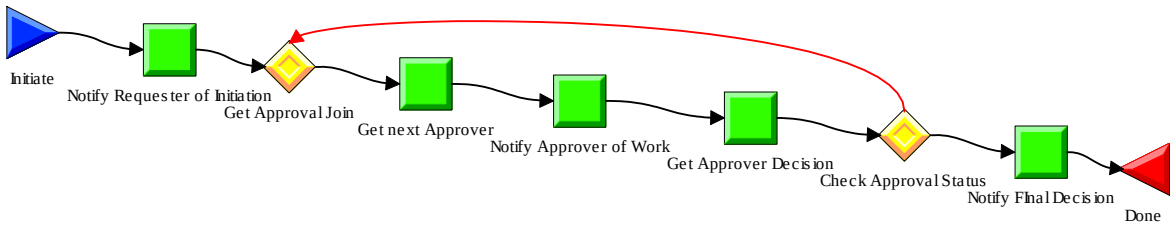


FIG. 6.5 – Procédé d'approbation de dépenses

large (plus de 100 colonnes). Nous avons ensuite utilisé l'outil de *data mining* SAS Enterprise Miner (un outil commercial de référence pour le domaine de *data mining*) pour la génération des arbres de décision. Pour la génération de l'arbre de décision nous avons utilisé Ξ^2 comme critère de branchement de noeud, et nous avons utilisé la proportion d'enregistrements correctement classés comme valeur d'évaluation. 60% des enregistrements ont été utilisés comme corpus d'apprentissage tandis que 40% des enregistrements ont été utilisés pour la validation. Ce sont les paramètres qui nous ont donné les meilleurs résultats globaux. Une analyse complète et formelle pour déterminer les paramètres qui donnent les meilleurs résultats pour les différents types de procédés et d'exceptions fait partie de nos futures directions de recherche.

Après plusieurs essais infructueux, qui nous ont menés à restructurer les schémas des bases de données et les scripts de préparation comme décrit plus haut dans cette section, l'outil a finalement produit des résultats intéressants. Pour des raisons de simplicité et de confidentialité, nous ne montrons pas l'arbre exact de décision et tous les détails des résultats. Cependant, nous récapitulons les résultats principaux dans la figure 6.6. Les résultats montrent ce qui suit :

- les cas qui ont exigé beaucoup de contrôleurs de dépenses avaient une probabilité plus grande de durer plus longtemps. En particulier, les instances exceptionnelles ont typiquement eu plus de 6 contrôleurs.
- quand, en plus d'avoir plus de 6 contrôleurs, des employés dans un groupe spécifique ont exécuté des activités de secrétariat, alors 70% des instances étaient « exceptionnelles ». Par contre, la majorité d'instances de procédé étaient à l'heure quand d'autres employés ont exécuté de telles activités de secrétariat.
- les instances de procédé commencées le vendredi avaient une probabilité plus grande de durer plus longtemps, puisque, en fait, le travail était reporté au lundi suivant.

Comme c'est souvent le cas dans notre analyse, certaines des corrélations identifiées ne sont pas immédiatement utiles pour comprendre et résoudre le problème. Par exemple, le fait que les procédé avec plusieurs contrôleurs (et donc avec plusieurs exécutions de noeuds) durent plus longtemps, était prévisible. Cependant, une utilisation possible de cette information est d'avoir une bonne estimation du temps d'exécution de procédé. Si toutes les corrélations identifiées sont de ce type, alors il est très très probable que ce qu'on a classé comme « exception » n'est pas vraiment une exception, mais simplement un aspect qui fait partie de la nature du procédé.

Cependant, les corrélations identifiées sont souvent utiles pour identifier les aspects du procédé ou de l'organisation qui peuvent être améliorés. Par exemple, suite à la découverte de la corrélation entre les exceptions et le groupe de ressources, une deuxième inspection des données du *data warehouse* a indiqué que les employés dans ce groupe ont eu plus de charge de travail par rapport aux autres. Ainsi, l'analyse a permis d'identifier le problème principal et de suggérer une solution,

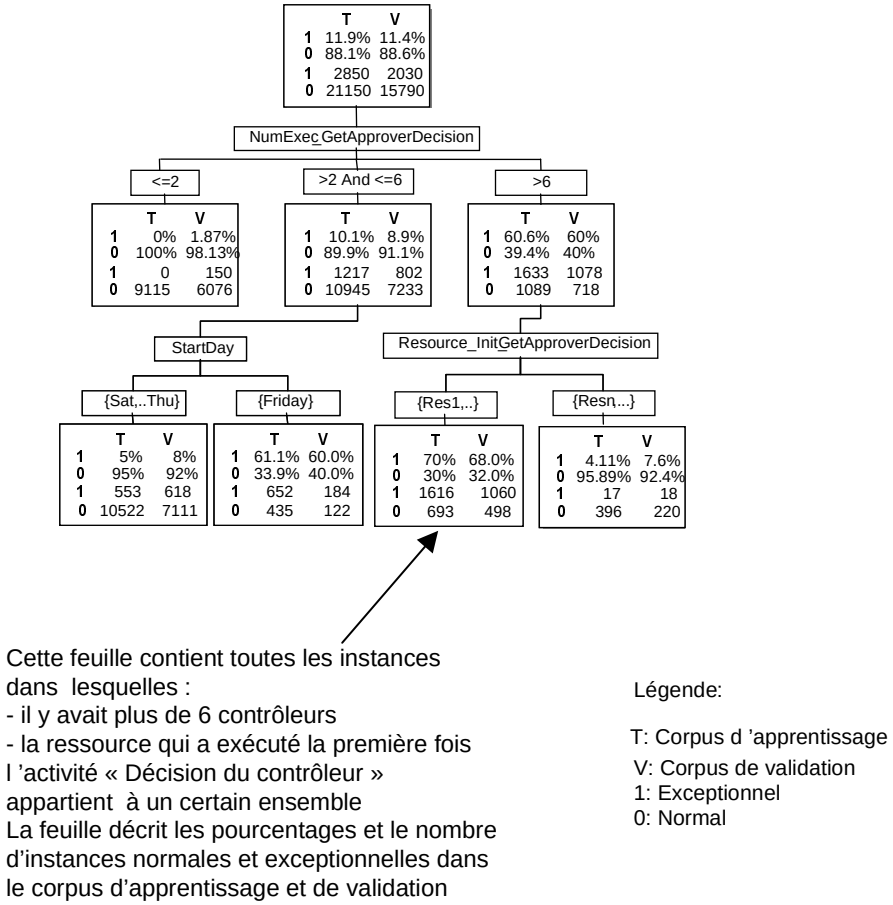


FIG. 6.6 – Arbre de décision simplifié obtenu en analysant le procédé d'approbation de dépenses

par exemple, une nouvelle répartition du travail.

Les résultats de l'analyse du composant IP, comme c'est souvent le cas pour les méthodes de *data mining*, doivent être interprétées avec précaution et les polarisations des données disponibles doivent être identifiées. Un problème que nous avons rencontré souvent et qui est caractéristique aux applications de *data mining* est l'effet de bord : typiquement, l'analyse est exécutée sur des procédés commencés (et/ou terminés) dans une certaine fenêtre de temps. Par exemple, un corpus des données peut contenir toutes les instances terminées en octobre. Si un outil de *data mining* analyse ces données, il déterminera que les instances commencées au printemps durent plus longtemps que celles commencées pendant l'été ou l'automne. En effet, l'outil déclarera que l'exactitude de cette règle est très bonne. Cependant, le résultat est seulement dû à la façon dont les données sont rassemblées, plutôt qu'à une propriété du procédé : en fait, le corpus est polarisé en ce qui concerne la date de début, parce qu'il contient les instances commencées au printemps seulement si elles ont duré très longtemps, c'est à dire jusqu'en octobre.

6.7.2 Prédiction d'exceptions

Nous montrons maintenant des résultats préliminaires obtenus en appliquant l'approche pour la prédiction des exceptions présentée dans la section 6.6 aux procédés administratifs de HP. Nous nous référerons de nouveau au procédé d'approbation de dépenses de la figure 6.5, et spécifiquement à l'exception liée à la durée du procédé du schéma 6.3.

Dans le procédé d'approbation de dépenses, il était possible d'avoir une bonne prédiction pour la durée de l'instance même au démarrage des instances. En fait, l'arbre de décision résultant a indiqué que la longueur du procédé est corrélée avec le nom du demandeur (le créateur de l'instance) et avec la longueur de la donnée élémentaire « Contrôleur », qui contient les noms des contrôleurs (et donc sa longueur indique le nombre de boucles à exécuter). Pour quelques combinaisons de ces valeurs, plus de 70% des instances ont été affectées par l'exception.

Comme on pourrait s'y attendre, les prédictions deviennent de plus en plus précises au fur et à mesure que l'exécution de l'instance de procédé avance, puisque plus d'information devient disponible. En particulier, des prédictions très précises peuvent être faites juste après l'exécution du noeud « Décision du contrôleur ». Cette activité est la plus longue, et donc après son exécution il est possible d'avoir plus d'informations sur la probabilité que le procédé dépasse le temps d'exécution acceptable. L'arbre de décision pour la prédiction construit pour cette étape du procédé (représenté dans la figure 6.7) prouve en fait que, si la première exécution du noeud « Décision du contrôleur » prend plus de cinq jours et demi, alors l'instance peut être prévue comme exceptionnelle avec une probabilité de 93%.

Nous observons également que la durée de ce noeud est également apparue dans les règles produites pour l'analyse de cette exception. Cependant, dans ce contexte, nous avons supprimé cet attribut du corpus d'entrée, puisqu'il n'était pas particulièrement utile pour identifier des corrélations « intéressantes » et pour comprendre pourquoi l'exception se produit. Par contre, dans ce cas-ci nous nous focalisons sur les prédictions et chaque corrélation devient utile.

En général, nous avons observé qu'il existe seulement quelques étapes dans une instance de procédé où l'exactitude de la prédiction s'améliore, typiquement après l'exécution de quelques noeuds « critiques ». Une optimisation intéressante de nos algorithmes, une partie de nos travaux futurs, est donc basée sur l'identification de telles étapes, de sorte qu'on puisse exécuter les diverses phases de prédiction d'exceptions pour ces étapes seulement.

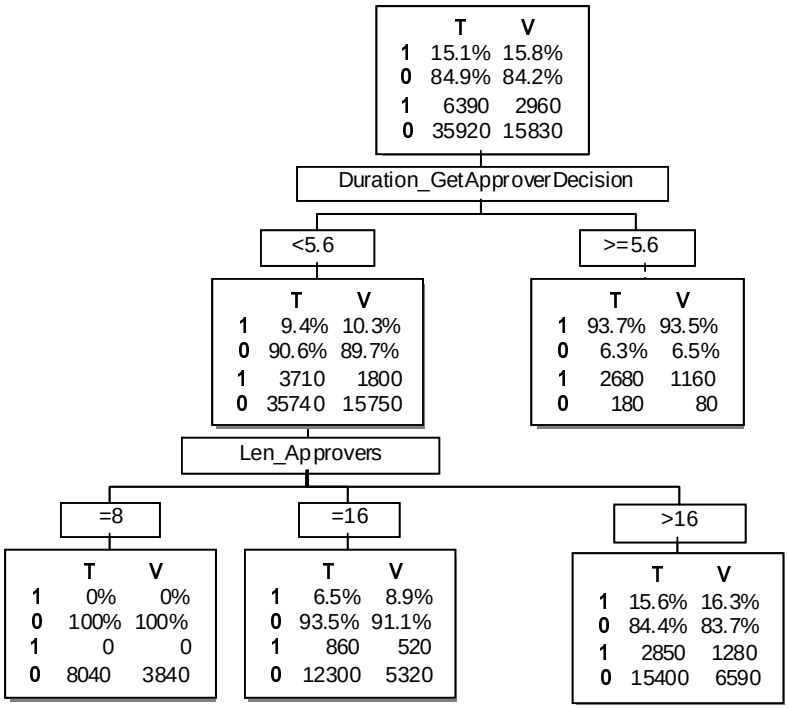


FIG. 6.7 – Arbre de décision simplifié pour la prédiction de l'exception après l'exécution de l'activité "Décision du Contrôleur"

6.8 Conclusion

Ce chapitre a présenté une approche et une suite d'outils pour l'analyse, la prédiction et la prévention des exceptions. Nous avons discuté les difficultés principales que nous avons rencontrées, et nous avons décrit comment nous les avons abordées dans notre approche et notre implantation. Les résultats expérimentaux ont été prometteurs. En conséquence, nous envisageons de continuer cette démarche et d'aborder les problèmes qui restent ouverts. En particulier, nos perspectives de recherche incluent l'amélioration des algorithmes de prédiction des exceptions, pour améliorer le traitement du problème des exécutions multiples d'un noeud dans la même instance de procédé. Par la suite, en accumulant plus de connaissances sur le problème, nous visons à développer de nouveaux algorithmes de classification, capables de balayer des données significatives pour une exception, dispersées dans plusieurs enregistrements et dans plusieurs tables et d'extraire des règles de classification, sans avoir besoin d'agréger des données dans une seule table.

D'autres objectifs de recherche incluent : une phase raffinée de préparation de données de procédé qui choisit des attributs également basés sur l'exception étant analysée, l'élaboration d'une méthodologie complète pour l'analyse des exceptions et des mécanismes améliorés et automatiques pour éviter les exceptions.

D'autres objectifs de recherche dans le contexte de l'intelligence de procédé incluent la découverte de la définition du procédé, analyse et prédiction de chemin d'exécution, et l'optimisation dynamique du système, du procédé, et de l'assignation des ressources.

Chapitre 7

Bilan et perspectives

Comme nous l'avons indiqué en introduction, les travaux réalisés dans cette thèse visent à supporter les procédés coopératifs d'une équipe virtuelle, pour des applications créatives de type co-ingénierie ou co-conception.

Les systèmes qui proposent actuellement des solutions pour la coordination des équipes distribuées peuvent être classés en deux grandes catégories. La première est l'ensemble des outils de type collecticiels qui mettent en avant les aspects de communication et de conscience de groupe. Ils permettent la coopération à travers des données partagées pour des procédés non-structurés, mais ignorent les aspects organisationnels et n'offrent pas de mécanismes pour mesurer l'avancement d'un projet. La plupart des systèmes n'incluent pas de support pour la modélisation et la gestion des dépendances entre activités, leur objectif principal étant de capter des pratiques réelles de travail. La seconde catégorie est celle des outils de workflow qui permettent une modélisation précise mais rigide des procédés et nécessite un travail important d'intégration avec l'environnement existant. Cette approche, valide pour des procédés de production bien maîtrisés, est inadaptée pour des procédés de conception plus complexes et moins répétables. Notre approche a consisté à utiliser cette catégorie d'outil et de modèle pour l'adapter au contexte particulier des procédés coopératifs.

Nous avons proposé pour cela un modèle de workflow coopératif qui permet une modélisation simple des procédés, une exécution flexible et un échange de données flexibles entre activités. Nous détaillons maintenant ces contributions.

7.1 Rappel des contributions

7.1.1 Modèle de workflow coopératif

Les procédés créatifs de co-conception et co-ingénierie peuvent être décrits avec le formalisme graphique utilisé pour les procédés administratifs ou de production, « en dessinant des boîtes et des arcs ». Ce type de modèle est facilement accepté par les utilisateurs. De plus, il permet la visualisation graphique du workflow et offre un bon support pour l'analyse. La **simplicité du modèle** facilite l'intervention des participants pour faire des modifications dynamiques.

Le système de workflow coopératif permet une **exécution flexible** de ce modèle. Les activités peuvent « anticiper » par rapport au flot de contrôle, c'est à dire elles peuvent commencer à s'exécuter même si leurs conditions de démarrage ne sont pas encore satisfaites. La condition de démarrage de l'activité qui anticipe est vérifiée quand l'activité précédente termine. Ainsi, le contrôle global est assuré.

Ce type d'exécution permet un retour d'expérience rapide et accélère l'exécution du procédé. Il reflète une pratique de travail fréquente dans les applications coopératives qui est difficile à supporter en utilisant les systèmes actuels.

L'anticipation augmente le **parallélisme** entre activités et entre un procédé et son sous-procédé. Dans notre modèle, le sous-procédé n'est pas une boîte noire pour le procédé principal, comme c'est le cas dans le modèle traditionnel. La publication des données produites par le sous-procédé pendant son exécution permet l'anticipation des activités du procédé principal. Ainsi le parallélisme entre les activités des deux procédés est augmenté.

Différentes stratégies d'anticipation peuvent être définies afin de permettre différents degrés de flexibilité dans l'exécution d'une instance ou d'un type de procédé.

L'échange de données flexible est assuré en encapsulant les activités dans des transactions coopératives. Ceci permet aux activités de travailler de manière coopérative sur les données communes : s'échanger des versions intermédiaires, travailler sur les mêmes données. Ce modèle de flot de données permet de dépasser les limites du modèle « activités - boîtes noires » considéré par les workflows traditionnels, tout en maintenant des garanties de cohérence offertes par le protocole de transactions coopératives. La gestion de données partagées est assurée en intégrant un gestionnaire de transactions coopératives dans le système de workflow.

Le système de workflow proposé permet de supporter des procédés coopératifs aussi bien que compétitifs. Les procédés non-structurés, dans lesquels les activités n'ont pas un ordre défini d'avance, peuvent être définis. Les activités s'échangent des données et les dépendances entre elles sont définies dynamiquement et gérées par le gestionnaire de transactions. Les procédés compétitifs sont considérés comme des restrictions du modèle coopératif. Un procédé compétitif (ou traditionnel) est simplement un procédé coopératif où l'anticipation est interdite dans le composant de gestion d'activités et la visibilité des résultats intermédiaires est interdite dans le composant de gestion de transactions.

Les stratégies d'anticipation et les mécanismes d'échange de données permettent l'exécution des procédés coopératifs avec des différents degrés de contrôle, en fonction des besoins de l'application et des pratiques de travail.

Les **modifications dynamiques** du modèle sont permises et assurent la liberté des utilisateurs d'adapter le procédé à la situation courante de travail. Les activités peuvent être ajoutées, supprimées et les dépendances entre elles peuvent être modifiées.

Les éléments de flexibilité proposés ont fait l'objet de deux **mises en œuvre**. La première mise en œuvre a repris les résultats décrits dans cette thèse sur l'échange flexible de données dans le cadre d'un projet industriel, le projet Corvette. Ce projet a montré l'intérêt et les limites de l'intégration d'un système transactionnel coopératif avec un système de workflow traditionnel pour supporter des procédés coopératifs. Le prototype résultant fournit un support pour l'échange flexible et contrôlé des données, mais impose un contrôle rigide concernant l'ordre d'exécution des activités. La deuxième mise en œuvre réalise l'intégration d'un moteur de workflow flexible dans le prototype MOTU, la plate-forme pour le travail coopératif développée au sein de l'équipe ECOO. Elle a montré la place et l'intérêt des mécanismes de coordination parmi les autres services de coopération.

7.1.2 Méthodes et outils d'analyse, prédiction et prévention des exceptions

L'intelligence de procédé a comme objectif l'intégration des outils intelligents dans le système de workflow pour faciliter l'analyse et l'optimisation des procédés.

Nous avons identifié les principaux problèmes de recherche et nous avons proposé une technique et un outil pour l'analyse et la prédiction des exceptions. La méthode d'analyse des excep-

tions se base sur des techniques de *data mining* appliquées à l'historique du système de workflow. Ces méthodes visent à identifier les facteurs qui ont des corrélations fortes avec l'apparition d'une exception.

L'utilisateur définit le type d'exception à analyser (par exemple, l'exception liée au dépassement d'une date limite ou au nombre d'exécutions d'une activité). L'étape de *data mining* de notre méthode d'analyse est appliquée aux données des exécutions de procédés et génère des règles pour classer une instance de procédé comme normale ou exceptionnelle. Ces règles sont des associations entre des prédicats sur les attributs de l'instance de procédé et la classe de procédé (normale ou exceptionnelle). En analysant ces règles, l'analyste peut comprendre les causes des exceptions. Cette analyse peut mettre en évidence les aspects du procédé qui ne sont pas efficaces et des solutions pour l'amélioration du procédé.

La méthode de prédiction des exceptions se base sur le même principe. L'étape de *data mining* génère les règles qui classifient une instance comme normale ou exceptionnelle. En appliquant ces règles sur les données de l'instance en cours d'exécution, la probabilité d'apparition d'une situation exceptionnelle peut être estimée. De cette manière, l'administrateur de procédé peut essayer d'éviter l'apparition de l'exception. Le système peut être configuré pour prendre des mesures automatiques de prévention quand la probabilité dépasse un certain seuil (par exemple augmenter la priorité de l'instance).

Ce travail a été réalisé en coopération avec Fabio Casati, Umesh Dayal et Ming Chen Shan de Centre de recherche Hewlett Packard (Palo Alto).

7.2 Perspectives

7.2.1 Modèle de workflow coopératif

A court terme, nous entrevoyons les évolutions suivantes de notre modèle de workflow coopératif.

Recouvrement Un aspect que nous n'avons pas étudié et qui est important pour un procédé coopératif est le recouvrement d'un état cohérent en cas d'échec d'une activité. La solution offerte typiquement par les modèles de workflow est de revenir en arrière jusqu'à une certaine activité et reprendre l'exécution après avoir compensé les activités sur le chemin exécuté. Pour les activités coopératives, il est difficile de définir à l'avance les activités correspondantes de compensation. En plus, la complexité du problème de recouvrement est augmentée par l'échange spontané de données entre activités. Nous pensons que les utilisateurs ont besoin de définir dynamiquement un fragment de procédé pour la compensation.

Intégration d'autres gestionnaires de transactions Pour assurer un échange de données flexible, mais en même temps contrôlé entre activités, nous avons choisi d'implanter les activités du procédé par des COO transactions. Ceci oblige les utilisateurs à se conformer au protocole COO. Pour répondre aux besoins de différents types d'applications, le système de workflow devrait permettre aux utilisateurs de choisir différentes politiques de partage des données. En conséquent, le système de workflow devrait permettre l'intégration des différents gestionnaire de données. L'intégration du système de workflow avec le gestionnaire de transactions coopératives COO nous a permis de mettre en évidence les modifications qui doivent être apportées au système de workflow. Ces modifications sont en partie spécifiques au modèle de transactions choisi. Une évolution de ce travail consiste à définir les fonctionnalités qui doivent être offertes par le système

de workflow afin de permettre l'intégration avec différents gestionnaires de données partagées. Ceci permettra aux utilisateurs de choisir entre différentes politiques de gestion de données partagées, en fonction des besoins de l'application. En plus, des primitives simples du flot de données doivent être définies pour cacher aux utilisateurs toute la complexité des différents modèles d'accès aux données.

Analyse d'usage L'analyse des besoins pour la flexibilité des systèmes de workflow qui a motivé notre travail, se base sur notre propre expérience dans le développement de logiciel et intègre les conclusions des discussions avec nos collaborateurs, des architectes dans le domaine de constructions de bâtiment. Cependant, nous n'avons pas encore fait des évaluations formelles de notre prototype dans un environnement réel de travail. La prochaine étape de notre travail est de valider le modèle proposé à travers d'expérimentations avec retour chez les usagers de départ. Le retour d'expérience obtenu nous permettra d'améliorer notre modèle et d'étudier le rôle d'un support formel pour la coordination pour les procédés coopératifs.

7.2.2 Workflow émergent

Nos perspectives de recherche à long terme visent à développer une solution de « workflow émergent » pour la *gestion des cas*[20]. La gestion des cas est un nouveau paradigme pour supporter les procédés flexibles, qui incluent une forte composante créative, intellectuelle. Ces procédés sont fortement basés sur des données, leur résultat étant concrétisé dans la production des données. A la différence des systèmes de workflow, qui utilisent des structures de contrôle prédéfinies pour déterminer ce qui devrait être fait pendant un procédé, la gestion de cas se concentre sur ce qui peut être fait pour réaliser un certain objectif. Dans la gestion des cas, les participants responsables d'un cas particulier possèdent les compétences nécessaires pour décider activement et dynamiquement de la façon dont l'objectif de ce cas peut être atteint. Le rôle d'un système de gestion de cas est d'aider plutôt les utilisateurs que de les guider dans leur travail. Dans ce cadre, divers aspects des systèmes de gestion de cas doivent être étudiés : la conception des langages appropriés, la modélisation du comportement dynamique des systèmes de gestion de cas, les problèmes liés à la conception de ces systèmes et l'intégration des outils intelligents pour exploiter l'expérience et les connaissances sur les exécutions passées des procédés.

7.2.3 Intelligence de procédé

Les perspectives de recherche à court terme ont comme objectif les améliorations directes de notre méthode qui ont été présentées dans le chapitre 6.

L'intelligence de procédé est un domaine nouveau de recherche qui ouvre beaucoup de perspectives [Day01] de recherche à long terme. Nous sommes spécialement intéressés par l'application de ces méthodes aux procédés coopératifs.

Analyse et prédiction des chemins d'exécution Nous avons vu que pour l'anticipation d'une activité située sur une branche conditionnelle, il serait utile de connaître la probabilité que l'activité soit exécutée dans l'instance en cours (la probabilité que la condition associée à la branche soit évaluée à vrai). Des requêtes simples dans la base de données de l'historique de procédé nous donnent le pourcentage des exécutions de l'activité par rapport au nombre d'exécutions de l'activité précédente (activité de branchement). En appliquant les techniques présentées dans la partie 6 il est certainement possible d'obtenir de meilleurs estimations de la

[20] en anglais, case handling

probabilité d'exécution de l'activité (par exemple en découvrant des corrélations fortes avec les données initiales du procédé, l'exécution des branches précédentes, etc.)

Découverte de patrons de travail coopératif Notre modèle de workflow coopératif permet une exécution flexible et un échange de données flexibles. Nous pensons qu'il facilite l'acceptation initiale du système de workflow. Une fois que le système est en place et un nombre important d'instances a été exécuté, l'analyse de l'historique permet de comprendre la manière dont les gens travaillent, pourquoi et quand ils anticipent, comment ils échangent des données. Cette analyse peut aussi révéler des structures de procédé qui apparaissent régulièrement. Ces structures pourraient être modélisées en utilisant de nouveaux opérateurs (par exemple les opérateurs COO [God99]), ou de nouvelles dépendances qui représentent mieux les pratiques de travail représentées par les structures données.

D'autres problèmes plus généraux sont le développement d'une méthodologie pour l'analyse et l'amélioration des procédés, l'étude des types d'analyse utiles pour l'optimisation du procédé et des méthodes d'optimisation basées sur les résultats de l'analyse.

Bibliographie

[Ago96] A. Agostini et G. De Michelis. Modeling the document flow within a cooperative process as a resource for action. Rapport no. CTL-DSI, University of Milano, 1996.

[Ago98] A. Agostini et G. De Michelis. Simple models for articulating complex work processes. *Towards Adaptive Workflow Systems,Workshop within the Conference on Computer Supported Cooperative Work*, 1998.

[Ago00a] A. Agostini et G. De Michelis. *Business Process Management*, chapitre Improving Flexibility of Workflow Management Systems, pages 218–234. Number 1806. LNCS, 2000.

[Ago00b] A. Agostini et G. De Michelis. A light workflow management system using simple process models. *Computer Supported Cooperative Work*, number 3/4, pages 335–363, 2000.

[Aks88] R. Aksyn, D. McCracken et E. Yoderer. Kms : A distributed hypermedia systems for managing knowledge in organizations. *Communications of the ACM*, 31(7), 1988.

[Alo94] G. Alonso, M. Kamath, D. Agrawal, A. El Abbadi, R. Gunthor et C Mohan. Failure handling in large scale workflow management systems. Rapport, IBM Almaden research Center, 1994.

[Alo96] G. Alonso, D. Agrawal, A. El Abbadi, M. Kamath, R. Gunthor et C. Mohan. Advanced transaction model in workflow context. *12th International Conference on Data Engineering*, 1996.

[Ant95] P. Antunes, J. Segovia, N. Guimaraes et J. Cardenosa. Beyond formal processes : Augmenting workflow with group interaction techniques. *Conference on Organizational Computer Systems (COOCS '95)*, 1995.

[Att93] P.C. Attie, M. Rusinkiewicz, A. Sheth et M.P. Singh. Specifying and enforcing intertask dependencies. *19th International Conference on Very Large Data Bases (VLDB)*, 1993.

[Ban93a] S. Bandinelli, L. Baresi, A. Fuggetta et L. Lavazza. Requirements and early experiences in the implemenation of the spade repository. *8th international Sotware process Workshop*, 1993.

[Ban93b] S. Bandinelli, A. Fuggetta et C. Ghezzi. Software process model evolution in the spade environment. *IEEE Transactions on Software Engineering*, 19(12), 1993.

[Bel94] N. Belkhatir, J. Estublier et W.L. Melo. *Software process modelling and technology*, chapitre ADELE-TEMPO : An Environment to Support process Modelling and Enaction, pages 187–222. Research Studies press, 1994.

[bil94] A. biliris, S. Dar, N. Gehani, H.V. jagadish et Ramamritham. Asset : A system for supporting extended transactions. *ACM SIGMOD Int. Conference on Management of Data*, 1994.

[Bon01] A. Bonifati, U. Casati, F.and Dayal et M.C. Shan. Warehousing workflow data : challenges and opportunities. *25th International Conference on Very Large Data Bases*, 2001.

[Can96] G. Canals, P. Molli et C. Godart. Concurrency control for cooperating software processes. *Proceedings of the 1996 Workshop on Advanced Transaction Models and Architecture (ATMA'96)*, Goa, India, 1996.

[Can98] Gérôme Canals, Claude Godart, Frangois Charoy, Pascal Molli et Hala Skaf. COO approach to support cooperation in software developments. *IEE Proceedings Software Engineering*, 145(2-3) :79–84, 1998.

[Cas96] F. Casati, S. Ceri, B. Pernici et G. Pozzi. Workflow evolution. *15th Int. Conf. On Conceptual Modeling (ER'96)*, 1996.

[Cas98] F. Casati, S. Ceri, B. Pernici et G. Pozzi. Workflow evolution. *Data and Knowledge Engineering*, 1998.

[Cel94] W. Cellary et G. Jomier. Apparent versioning and concurrency control in object-oriented databases. *Proceedings of the 6th International Conference on Computing and Information*, 1994.

[Cel96] W. Cellary et J. Rykowski. Multiversion databases - support for software engineering. *2nd World Conference on Integrated Design and Process Technology*, 1996.

[Chr90] P.K. Chrysanthis et K. Ramamritham. Acta : A framework for specifying and reasoning about transaction structure and behavior. *ACM SIGMOD Int. Conference on Management of Data*, 1990.

[Coa95] Workflow Management Coalition. Workflow reference model, 1995.

[Coa96] Workflow Management Coalition. Terminology and glossary, 1996.

[Coa99] Workflow Management Coalition. Interface 1 - process definition interchange, 1999.

[Con91] R. Conradi, E. Osjord, P.H. Westby et C. Liu. Initial software process management in epos. *Software Engineering Journal*, 1991.

[Day91] U. Dayal, M. Hsu et R. Ladin. A transaction model for long-running activities. *17th International Conference on Very Large Data Bases (VLDB)*, 1991.

[Day01] U. Dayal, M. Hsu et R. Ladin. Business process coordination : State of the art, trends, and open issues. *25th International Conference on Very Large Data Bases(VLDB)*, 2001.

[Ede95] J. Eder et W. Leibhart. The workflow activity model wamo. *3rd International Conference on Cooperative Information Sytems (CoopIs)*, 1995.

[Ede98] J. Eder et W. Leibhart. Contributions to exception handling in workflow management. *Sixth International Conference on Extending Database Technology*, 1998.

[Ede99] J. Eder, E. Panagos, H. Pozewaunig et M. Rabinovich. Time management in workflow systems. *BIS'99*, 1999.

[Ell91] C. Ellis, S. Gibbs et G. Rein. groupware : Some issues and experiences. *Communications of the ACM*, 34(1), 1991.

[Ell96] C. Ellis et G. Nutt. Workflow : The process spectrum. *NSF Workshop on Workflow and Process Automation in Information System*, 1996.

[Ell97] C. Ellis et C. Maltzahn. Chautaqua workflow system. *30th Hawaii Int Conf. On System Sciences, Information System Track,*, 1997.

[Elm90] A. Elmagarmid, Y. Leu, W. Litwin et M. Rusinkiewicz. A multidatabase transation model for interbase. *16th International Conference on Very Large Data Bases (VLDB)*, 1990.

[Elm92] Elmagarmid, éditeur. *Database Transaction Models for Advanced Applications.* Morgan-Kaufmann, 1992.

[Eng94] G. Engels et L. groenewegen. Socca : Speicfication of coordinated and cooperative activities. *Software process modelling and technology.* 1994.

[Est97] J. Estublier, S. Dami et M. Amiour. High level process modeling for scm systems. *Software Configuration Management Workshop (SCM-7) at ICSE'97*, 1997.

[Faa97] F.J. Faase, S.J. Even, , R.A. de By et P. Apers. Integrating organisational and transactional aspects of cooperative activities. *Workshop on Database Programming Languages*, pages 336–354, 1997.

[Fei91] P.H. Feiler. Configuration management models in commercial environments. Rapport, Software Engineering Institute, Carnegie Mellon University, 1991.

[Fin94] A. Finkelstein, J. Kramer et B. Nuseibeh. *Software process modelling and technology.* John Wiley and sons, Inc, 1994.

[Geo94] D. Georgakopolous, M. Hornick, P. Krychniak et F. Manola. Specificaation and management of extended transactions in a programmable transaction environment. *10th IEEE Int. Conference on data Engineering (ICDE)*, 1994.

[Geo99] D. Georgakopoulos, H. Schuster, A. Cichocki et D. Baker. Managing process and service fusion in virtual enterprises. *Information Systems, Special Issue on Information Systems Support for Electronic Commerce*, 24(6) :429–456, 1999.

[Geo00] D. Georgakopoulos, H. Schuster, D. Baker et A. Cichocki. Managing escalation of collaborative processes in crisis mitigation situations. *16th Int. Conference on data Engineering (ICDE'2000)*, 2000.

[GM87] H Garcia-Mollina et Salem K. Sagas. *ACM SIGMOD International Conference on Management of Data*, 1987.

[GM91] H. Garcia-Mollina, D. Gawlick, J. Klein, K. Kleissner et K. Salem. modelling long-running activities as nested sagas. *IEEE Data Engineering Bulletin*, 14(1), 1991.

[God96] C ; Godart, G. Canals, F. Charoy, P. Molli et H. Skaf. Designing and implementing coo : Design process, architectural style, lessons learned. *International Conference on Software Engineering (ICSE18)*, 1996.

[God99] C. Godart, O. Perrin et H. Skaf. coo : a workflow operator to improve cooperation modeling in virtual processes. *9th Int. Workshop on research Issues on Data Engineering Information technology for Virtual Entreprises (RIDEVE'99)*, 1999.

[God01] Claude Godart, Christophe Bouthier, Philippe Canalda, François Charoy, Pascal Molli, Olivier Perrin, Helene Saliou, Jean-Claude Bignon, Gilles Halin et Olivier Malcurat. Asynchronous coordination of virtual teams in creative applications (co-design orco-engineering) : requirements and design criteria. *Information Technologies for Virtual Enterprises*, 2001.

[Gre97] P. Grefen, J. Vonk, E. Boertjes et P. Apers. Two layer transaction management for workflow management application. *8th International Conference on Database and Expert Systems Applications (DEXA 97)*, 1997.

[Gri00a] D. Grigori. Understanding, predicting, and preventing exceptions in business processes. Rapport, Hewlett-Packard Labs, 2000.

[Gri00b] D. Grigori, H. Skaf-Molli et F. Charoy. Adding flexibility in a cooperative workflow execution engine. *The 8th International Conference on High - Performance Computing and Networking Europe HPCN Europe'2000*, 2000.

[Gri01a] D. Grigori, F. Casati, U. Dayal et M.C. Ming-Chien Shan. Improving business process quality through exception understanding, prediction, and prevention. *25th International Conference on Very Large Data Bases (VLDB)*, 2001.

[Gri01b] D. Grigori, F. Charoy et C. Godart. Anticipation to enhance flexibility of workflow execution. *12th International Conference on Database and Expert Systems Applications (DEXA'2001)*, 2001.

[Gri01c] D. Grigori, F. Charoy et C. Godart. Cooperative flexible workflow management. *the 13th International Conference on Control Systems and Computer Science*, 2001.

[Gri01d] D. Grigori, F. Charoy et C. Godart. Flexible data management and execution to support cooperative workflow : the coo approach. *The Third International Symposium on Cooperative Database Systems for Advanced Applications (CODAS'2001)*, 2001.

[Gui97] N. Guimarães, P. Antunes et A. P. Pereira. The integration of workflow systems and collaboration tools. *Nato Advanced Studies Institute on Workflow Systems and Interoperability,NATO ASI Series*. Springer-Verlag, 1997.

[Haa99] J. M. Haake et W. Wang. Flexible support for business processes : Extending cooperative hypermedia with process support. *Information and SoftwareTechnology*, 6(41), 1999.

[Hag99] C. Hagen et G. Alonso. Beyond the black box : Event-based inter-process communication in process support systems. *9th International Conference on Distributed Computing Systems (ICDCS 99)*, Austin, Texas, USA, 1999.

[Hal88] F.G. Halasz. Reflections on notecards : Seven issues for the next generation of hypermedia sytems. *Communications of the ACM*, 31(7), 1988.

[Han98] Y. Han, A. Sheth et C. Bussler. A taxonomy of adaptive workflow management. *Towards Adaptive Workflow Systems, CSCW'98 Workshop*, Seattle, USA, 1998.

[Hei96] P. Heimann, G. Joeris, C.A. Krapp et B. Westfechtel. Dynamite : Dynamic task nets for software process management. *the 18th International Conference on Software Engineering (ICSE18)*, pages 331–341, 1996.

[Hei99] P. Heinl, S. Horn, S. Jablonski, J. Neeb, K. Stein et M. Teschke. A comprehensive approach to flexibility in workflow management systems. *Joint Conference on Work Activities Coordination and Collaboration (WACC'99)*, San Francisco, 1999.

[Hew00a] Hewlett-Packard. *HP Changengine Process Design Guide. Edition 4.4.*, 2000.

[Hew00b] Hewlett-Packard. *HP Changengine Technical Reference Guide. Edition 4.4*, 2000.

[Hon88] M. Honda. Support for parallel development in the sun network software environment. *Second International Workshop on Computer-Aided Software Engineering*, 1988.

[HS00] H. Hans Schuster, D. Baker, A. Cichocki, D. Georgakopoulos et M. Rusinkiewicz. The collaboration management infrastructure. *ICDE*, pages 677–678, 2000.

[Hwa99] S. Hwang, S ; Ho et J. Tang. Mining exception instances to facilitate workflow exception handling. *DASFAA'99*, 1999.

[Jab94] S. Jablonski. Mobile : A modular workflow model and architecture. *4th International Working Conference on Dynamic Modeling and Information Systems*, 1994.

[Jab96] S. Jablonski et C. Bussler. *Workflow management - Modeling Concepts, Architecture and implementation.* International Thomson Computer Press, 1996.

[Jac93] M.L. Jaccheri et R. Conradi. Techniques for process model evolution in epos. *IEEE Transactions on Software Engineering*, 1993.

[Jah89] U. Jahn, K. Jarke, K. Kreplin et M. Farusi. Coauthor : A hypermedia group authoring environment. *1st European Conference on Computer Supported Cooperative Work*, 1989.

[Jaj97] S. Jajodia et L. Kerschberg, éditeurs. *Advanced Transactions Models and Architectures.* Kluwer Academic Publishers, 1997.

[Joe97a] G. Joeris. Change management needs integrated process and configuration management. *European Software Engineering Conference (ESEC'97)*, 1997.

[Joe97b] G. Joeris. Cooperative and integrated workflow and document management for engineering application. *8th Int. Workshop on Database and Expert Systems Applications, Workshop on Workflow Management in Scientific and Engineering Applications*, pages 68–73, 1997.

[Joe99] G. Joeris. Defining flexible workflow execution behaviors. *Enterprise-wide and Cross-enterprise Workflow Management - Concepts, Systems, Applications', GI Workshop Proceedings - Informatik'99, Ulmer Informatik Berichte Nr. 99-07, University of Ulm, 1999.*, 1999.

[Joe00] G. Joeris. 'modeling of flexible workflows and their decentralized enactment in flow.net. *International Journal of Computer Systems Science and Engineering (IJ-CSSE)*, 15(5) :327–343, 2000.

[Joear] H. Joeris, G.and Wache, O. Herzog et B. Gronemann. flow.net : Workflow support for inter-organizational engineering and production processes. *special issue on 'Information Management for Productivity Enhancement' of the International Journal of Agile Manufactoring (IJAM)*, to appear.

[Kai87] G.E. Kaiser et D.E Perry. Workspaces and experimental databases : Automated support for software engineering. *Conference of Software maintainance*, 1987.

[Kai89] G.E. Kaiser, D.E. Perry et W.M. Schell. infuse : Fusing integration test management with change management. *13th IEEE Computer Software and Applications Conference*, 1989.

[Kai93] G.E. Kaiser. Marvel 3.1 : A multi-user software development environment. *International Symposium on Logic Programming*, 1993.

[Kam98] M. Kamath et K. Ramamritham. Failure handling and coordinated execution of concurrent workflows. *14th International Conference on Data Engineering*, 1998.

[Kap98] G. Kappel, S. Rausch-Schott et W. Retschitzegger. Coordination in workflow management sytems - a rule-based approach. G. Conen, W. ; Neumann, éditeur, *Coordination technology for Collaborative Applications (Workshop at ASIAN'96)*, volume 1364. LNCS, 1998.

[Kar90] B. Karbe et N. Rasperger. influence of exception handling on the support of cooperative office work. *Sigois bulletin*, 1990.

[Kle98] M. Klein et Dellarocas, éditeurs. *Towards Adaptive Workflow Systems,Workshop within the Conference on Computer Supported Cooperative Work*, http ://ccs.mit.edu/klein/cscw98, 1998.

[Kli] R. Kling, K.L. Kraemer, J. P. Allen, Y. Bakos, V. Gurbaxani et M. Elliot. Transforming coordination : The promise and problems of information technology in coordination. to appear in Tom Malone, Gary Olson, and John B. Smith (Eds.) Coordination Theory and Collaboration Technology. Mahwah, NJ : Lawrence Erlbaum (in press).

[Kor88] H. Korth et G. Speegle. Formal method of correctness without serializability. *ACM SIGMOD International Conference on Management of Data*, 1988.

[Kri95] N. Krishnakumar et A. Sheth. Managing hetereogeneous multi-sytem tasks to support enterprise-wide operations. *Distributed and Parallel Databases*, 3(2), 1995.

[Leb94] D.B. Leblang. *Configuration Management*, chapitre The CMChallenge : Configuration management that Works. John Wiley and Sons, 1994.

[Leb96] D.B. Leblang. Managing the software development process with clearguide. *Software Configuration Management Workshop (SCM-7) at ICSE'97*, 1996.

[Lel88] M. Leland, R. Fish et R. Kraut. Collaborative document production using quilt. *ACM Conference on Computer-Supported Cooperative Work*, 1988.

[Ley00] F. Leymann et D. Roller. *ProductionWorkflow, COncepts and techniques.* Prentice-Hall, 2000.

[Liu98] C. Liu, M. Orlowska et H. Li. Automating handover in dynamic workflow environments. *10th International Conference on Advanced Information Systems Engineering CAISE'98*, number 1413 in LNCS, 1998.

[Lot97] Lotus Development Corporation. *Lotus Domino release 4.6 : Developer's Guide*, 1997.

[Mia98] Y. Miao et J. Haake. Supporting concurrent design by integrating information sharing and activity synchronization. *CE'98*, 1998.

[Mic96] J. Micallef et G. Clemm. The asgard system : Activity-based configuration management. *Software Configuration management Workshop (SCM-6) at ICSE'96*, 1996.

[Mic98] Microsoft Corporation. *Deploying Microsoft Exchange Server*, 1998.

[Mol96] Pascal Molli. *Environnements de Développement Coopératifs.* Thèse en informatique, Université de Nancy I – Centre de Recherche en Informatique de Nancy, 1996.

[Neu92] C. Neuwirth, D. kaufer, R. Chanhok et J. Morris. Flexible diff-ing in a collaborative writing system. *ACM Conference on Computer-Supported Cooperative Work*, 1992.

[Neu96] O. Neumann, S. Sachweh et W. Schafer. A high-level object-oriented specification language for configuration management and tool integration. *Software Process technology - Fifth European Workshop EWSPT'96*, 1996.

[Nut96] Gary J. Nutt. The evolution toward flexible workflow systems. *Distributed Systems Engineerin*, 1996.

[Ost98] L. Osterweil. Software processes are software too, revisited. *International Conference on Software Engineering*, pages 540–548, 1998.

[Pan97a] E. Panagos et M. Rabinovich. Escalations in workflow management systems. *DART'97*, 1997.

[Pan97b] E. Panagos et M. Rabinovich. Predictive workflow management. *NGITS'97*, 1997.

[Pro00] F. Provost et P. Domingos. Well-trained pets : Improving probability estimation trees. Rapport, Stern School of Business, 2000.

[Qui86] J.R. Quinlan. Induction of decision trees. *machine Learning*, 1986.

[Qui93] J.R. Quinlan. *C4.5 : Programs for Machine Learning*. Morgan Kaufmann, 1993.

[Rei97] M. Reichert et P Dadam. A framework for dynamic changes in workflow management systems. *8th International Workshop on Database and Expert Systems Applications (DEXA 97)*, pages 42–48, 1997.

[Rei98] M. Reichert et P Dadam. Adeptflex - supporting dynamic changes of workflows without losing control. *Journal of Intelligent Information Systems*, 10, 1998.

[Reu95] A. Reuter et F. Schwenkreis. Contracts - a low-level mechanism for building general-purpose workflow management-systems. *IEEE Data Engineering Bulletin*, 1995.

[Sad00] W. Sadiq, O. Marjanovic et M.E. Orlowska. Managing change and time in dynamic workflow processes. *International Journal of Cooperative Information Systems*, 9(1-2) :93–116, 2000.

[She93] A. Sheth et M. Rusinkiewicz. On transactional workflows. *IEEE Data Engineering Bulletin*, pages 37–40, 1993.

[Sik98] K. Sikkel, O. Neumann et S. Sachweh. Process support for cooperative work on the world wide web, 1998.

[Suc87] L. A. Suchman. *Plans and Situated Actions. the Problem of Human-machine Communication.* Cambridge University Press, 1987.

[Swe94] K.D. Swenson, R.J. Maxwell, B. Matsumoto, T. ans Saghari et K. Irwin. A business process environment supporting collaborative planning. *the Journal of Collaborative Computing*, 1, 1994.

[VdA98] W.M. Van der Aalst. Application of petri nets to workflow management. *The journal of Circuitus, Systems and Computers*, 8(1), 1998.

[Wac92] H ; Wachter et A. reuter. The contractmodel. In Elmagarmid [Elm92].

[Wan00] A. Wang, R. Conradi et C. Liu. Integrating workflow with interacting agents to support cooperative software engineering. *Software Engineering and Applications'2000 (SEA'2000)*, 2000.

[Wes96] M. Weske. Flexible modeling and execution of workflow activities. *31st Hawaii International Conference on System Sciences, Software Technology Track (Vol VII)*, 1996.

[Wes98] M. Weske. State-based modeling of flexible workflow executions in distributed environments. *3rd Biennial World Conference on integrated Design and process technology (IDPT'98)*, 1998.

[Wie96] W. Wieczerzycki. *Business Process Modeling*, chapitre Process Modeling and Execution in Workflow Management Systems by Event-Driven Versioning. Springer-Verlag, 1996.

[Wii91] U. Wiil. Using events as support for data sharing in collaborative work. *International Workshop on CSCW*, 1991.

[Wii92] U. Wiil et J. Leggett. Hyperform : using extensibility to developdynamic, open and distributed hypertext systems. *ACM Conference on Hypertext*, 1992.

[Wii93] U. Wiil. Experiences with hyperbase : A hypertext database supporting collaborative work. *ACM Sigmod Record*, 22(4), 1993.

[Yan88] Y. Yankelovich, B. haan, N. Meyrowitz et S. Drucker. intermedia : The concept and the construction of seamless information environments. *IEEE Computer*, 21(1), 1988.

Annexe A

Eléments du modèle de définition de workflow

Conteneurs On va noter l'ensemble des activités et procédés avec $\mathcal{H}$; c.a.d. $\mathcal{H} = N \cup P$ est l'ensemble de tous les activités et modèles de procédés. V est l'ensemble de toutes les données associées à tous les modèles de procédés (habituellement on a besoin seulement des données de procédés associées à un modèle de procédé particulier, mais pour la définition suivante on a besoin d'une portée plus large). $\mathcal{C}$ est l'ensemble de toutes les conditions. Soit $\rho(M)$ l'ensemble de parties d'un ensemble M, c.a.d. l'ensemble de tous les sous-ensembles de M.

Définition A.1 (Conteneur)
L'application i associe à chaque activité, modèle de procédé et condition, son conteneur d'entrée :

$$i : \mathcal{H} \cup \mathcal{C} \to \rho(V)$$

C'est à dire,

$$\forall X \in \mathcal{H} \cup \mathcal{C} : i(X) \subseteq V \; avec \; card \; i(X) < \infty$$

L'application o associe à chaque activité et modèle de procédé, son conteneur de sortie :

$$o : \mathcal{H} \to \rho(V)$$

C'est à dire,

$$\forall X \in \mathcal{H} : o(X) \subseteq V \; avec \; card \; o(X) < \infty$$

Activités

Définition A.2 (Activité)
L'ensemble de toutes les activités d'un modèle de procédé $P \in \mathcal{P}$ est noté N. Habituellement, un membre A de N est écrit comme un opérateur $A : i(A) \to o(A)$.

Définition A.3 (Implantation d'une activité)
Soit $\mathcal{E}$ l'ensemble de toutes les implantations possibles de toutes les activités ; donc, un membre de $\mathcal{E}$ peut être un programme ou un modèle de procédé. L'application $\Phi : N \to \mathcal{E}$ associe à chaque activité son implantation. Une implantation d'activité est une application :

$$\Phi(A) : \times_{v \in i(A)} DOM(v) \to \times_{v \in o(A)} DOM(v)$$

$DOM(v)$ est le domaine de l'élément de donné v.

Définition A.4 (Requête de personnel)
Si on note une requête de personnel par q, la requête va retourner au moment de temps $i \in \mathbb{N}$ un ensemble d'acteurs $q(i) \in \rho(\mathcal{A})$, ou $\mathcal{A}$ dénote l'ensemble de tous les acteurs. Ainsi, on définit comme suit : un membre de l'ensemble $\mathcal{Q} = \{q | q : \mathbb{N} \to \rho(\mathcal{A})\}$ est appelé requête de personnel. L'application $\Omega : N \to \mathcal{Q}$ est appelée attribution de personnel et elle associe à chaque activité une requête de personnel.

Définition A.5 (Condition de sortie)
L'application $\varepsilon : N \to \mathcal{C}$ associe à chaque activité un prédicat appelé condition de sortie. La condition de sortie $\varepsilon(A)$ de l'activité A a un conteneur d'entrée $i(\varepsilon(A)) \subseteq V$ de telle manière qu'une condition de sortie est considérée une fonction booléenne :

$$\varepsilon(A) : \times_{v \in i(\varepsilon(A))} DOM(v) \to \{0, 1\}$$

Définition A.6 (Connecteur de contrôle)
L'ensemble $E \subseteq N \times N \times \mathcal{C}$ s'appelle l'ensemble des connecteurs de contrôle du modèle de procédé $P \in \mathcal{P}$. Pour un connecteur de contrôle $(A, B, p) \in E$, le prédicat $p \in \mathcal{C}$ s'appelle condition de transition. Chaque condition de transition est considérée comme une fonction booléenne de son conteneur d'entrée $i(p) \subseteq V$:

$$p : \times_{v \in i(p)} DOM(v) \to \{0, 1\}$$

Attribution des activités Nous introduisons les symboles suivants :

- L'ensemble de toutes les conditions de transition de tous les connecteurs de contrôle qui pointent vers l'activité A est noté $C^{\leftarrow}(A) := \pi_3(\{e \in E | \pi_2(e) = A\})$
- L'ensemble de toutes les conditions de transition de tous les connecteurs de contrôle qui partent de l'activité A est noté $C^{\rightarrow}(A) := \pi_3(\{e \in E | \pi_1(e) = A\})$

π dénote l'application de projection des produits carthesiens et ses indices spécifient les composants sélectionnés du domaine de projection. Pour $1 \leq i_1, \ldots, i_k$, on a $\pi_{i_1,\ldots,i_k} : M \times \ldots \times M_n \to M_{i_1} \times \ldots \times M_{i_k}$

Définition A.7 (Activités de branchement et de jointure)
Une activité $A \in N$ est appelée activité de branchement si et seulement si elle a plus d'un connecteur de contrôle sortant, c.a.d.
$A \in N$ est une activité de branchement $\Leftrightarrow \exists e, f \in E : \pi_1(e) = \pi_1(f) = A \wedge e \neq f$
Une activité $A \in N$ est appelée activité de jointure si et seulement si elle a plus d'un connecteur de contrôle entrant, c.a.d.
$A \in N$ est une activité de jointure $\Leftrightarrow \exists e, f \in E : \pi_2(e) = \pi_2(f) = A \wedge e \neq f$
On note l'ensemble de toutes les activités de jointures avec N_* et on définit $N_\bullet := N - N_*$ l'ensemble des noeuds réguliers.

Définition A.8 (Condition de jointure)
Soit $\phi(A)$ l'ensemble de toutes les conditions booléennes dans la forme normale disjonctive des conditions de transitions des connecteurs de contrôle qui entrent dans l'activité A, c.a.d.

$$\phi(A) := \{ \bigvee_{1 \leq j \leq k} \bigwedge_{1 \leq i \leq l_j} p'_i | p'_i \in \{p, p | p \in \{C^{\leftarrow}(A)\}\}\}$$

Alors, $\Phi : N \to \bigcup_{A \in N} \phi(A)$ associe des conditions de jointure :
– $\forall A \in N : \Phi(A) \in \phi(A)$, et
– $\forall A \in N : \Phi = 1$ (un noeud a une condition de jointure triviale
Ainsi, une condition de jointure $\Phi(A)$ peut être vue comme une fonction booléenne

$$\Phi(A) : \times_{p \in C^{\leftarrow}(A)} \times_{v \in i(p)} DOM(v) \to \{0, 1\}$$

Flot de données

Définition A.9 (Application de connecteur de données)
Soit $A \in N$ une activité et soit $B \in N \cup \mathcal{C}$ une activité ou un prédicat. L'application

$$\Delta : N \times (N \times \mathcal{C}) \to \bigcup_{A \in N, B \in N \cup C} \rho(o(A) \times i(B))$$

qui satisfait les conditions
– $\Delta(A_1, A_2) \in \rho(o(A_1) \times i(A_2))$,
– $\Delta(A_1, A_2) \neq 0 \Rightarrow A_2$ est atteignable à partir de A_1,
– $\forall A_2 \in N : (x, z), (y, z) \in \bigcup_{A_1 \in N} \Delta(A_1, A_2) \Rightarrow x = y$,
est appelée application de connecteur de données. Un élément (v_1, v_2) s'appelle un connecteur de données. L'ensemble de tous les connecteurs de données D est défini comme

$$D := \{(A, B, \Delta(A, B)) \in N \times N \times \rho(V \times V) | \Delta(A, B) \neq \phi\}$$

La première condition spécifie qu'un connecteur de données peut lier seulement une donnée du conteneur de sortie de l'activité origine et une donnée du conteneur d'entrée de l'activité destination.

La deuxième condition permet à un connecteur de données d'exister seulement si le flot de contrôle connecte la source du connecteur de donnée avec la destination du connecteur de données.

La troisième condition interdit que deux connecteurs de données aient le même élément de donnée comme destination. Cette contrainte évite les conflits à la matérialisation d'un conteneur qui pourraient être provoqués si deux connecteurs de données fournissent des données différentes.

Conteneur d'entrée et de sortie d'un procédé Un modèle de procédé $P \in \mathcal{P}$ peut avoir un conteneur d'entrée $i(P) \subseteq V$ et un conteneur de sortie $o(P) \subseteq V$.

La définition suivante reflète comment les données sont échangées entre les conteneurs du procédé et ses activités.

DÉFINITION A.10 (APPLICATION DE CONNECTEURS DE DONNÉES POUR LE PROCÉDÉ)
Soit $P \in \mathcal{P}$ un modèle de procédé et soit N l'ensemble de toutes ses activités. L'application

$$\Delta^{\rightarrow} : N \rightarrow \bigcup_{A \in N} (\rho(i(P) \times i(A)) \cup \rho(o(A) \times o(P)))$$

est appelé l'application de connecteurs de données pour le procédé et satisfait les conditions suivantes :

1. $\forall A \in N : \Delta^{\rightarrow}(A) \in (\rho(i(P) \times i(A)) \cup \rho(o(A) \times o(P))$
2. $\forall B \in N : (x, z), (y, z) \in (\rho(i(P) \times i(B)) \cup \bigcup_{A \in N} \Delta(A, B) \Rightarrow x = y$
3. $(x, z), (y, z) \in \cup \bigcup_{B \in N} (\rho(o(B) \times o(P))$

Un élément (v_1, v_2) est appelé un connecteur de données.

La première condition assure qu'une application de connecteurs de données pour le procédé qui a comme origine ou destination l'activité A spécifie seulement des applications de données qui ont A comme origine, ou respectivement destination.

Pour éviter des opérations conflictuelles de copie à l'exécution, les conditions (2) et (3) interdisent d'avoir plus d'une application de donnée avec le même élément de donnée comme destination.

Annexe B

Liste de symboles

C	Ensemble de toutes les conditions
$C^{\leftarrow}(A)$	Ensemble de toutes les conditions de transition de tous les connecteurs de contrôle qui pointent vers l'activité A
$C^{\rightarrow}(A)$	Ensemble de toutes les conditions de transition de tous les connecteurs de contrôle qui partent de l'activité A
Δ	Application de connecteurs de données
$\vec{\Delta}$	Application de connecteurs de données pour le procédé
Δ_g	Application de connecteurs de données partagées
E	Ensemble de tous les connecteurs de contrôle d'un modèle de procédé
$\varepsilon(A)$	Condition de sortie de l'activité A
$i(X)$	Conteneur d'entrée de l'activité X
N	Ensemble de toutes les activités
$N_\bullet$	Ensemble de toutes les activités de jointure
N_*	Ensemble de toutes les activités régulières
$o(X)$	conteneur de sortie de l'activité X
$\rho(N)$	l'ensemble de parties de l'ensemble N
$\mathcal{P}$	Ensemble de tous les modèles de procédé
π_j	Application de projection du produit carthésien sur le composant j
$\Omega(A)$	L'attribution des acteurs à l'activité A
V	Ensemble de toutes les données
$\Phi(A)$	Condition de jointure de l'activité A
$\Psi(A)$	Implantation de l'activité A