Développement et validation des fonctionnalités de la plateforme de s-maintenance

1. Introduction

Une des caractéristiques de la plateforme de s-maintenance est de proposer des services dynamiques et d'assurer des fonctionnalités (auto-traçabilité, auto-apprentissage, autogestion, etc.) susceptibles de faire évoluer l'intelligence et les comportements de la plateforme en tenant compte de l'aspect dynamique des connaissances. En effet, la dynamicité de la connaissance est un enjeu essentiel en ingénierie des connaissances. Bachimont (Bachimont, 2004) souligne que l'ingénierie des connaissances manipule des inscriptions de connaissances qui permettent elles-mêmes « l'expression, la transmission et l'appropriation de la connaissance proprement dite ». Cette inscription de connaissances doit se prêter à une interprétation critique et à une (re)construction de la connaissance.

Ce chapitre est consacré à la mise en place de fonctionnalités de remise à jour de connaissances, en fonction des interactions entre composants et expérience des utilisateurs, plus précisément des traces numériques d'interaction entre utilisateur et plateforme. Ces traces nous permettront de comprendre les comportements des utilisateurs : chaque utilisateur doit mentionner pour quoi l'a-il fait ? Par conséquent, nous envisagerons de formaliser les décisions prises par rapport aux contraintes, afin de comprendre et réutiliser les expériences passées. Ces traces d'activités sont assurées par les applications (les services) de la plateforme. Elles permettront de formaliser dans la base de connaissances l'expérience associée à l''exécution des activités, ses entrées et ses sorties ainsi que les contraintes subies. Ces connaissances dans la base de connaissances permettent à ces applications (application de diagnostic, application de pronostic, etc.) d'avoir un retour d'expérience sur leurs fonctionnements. Ce retour d'expérience exploité par ces applications aide à proposer des services dynamiques faisant évoluer leur comportement en fonction de leurs expériences.

Les traces numériques d'interaction associées à un modèle, seront définies par Mille comme des traces modélisées. Alain Mille dans (Laflaquière, Prié, & Mille, 2008) défend que les traces puissent prétendre au statut d'inscription de connaissance. Il dit que : « en s'ouvrant à l'interprétation d'un observateur, les traces modélisées ne sont pas seulement une présentation de l'activité pour elle-même, elles deviennent potentiellement une ressource de cette activité au sein de l'environnement observé. »

Partant de ce postulat, nous développons les fonctionnalités d'auto-traçabilité, d'auto-apprentissage et d'autogestion en prenant appui sur ces inscriptions de connaissances, ceci à l'aide d'une ingénierie susceptible d'exploiter l'aspect dynamique des connaissances, et de réutiliser ces inscriptions à partir de l'expérience passée de ses utilisateurs, l'ingénierie des traces.

L'ingénierie des traces, faisant partie de l'ingénierie des connaissances, fournies une nouvelle forme de réutilisation d'expérience et nous semble la mieux adaptée pour fournir un support aux fonctionnalités d'auto-X par l'aspect dynamique des connaissances utilisées dans la plateforme et par conséquent des services.

A ces fins, nous proposons d'élaborer un système à base de traces (SBT) permettant de tracer les activités effectuées via la plateforme. En effet, l'analyse de ces traces a pour objectif de fournir un retour d'expérience, sous forme de règles de connaissances aussi bien pour l'utilisateur de la plateforme que pour les applications qu'elle intègre.

Ceci est réalisé via un SBT adapté à la s-maintenance, son architecture sera fourni dans la deuxième section. L'architecture de ce système prend appui sur l'ontologie de maintenance, et plus particulièrement la partie réservée au processus relatif aux différentes activités dans la plateforme. En effet, dans ce chapitre, nous faisons un zoom sur certains composants de l'architecture de s-maintenance : base de connaissances, le raisonneur, le coordinateur et le gestionnaire de traces modélisées.

La collaboration entre ces composants assurera les fonctionnalités d'auto-traçabilité et d'auto-apprentissage qui seront développées successivement dans les sections 3 et 4.

La section 5 porte sur l'exploitation des sorties (outputs) du SBT ainsi formé au profit de l'utilisateur et de la plateforme. La fonctionnalité d'autogestion sera développé et contribuera à créer un service de diagnostic dynamique.

Finalement, nous étudions dans la section 6 l'impact et la valeur ajoutée de ces fonctionnalités sur les performances de la maintenance et ceci par le biais d'un calcul d'indices de temps et de coûts. Une étude de cas en rapport avec l'entreprise TEM sera simulée.

2. Système à base de trace dans la plateforme de s-maintenance

2.1- Système à base de traces (SBT)

Une trace d'interaction est définie comme un objet contenant la représentation d'une partie de l'expérience entre un système et ses utilisateurs. De manière générale, une trace d'interactions peut contenir tout ce qui est en rapport avec l'activité menée par les utilisateurs sur un système (Cram, Jouvin, & Mille, 2007). Champin et al (Champin, Prie, & Mille, 2004) définissent la trace comme étant une séquence d'états et de transitions représentant l'activité de l'utilisateur. Il existe différents termes qui se réfèrent au concept de trace d'interactions comme les logs, les historiques, les traînées et les tracks (Mille, 2006). Laflaquière et al (Laflaquière, Prié, & Mille, 2008) considèrent que ces traces sont un support potentiel d'une expérience d'utilisation.

Ces traces d'interactions peuvent être réutilisées à deux fins : l'assistance et l'analyse. L'analyse des traces consiste à analyser l'activité qui a généré les interactions (Mille, 2006) sachant qu'il existe différentes techniques d'analyse comme la visualisation des données tracées (Rossi, Lechevallier, & El Golli, 2005), la transformation des traces (Georgeon, Mille, & Bellet, 2006), la détection de séquences fréquentes, etc.

En effet, la réutilisation des traces d'interactions est une approche de la réutilisation de l'expérience fondée sur des expériences passées. Ce type d'approche rentre dans le cadre de la deuxième génération de système expert qui ne se base pas que sur des connaissances générales mais qui dispose aussi d'un nouveau type de connaissance qui est l'expérience (Cram, Jouvin, & Mille, 2007).

Les traitements à appliquer aux traces, sont formalisés à l'aide de Systèmes à Base de Traces (SBT) (Cram, Jouvin, & Mille, 2007), système qui s'appuie sur trois phases principales : la collecte, souvent suivie d'une étape de prétraitement, l'analyse et l'exploitation (Bousbia, 2011). Afin de faciliter la mise en place d'un SBT, Settouti

et al, ont proposés dans (Settouti L. , Prié, Mille, & Marty, 2006) une architecture composée de trois systèmes interdépendants, les systèmes de collecte, de transformation et de visualisation (voir Figure 5.1).

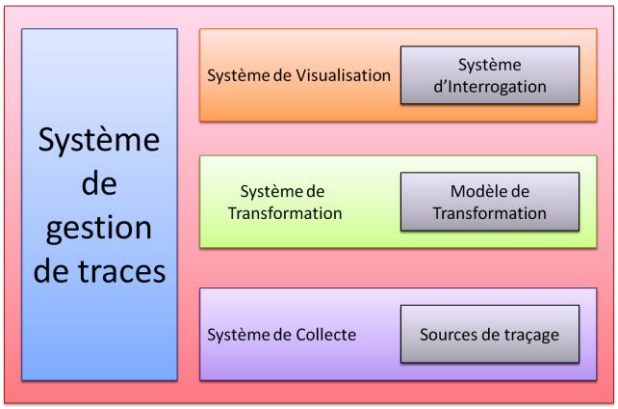


Figure 5-1 Architecture d'un SBT

Le système de collecte capte les interactions par l'intermédiaire de sources de traçage en créant une première trace. Ce système structure les traces collectées suivant une hiérarchie de classes observées appeler le modèle de trace (Cram, Jouvin, & Mille, 2007).

Le système de transformation constitue le cœur du SBT. Il permet de générer de nouvelles connaissances à partir des traces collectées. Le choix du modèle de transformation à appliquer dépend de l'objectif d'utilisation de cette trace.

L'ensemble des traces collectées et transformées est alors accessible par l'intermédiaire d'un système de visualisation (Bousbia, 2011) permettant d'exploiter, analyser et interpréter celles-ci (Cram, Jouvin, & Mille, 2007) et (Settouti L. , Prié, Mille, & Marty, 2006).

2.2- Système à base de traces adapté à la s-maintenance

Notre objectif est l'extraction des règles de connaissances à partir de l'activité de la plateforme. Le cas des activités portant sur les tâches décisionnelles prises par l'utilisateur semble intéressant. A cet effet, nous proposons un système à base de traces qui ne se limite pas uniquement à la visualisation, mais qui a l'ambition d'interpréter ces traces via les connaissances de la plateforme et ses modules d'apprentissage.

Pour répondre à cet objectif, l'architecture du SBT sera composée de deux systèmes principaux, un pour l'auto-traçabilité et l'autre d'auto-apprentissage. Le système d'auto-apprentissage est lui-même composé de deux systèmes, un concernant la collecte de trace et l'autre sa transformation.

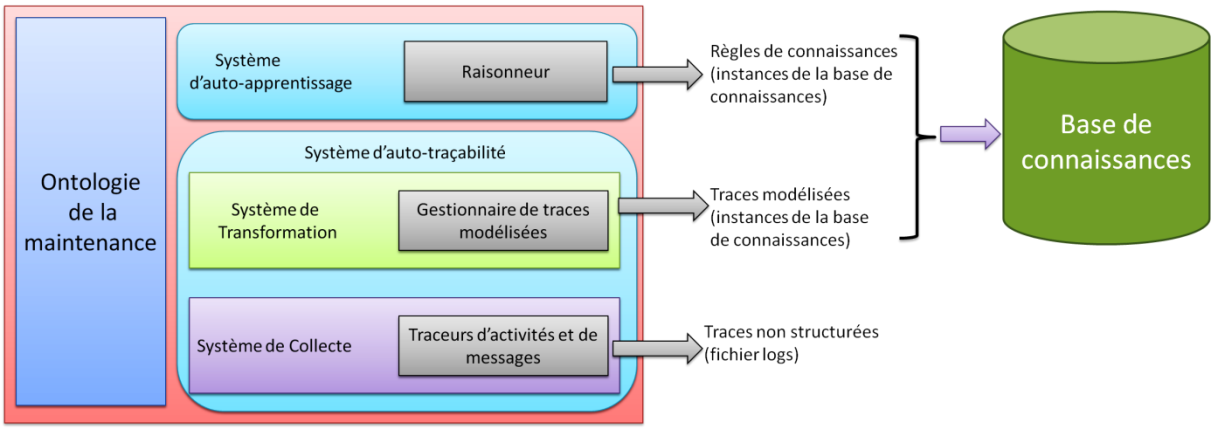


Figure 5-2 SBT adapté à la s-maintenance

Ce SBT prend appui sur l'ontologie du domaine de maintenance pour modéliser les traces, les interpréter et en extraire de la connaissance. Comme le montre la Figure 5.2, l'ontologie servira à tout niveau.

Le système de collecte est dirigé par le composant traceur d'activités et de messages qui collecte les traces d'interactions des activités et fournit des fichiers logs contenant des traces non structurées.

Soulignons que le système de transformation est dirigé par le composant gestionnaire des traces modélisées qui transforme les traces non structurées en traces modélisées à partir de l'ontologie, ensuite, il les enregistre comme instance dans la base de connaissances.

Remarquons que, nous avons ajouté aux systèmes de visualisation d'interrogation, un système d'interprétation et d'apprentissage. Ce système permet d'inférer de nouvelles règles de connaissances sur les activités des processus gérées par la plateforme, à partir des traces déjà enregistrées dans la base de connaissances.

La Figure 5.3 illustre le fonctionnement étape par étape de notre SBT en présentant les entrées et les sorties des différentes fonctionnalités impliquées dans ce système.

- **Etape 0 :** Grâce aux connaissances enregistrées dans la base de connaissances concernant l'exécution des processus (composés d'activités), **la fonctionnalité de gestion** orchestre l'exécution des opérations de la plateforme (labels 0, 0'et 0'').
- **Etape 1 :** Lors des interactions des opérateurs de maintenance avec la plateforme et l'exécution des activités dans celle-ci, ***la fonctionnalité de traçabilité*** collecte les traces d'interactions et les enregistre dans des fichiers logs (labels 0'' et 1).
- **Etape 2 :** ***la fonctionnalité de transformation*** récupère les fichiers logs, elle extrait des traces modélisées par un mapping avec le modèle de trace (l'ontologie de maintenance dans notre cas) et finalement elle enregistre ces traces modélisées comme de nouvelles instances dans la base de connaissances (labels 1 et 2).
- **Etape 3 :** ***la fonctionnalité d'interprétation et d'apprentissage*** récupère les instances concernant les traces modélisées des processus et les analyse en vue d'extraire de nouvelles connaissances sur ces processus sous forme de règles (label 3 et 4).

- **Etape 4 :** ***la fonctionnalité de validation et d'adaptation*** traite les règles issues de l'apprentissage pour les modéliser de façon qu'elles respectent la structure de la base de connaissance avant de les enregistrer dedans comme de nouvelles instances (label 4 et 5).

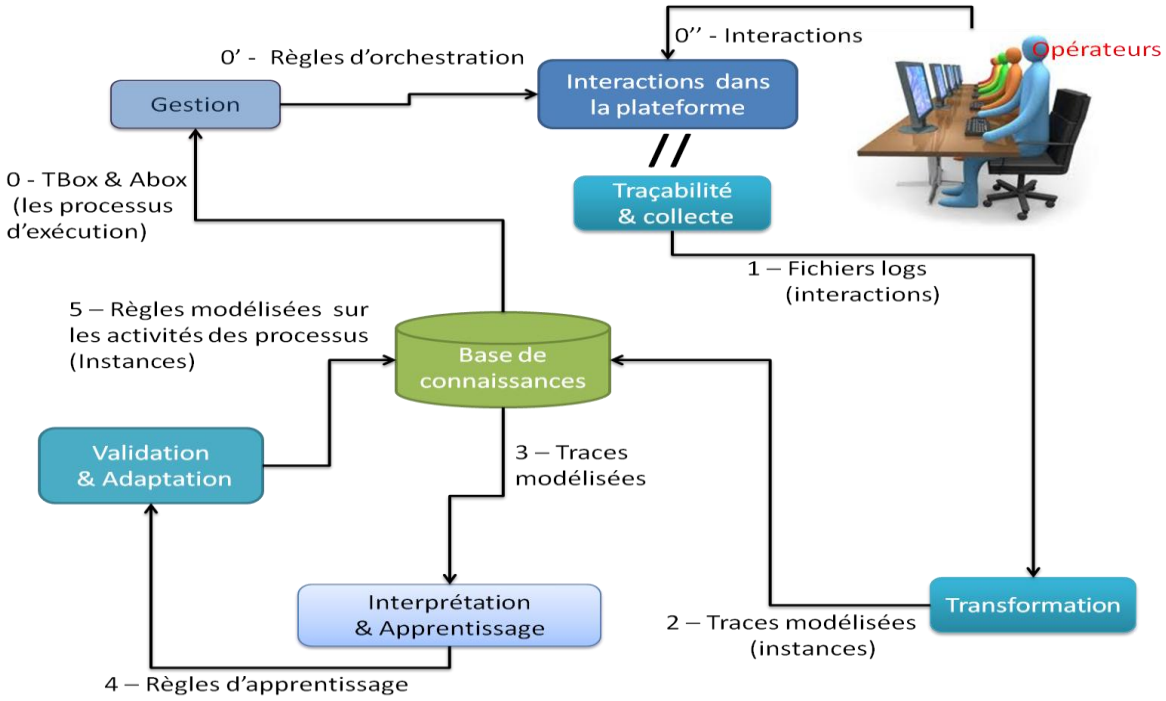


Figure 5-3 Boucle fermée du fonctionnement du SBT

3. Système d'auto-traçabilité : Collecte et transformation de traces

Le processus de traçabilité consiste à collecter l'ensemble des traces enregistrables en utilisant un environnement informatique sauvegardant les activités des opérateurs de maintenance. L'enregistrement de ces traces peut se référer à différents formats comme les fichiers logs, les *tracks* ou les *trails*. Selon Cram et al, les logs désignent des fichiers dans lesquels sont stockées au format textuel des traces de chaque requête ou opération effectuée sur un système, ou chaque erreur qu'il rencontre. Les logs représentent l'histoire d'interaction, mais comporte un certain nombre de défauts lors de la réutilisation de l'expérience (Cram, Jouvin, & Mille, 2007). Les *trails* sont notamment utilisés dans un contexte professionnel pour désigner la traçabilité d'un système, où le système a été conçu avec un mécanisme d'écoute sur lequel il est possible de se greffer. Quant aux *tracks*, ils désignent généralement des objets intentionnellement structurés pour représenter l'expérience d'une activité passée (Choquet & Iksal, 2007).

Selon (Bousbia, 2011) la collecte des données fournit des données qualifiées de « traces brutes » ou encore « traces primitives », difficilement exploitables en tant que telles. La création des traces à partir de ces logs est un processus complexe qui nécessite de nombreuses opérations (filtrage, recomposition en sessions, etc.). En effet, les logs peuvent contenir de grands volumes d'informations, parfois sans le contexte dans lequel les informations

ont été générées. Les données stockées ne sont pas structurées et pour la plupart inadaptées à l'objectif souhaité (Cram, Jouvin, & Mille, 2007).

Pour pouvoir mettre en pratique la réutilisation de l'expérience d'interaction, il faut donc pouvoir modéliser les traces (Cram, Jouvin, & Mille, 2007). Dans ce cadre, Settouti et al définissent formellement une trace comme « une collection d'observés qui peuvent être temporellement situés ». Un observé peut être considéré comme tout élément de l'environnement de l'utilisateur (une entité, une action) qui fait sens dans l'observation de son activité (Settouti L.-S. , Prié, Marty, & Mille, 2007).

Selon (Cram, Jouvin & mille, 2007), chaque observé de la trace est une instance de classe d'observé appartenant à une structure hiérarchique de classes d'observées appelées le modèle de trace. En outre, les observés de la trace sont reliés entre eux via les relations instances définies dans le modèle de trace. Ce modèle de trace peut être lié à une ontologie définissant les interactions utilisateur/système.

Pour tracer une activité à partir de la plateforme de s-maintenance, nous devons associer un modèle de trace. Or, nous savons que le modèle de processus de IMAMO (vue de processus) peut être exploité en tant quemodèle de trace pour les activités. Cette vue de processus sera développée dans la section 3.2 juste après la section 3.1 portant sur le fonctionnement du processus de traçabilité du système d'auto-traçabilité.

3.1- Processus de traçabilité du système d'auto-apprentissage

Le système d'auto-traçabilité consiste à collecter l'ensemble des traces enregistrables en utilisant un environnement informatique sauvegardant l'activité de l'opérateur de maintenance. Cet environnement informatique doit donc suivre des processus d'exécution dans la plateforme de s-maintenance. Comme déjà mentionné, les fichiers logs sont un bon outil pour sauvegarder les traces mais ils sont contraints par la complexité de leur structure et de leur analyse. Pour cela, dans le processus de traçabilité (collecte et transformation) portant sur le fonctionnement de ce système que nous présentons à la Figure 5-5, nous exploitons les fichiers logs ainsi que les traces modélisées. Donc, il y a une transformation du fichier logs à une trace modélisée par différents modules et finalement ces traces modélisées sont enregistrées dans la base de connaissance. La Figure 554 fournie un exemple d'instances de traces modélisées enregistrées dans la base de connaissances en langage PowerLoom.

```
(Assert (Activity A235))
(Assert (has-Reference-step A235 Validate-Alarm))
(Assert (ActivityInPutOutPut AIO569))
(Assert (has-Reference-Transition AIO569 Alarm.Measure))
(Assert (has-Input A235 AIO569))
(Assert (ActivityInPutOutPut AIO589))
(Assert (has-Reference-Transition AIO589 Alarm-Valid))
(Assert (has-Output A235 AIO589))
```

Figure 5-4 Exemple de mise à jour de la base de connaissances

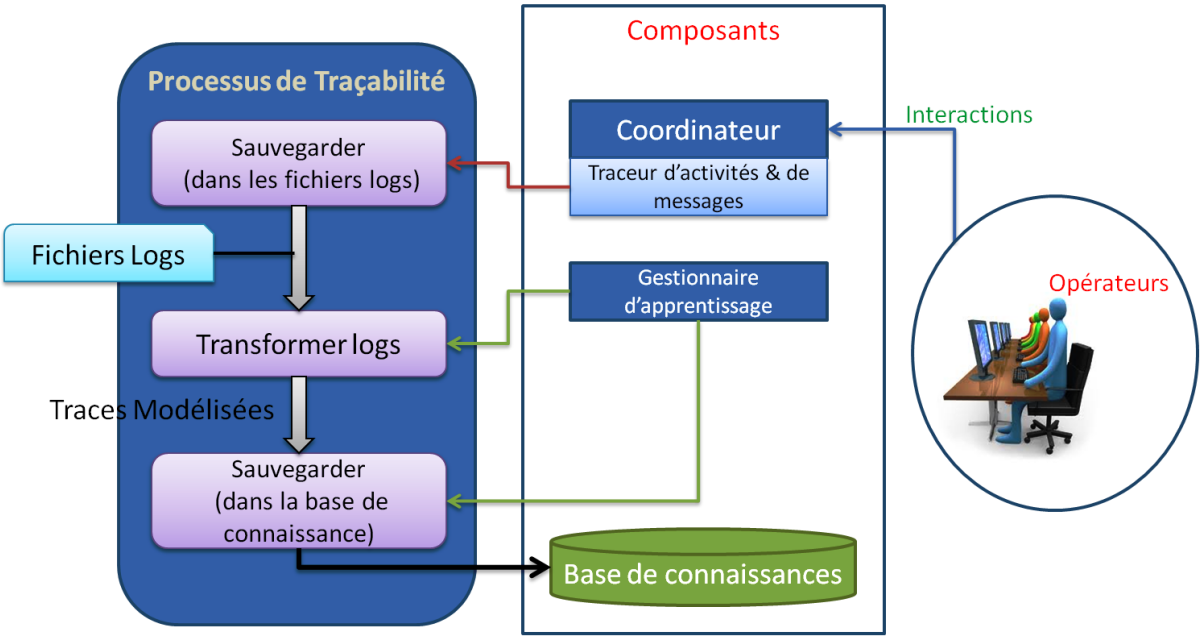


Figure 5-5 Activités du processus de traçabilité

La Figure 5-5, illustre que le *«Traceur d'activités et de messages »* est le sous composant du *Coordinateur* qui prend en charge la collecte des traces dans les fichiers logs pour les transférer ensuite au composant « Gestionnaire de traces modélisées). Ce dernier prend en charge la transformation des traces brutes des fichiers logs en traces exploitables par l'extrait des concepts du modèle de trace. De plus, ce modèle permet la structuration et la sauvgarde dans la base de connaissances comme de nouvelles instances.

3.2- Modèle de trace dans IMAMO

Cram et al affirment que les processus d'interaction entre l'humain et la machine se font à plusieurs niveaux : des relations de composition peuvent exister entre les interactions de différents niveaux (e.g., un projet est fait de réunions, des réunions sont composées de prise de parole) (Cram, Jouvin, & Mille, 2007). Hilbert et Redmiles classent ces interactions selon six niveaux d'abstraction (Hilbert & Redmiles, 2000) allant des manipulations de la souris ou du clavier jusqu'aux traces à analyser aux niveaux des tâches relatifs au domaine. Le niveau le plus abstrait (niveau 6) décrit des interactions relatives aux tâches (activités) réalisées (e.g., faire un diagnostique, corriger un bug, etc.). Le niveau que nous exploitons dans notre SBT car l'objectif du processus de traçabilité est d'apprendre à partir des activités des processus. La collecte sera donc faite sur ce sixième niveau. Ainsi, le modèle de trace que nous devons mettre en place pour le processus de transformation doit porter sur les activités des processus de maintenance gérées par les utilisateurs de la plateforme.

En effet, l'idée de conceptualiser les processus et les activités dans la « vue des processus de IMAMO » est motivée par le besoin d'avoir une conceptualisation des interactions métier pour les enregistrer dans la base de connaissances et les exploiter dans la phase d'analyse des traces grâce au moteur de raisonnement.

3.2.1- Vue de processus

La Figure 5-6 présente une partie de cette vue de processus et le tableau 1 présente le dictionnaire de données (la vue globale des processus est présentée dans l'annexe ONTOL ainsi que le dictionnaire de données détaillé).

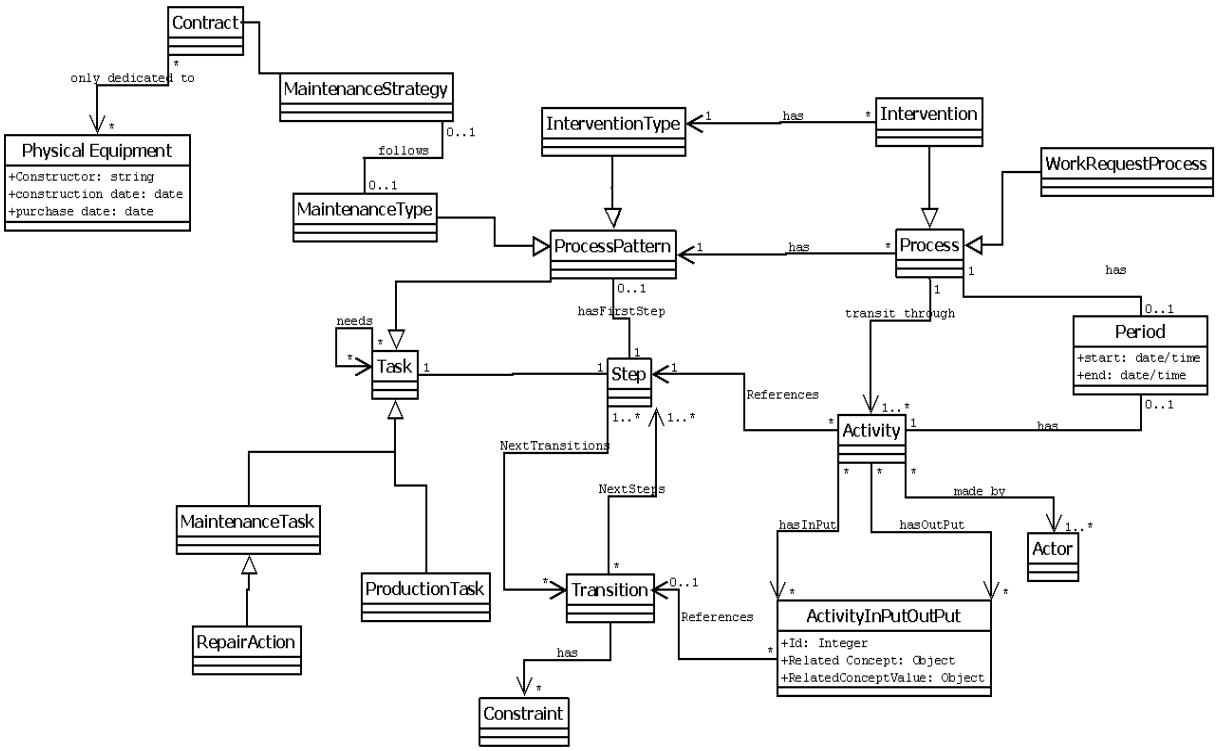


Figure 5-6 Vue des processus dans IMAMO

Tableau 5-1 Dictionnaire de données de la vue des processus

Concept	Termes français	Termes anglais	Description
process;	Processus;	Process;	Séquence d'activités interdépendantes et liées, qui, à chaque étape, consomme une ou plusieurs ressources pour convertir les entrées en sorties, tout en respectant un modèle de processus (le modèle de processus présente le modèle général du processus). Ces sorties de transition servent comme entrées de transition pour l'étape suivante jusqu'à ce qu'un but connu soit atteint.
process pattern;	Modèle de processus; Pattern de processus;	Process pattern;	Motif qui décrit une série d'actions. Un modèle est une description d'une solution générale à un problème commun ou une question à partir de laquelle une solution détaillée à un problème spécifique peut être déterminée.
maintenance type;	Type de maintenance;	Maintenance type;	Type spécifique de modèle de processus concernant la méthode pour effectuer la maintenance et l'orchestration des processus utilisés en vue d'atteindre les objectifs de la stratégie de maintenance. Il ya 12 types possibles de maintenance qui sont: maintenance préventive, maintenance planifiée, maintenance prédéterminée, maintenance basée sur les conditions, maintenance prédictive, maintenance corrective,

			maintenance à distance, maintenance différée, maintenance immédiate, maintenance en ligne, maintenance sur site et maintenance de l'opérateur. (la traduction en anglais est la suivante : Preventive maintenance, Scheduled maintenance, Predetermined maintenance, Condition based maintenance, Predictive maintenance, Corrective maintenance, Remote maintenance, Deferred maintenance, Immediate maintenance, On line maintenance, On-site maintenance and Operator maintenance.)
maintenance task;	Tâche de maintenance;	Maintenance task;	Type de tâche spécifique concernant une partie des travaux de maintenance (par exemple, réparation, remplacement, inspection, lubrification, etc.)
intervention type;	Type d'intervention;	Intervention type;	Type spécifique de modèle de processus. Il présente le modèle générique d'une intervention.
task;	Tâche;	Task;	Partie de travail effectuée par un acteur dans le cadre de ses fonctions.
repair action;	Action de reparation;	Repair action;	Type spécifique de tâches de maintenance définie comme une action physique prises pour rétablir la fonction requise d'un équipement physique défectueux.
production task;	Tâche de production;	Production task;	Type de tâche spécifique qui se termine par un produit discret ou un résultat qui est observable et mesurable.
constraint;	Contrainte;	Constraint;	Condition restrictive pour contrôler les transitions entre les étapes.
activity;	Activité;	Activity;	Unité d'organisation à exécuter une action spécifique. Une activité est l'exécution d'une tâche que ce soit une activité physique ou l'exécution de code. Il présente l'activité exercée par l'acteur dans le monde réel.
step;	Étape;	Step;	Manœuvre effectuée dans le cadre de progression vers l'état d'avancement d'un processus. Elle est référencée par une activité.
work request process;	Processus de demande d'intervention;	Work request process;	Type spécifique de processus lancé automatiquement ou par un acteur lors de la réception d'une demande de travail. Il permet de gérer la demande de travail jusqu'au résoudre le problème d'origine de l'événement déclencheur et la fin du processus d'intervention, y compris l'édition du rapport d'intervention.
Transition	Transitions ;	Transitions ;	Passage d'une étape à une autre dans le cadre d'un processus. Elle constitue une étape intermédiaire qui est envisagée comme un simple intermédiaire entre deux étapes dans un processus. C'est une connexion entre deux étapes. Une transition fait passer d'une étape à une autre représente la réponse au événement particulier. Une transition peut être un ensemble de valeurs ou événement déclencheur.
AcitivityInPutOutPut	Entrée Sortie d'une activité ;	Activity Input Output	Les paramètres, les valeurs et/ou les événements d'entrées ou déclencheurs de l'exécution d'une activité. Ces paramètres peuvent être les sorties de l'exécution d'autres activités. Le résultat d'exécution d'une activité (la sortie) est l'entrée d'une autre activité. Par ailleurs, une ActivityInPutOutPut est le lien de passage d'une activité à une autre. Par conséquent, chaque ActivityInPutOutPut peut faire référence à une Transition.

Un processus ***(Process)*** est une séquence d'activités ***(Acitivity)*** interdépendantes et liées faisant référence à des étapes ***(Step)***, qui, à chaque activité, consomme une ou plusieurs ressources pour convertir les entrées en sorties ***(ActivityInPutOutPut)***, tout en respectant un modèle de processus ***(Process Pattern)***. Ces sorties faisant

référence à des transitions ***(Transition),*** servent comme entrées qui elles aussi font référence à des transitions pour l'activité suivante jusqu'à ce qu'un but connu soit atteint (la clôture du processus).

Chaque *«Process»* fait référence à un *« Process Pattern »*, chaque *« Activity »* fait référence à un *« Step »* et chaque *« ActivityInPutOutPut »* fait référence à une *« Transition »*. Les instances des concepts processus et activité sont les interactions réelles faites (les traces enregistrées) dans la plateforme de s-maintenance et tracées par le processus de traçabilité. En outre, chaque activité est effectuée par un ou plusieurs acteurs (utilisateurs de la plateforme) qui sont définis par les instances du concept *« Actor »*. En conclusion, comme indiqué dans la Figure 5-7, le modèle de traces peut être résumé dans les concepts Process, Activity, Actor et ActivityInPutOutPut ainsi que les relations qui les structurent.

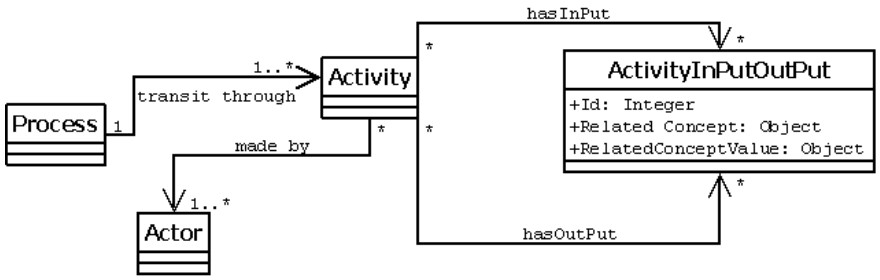


Figure 5-7 Modèle de Traces

3.2.2- Exemple illustratif

Pour mieux comprendre ce modèle, nous fournissons un exemple sur la maintenance conditionnelle. En effet, «Maintenance conditionnelle» est une instance du concept *« Maintenance Type »* qui est un sous-concept de *« Process Pattern »*. En outre, toutes les étapes et les transitions du processus de la «Maintenance conditionnelle» sont aussi instanciées.

Dans la Figure 5-8, nous présentons une illustration schématique de l'instanciation du concept *« Process Pattern »*. Nous faisons un focus sur le processus de gestion (présentation du «Process Pattern» de traitement de la demande de travail « work request ») qui fait référence à une étape (*Step*) dans le processus de maintenance conditionnelle (instance de Process Pattern).

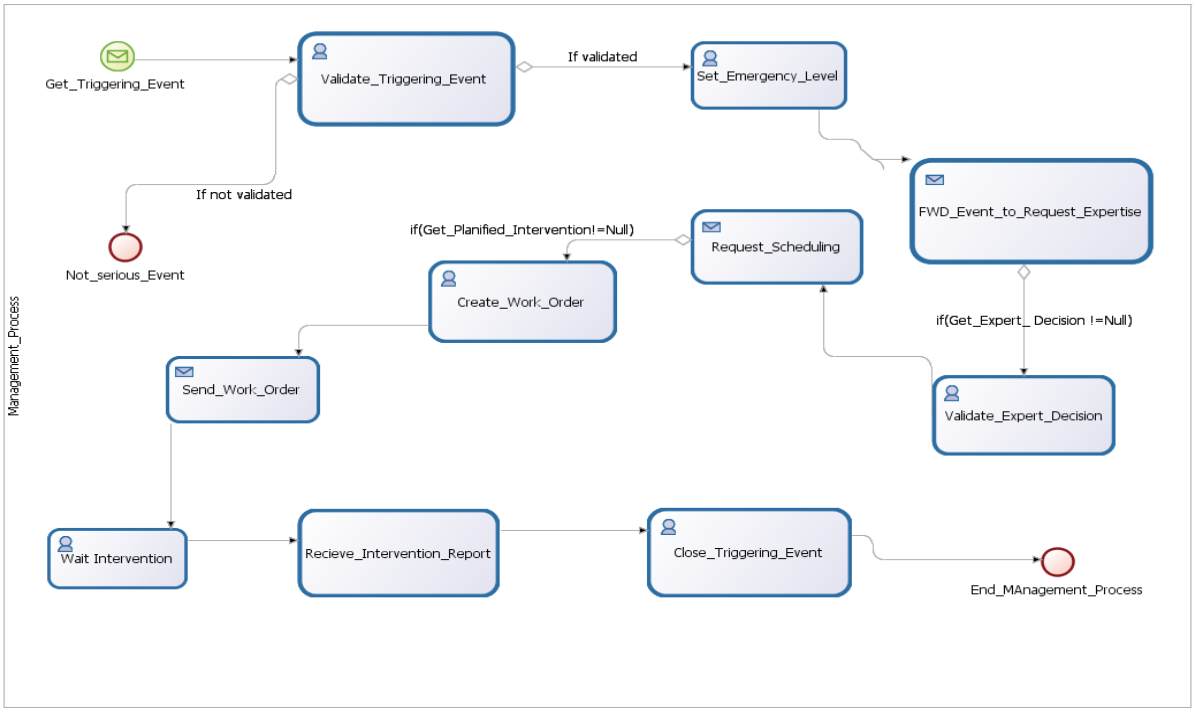


Figure 5-8 Exemple schématique d'une instance de *« Process Pattern »* (*management process*)

Les instructions PowerLoom suivantes présentent une partie de la base de connaissances concernant les processus et leurs instances.

```
(DEFCONCEPT Task)
(DEFCONCEPT Process-Pattern (?T Task))
(DEFCONCEPT Maintenance-Type (?PT Process-Pattern))
(DEFCONCEPT Step)
(DEFCONCEPT First-Step (?S Step))
(DEFCONCEPT Transition)
(DEFCONCEPT Next-Step (?S Step))
(DEFCONCEPT Constraint)

(Defrelation (StepInPut ((?T Transition) (?NS Step)))
(Defrelation (hasFirstStep ((?MT Maintenance-Type) (?FS First-Step)))
(Defrelation (StepOutPut ((?FS Step) (?T Transition)))
(Defrelation (Refrences ((?S Step) (?T Task)))

(Assert (Maintenace-Type Conditional-Maintenance)
(Assert (Maintenace-Type Predective-Maintenance)
(Assert (Maintenace-Type Systematic-Maintenance)
(Assert (Maintenace-Type Systematic-Maintenance)
(Assert (Step Monitoring-Process)
(Assert (Step Prognostic)
(Assert (Step Scheduling-Process)
(Assert (Management-Process)
```

```
(Assert (Transition Alarm)
(Assert (Transition RUL)
(Assert (Transition Notification-Date)

(ASSERT (hasFirstStep Conditional-Maintenance Monitoring-Process))
(ASSERT (hasFirstStep Predective-Maintenance Prognostic))
(ASSERT (hasFirstStep Systematic-Maintenance Scheduling-Process))

(ASSERT (StepOutPut Monitoring-Process Alarm))
(ASSERT (StepOutPut Prognostic RUL))
(ASSERT (StepOutPut Scheduling-Process Notification-Date))

(ASSERT (StepInPut RUL Management-Process))
(ASSERT (StepInPut Alarm Management-Process))
(ASSERT (StepInPut Notification-Date Management-Process))
```

Ainsi, nous rappelons que PowerLoom a un optimiseur de requêtes statiques et dynamiques qui est similaire aux optimiseurs utilisés dans les systèmes de bases de données. PowerLoom possède également une interface de base de données relationnelle qui lui permet d'utiliser la puissance de traitement des grandes bases d'instances (Chalupsky, MacGregor, & Russ, 2010). Ces avantages permettent la création de différentes vue de données, comme dans les bases de données grâce à des requêtes de jointure entre des concepts et des relations. Une vue est constituée d'une requête stockée accessible comme une table virtuelle composée de l'ensemble de résultats d'une requête (Bertino, 1992). Le tableau 5-2 présente un exemple d'une vue joignant chaque «Maintenance Type » avec sa première étape appelée « First Step » ainsi que la «Transition» entre la première et la deuxième étape («Next Step») dans IMAMO.

Tableau 5-2 Exemple du pattern "Maintenance Type"

Maintenance Type	**First Step**	**Transition**	**Next Step**
Conditional Maintenance	Monitoring process	Alarm	Management Process
Predictive maintenance	Prognostic	RUL	Management Process
Systematic maintenance	Scheduling	Notification (Date)	Management Process

Notre étude de cas se focalisera sur ce processus de gestion "Management Process" dans la maintenance conditionnelle pour illustrer le fonctionnement du système d'auto-apprentissage ainsi que la fonctionnalité d'autogestion dans la plateforme de s-maintenance.

Ainsi, comme hypothèse de travail, nous considérons que la plateforme de s-maintenance est utilisée pour gérer la maintenance des machines et des équipements industriels sur différents sites distribués un peu partout en France. Dans ces différents sites (parcs d'équipement), certains équipement font partie du même groupe d'équipement et partagent ainsi un même modèle d'équipement et de composants critiques de même capteurs à surveiller.

4. Système d'auto-apprentissage : Interprétation et Analyse des traces

Une fois les données de traces sont structurées et sauvegardées comme nouvelles instances dans la base de connaissance, l'étape d'apprentissage est lancée pour interpréter celles-ci et apprendre. Le processus d'auto-apprentissage est le processus de fonctionnement qui automatise le module d'apprentissage et d'interprétation dans l'étape 3. Selon Nilsson, l'apprentissage, comme l'intelligence, couvre un large éventail de processus qui est difficile à définir précisément. Nilson reprend sa définition de l'apprentissage les phrases du dictionnaire : « acquisition des connaissances, compréhension des compétences, instruction ou expérience par l'étude », et « la modification d'une tendance comportementale par l'expérience » (Nilsson, 1998). L'apprentissage automatique (machine learning en anglais) se réfère à un système capable d'acquérir et d'intégrer automatiquement des connaissances. La capacité des systèmes à apprendre de l'expérience, de la formation, de l'observation analytique, et d'autres moyens, résulte d'un système qui peut s'auto-améliorer en permanence et de ce fait, présente l'efficience et l'efficacité.

Un système d'apprentissage automatique commence habituellement avec une certaine connaissance organisée afin qu'il puisse interpréter, analyser et tester les connaissances acquises (worldofcomputing, 2011). Mettre en place des systèmes apprenants automatiquement sans intervention humaine, c'est mettre en place des systèmes capables d'auto-apprendre (Xu, Xu, Xiao, Zhou, Liu, & Jiang, 1995). Ainsi, tant que les techniques d'apprentissage automatique sont considérées comme le cœur de tout processus d'auto-apprentissage, un système d'auto-apprentissage est caractérisé par l'auto-ajustement de ses comportements en fonction des connaissances apprises (Lowe & Shirinzadeh, 2005).

Rappelons que la plateforme de S-maintenance est un système auto-apprenant (assure la fonctionnalité d'auto-apprentissage), avec la possibilité de faire évoluer son degré d'intelligence grâce à la dynamique de sa base de connaissances.

La fonctionnalité d'auto-traçabilité dans la plateforme se focalise sur l'ensemble de concepts suivant : *Process Pattern* (modèle de processus), *Process* (processus), *Step* (étape), *Activity* (activité), *Transition* et *Constraint* (contrainte) ainsi que les différentes relations entre ces concepts et leurs instances. Par conséquent, il sera possible pour le processus d'apprentissage de comprendre les différentes relations et interactions pour conclure certaines dépendances permettant de générer des règles pour des nouveaux modèles de processus.

Quoique, même après le prétraitement et la structuration dans le processus de traçabilité, les traces collectées sont généralement volumineuses et parfois non expressives. De ce fait, le processus d'apprentissage (d'interprétation) devient difficile pour l'analyste à cause de la complexité de la tâche. Pour cela, le module d'apprentissage associé à cette phase d'analyse, traite les traces en utilisant différentes méthodes d'analyse et d'apprentissage dont les méthodes statistiques et de fouille de données (data mining) qui restent les plus utilisées (Hilbert & Redmiles, 2000).

Dans cette étude, nous adoptons l'approche arbre de décision pour sa facilité à manipuler des données « symboliques[44] » et numériques. Nous rappelons succinctement qu'un arbre de décision affecte une classe (ou sortie) à un modèle d'entrée en filtrant le modèle à travers des tests dans un arbre de décision ce qui permet d'obtenir et que nous obtenons des règles mutuellement exclusives et exhaustives (Nilsson, 1998).

Nous avons adopté l'algorithme C4.5 (Quinlan, Induction of Decision Trees, 1990) et (Quinlan, C4.5: Programs for Machine Learning, 1993) issus de la plateforme Weka[45] pour créer un arbre de décision basé sur un ensemble de données étiquetées d'entrée. Le système C4.5 se compose de quatre programmes principaux: 1) le générateur d'arbres de décision ('C4.5') qui construit l'arbre de décision, 2) le générateur de règle de production ('c4.5rules') qui génère de règles de production à partir de l'arbre non élagué , 3) l'interpréteur d'arbres de décision qui classifie les éléments en utilisant un arbre de décision et 4) l'interpréteur de règles de production qui classifie les éléments en utilisant un ensemble de règles.

Les sorties du C4.5 constituent le bon arbre de décision, les tables d'erreurs de formation et des tests et la matrice de confusion. Dans l'architecture de la plateforme de s-maintenance que nous avons proposée, cet algorithme est inclu dans le composant *« Raisonneur »*.

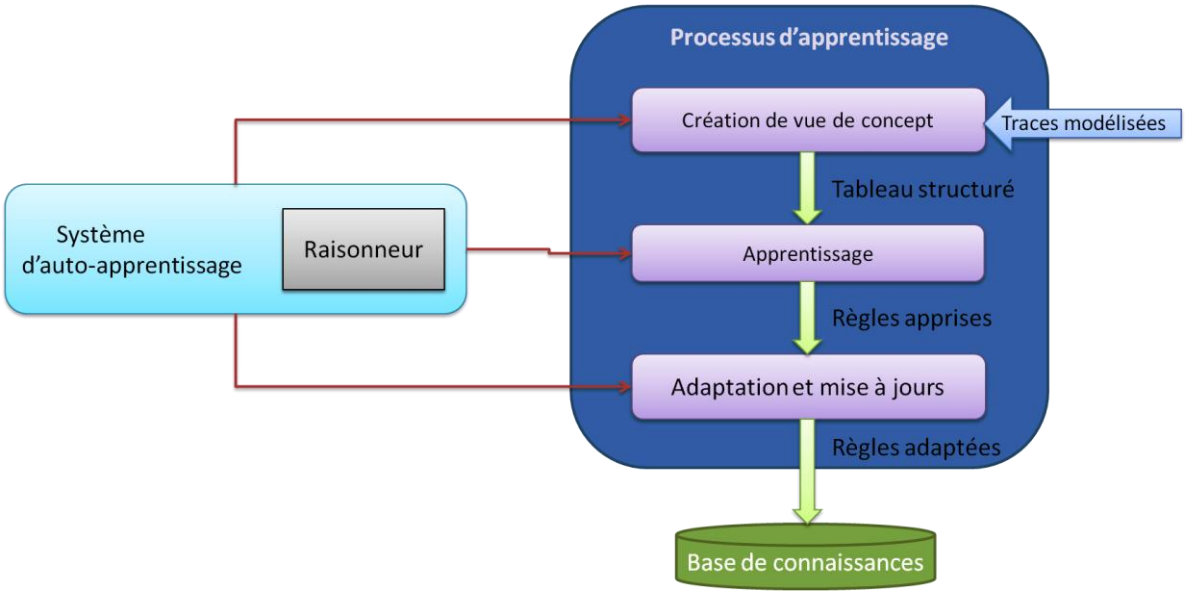


Figure 5-9 Activités du processus d'apprentissage

Tel que mentionné par Xu et al (Xu, Xu, Xiao, Zhou, Liu, & Jiang, 1995), un processus d'apprentissage doit satisfaire à la condition de cohérence qui signifie que le résultat d'apprentissage doit être compatible avec les connaissances d'origine déjà existantes dans la base de connaissances. Notre processus d'apprentissage est principalement basé sur les modèles de connaissances de IMAMO et les connaissances de la base de connaissances de la plateforme de s-maintenance. Comme indiqué dans la Figure 5-9, la première étape dans ce processus porte sur la création à partir de la base de connaissances de données contenant les connaissances

[44] *Les données sont symboliques* dans chaque case du tableau de données symboliques, on ne trouve pas nécessairement une seule valeur pour qu'elle soit quantitative ou qualitative.

[45] http://wekaclassalgos.sourceforge.net/

nécessaires à analyser. Cette étape est effectuée afin de présenter les connaissances nécessaires sous une forme structurée (tableau avec schéma spécifique). Puis, à partir de la deuxième vue, les différents fichiers nécessaires pour l'algorithme C4.5 sont créés de façon automatique dans la deuxième étape. Cette étape d'apprentissage consiste à exécuter l'algorithme C4.5 par le composant *« Raisonneur »*. Selon les pourcentages de la classification correcte ainsi que les pourcentages d'erreur relative, les résultats d'apprentissage sont validés ou pas. Finalement, s'ils sont validés, la troisième étape consiste à adapter les règles résultantes de cet apprentissage et par la suite à mettre à jour de la base de connaissances.

4.1- Création des vues

Comme présenté dans le modèle de processus dans IMAMO, chaque *«Process»* fait référence à un *« Process Pattern »* et chaque *« Activity »* fait référence à un *« Step »*. Les instances des concepts processus et activité sont les interactions réelles faites (les traces enregistrées) dans la plateforme de s-maintenance et tracées par le processus de traçabilité. En outre, chaque activité est effectuée par un ou plusieurs acteurs (utilisateurs de la plateforme) qui sont définis par les instances du concept *« Actor »*.

Les connaissances à la base de l'exercice d'apprentissage sont les instances des concepts activité et / ou processus. L'apprentissage, dans un soucis de génération a été appliquée aux concepts activité et/ou processus, et non pas à leurs instances.

L'intérêt de transformer et d'adapter les traces modélisées enregistrées dans la base de connaissances est de permettre au processus d'apprentissage analysant ces traces d'avoir des traces dans une forme adaptée à sa manière de fonctionnement.

Par conséquent, dans un premier temps, le composant *raisonneur* construit une vue contenant les instances, puis dans un second temps, il génère une seconde vue contenant des instances de concepts généraux, chaque instance d'activité ou de processus est remplacée par l'instance qui y fait référence sur les concepts *« Process Pattern »* (modèle de processus) ou *« Step »* (étape). Les tableaux 5-3 et 5-4 illustrent cette création de vues avec explication à l'appui. Il est à noter que les schémas de données des vues sont préalablement conçus dans le système en respectant le modèle ontologique de IMAMO. Les schémas proposés sont inspirés de la table de transition d'état qui est un tableau montrant dans quel état une machine à états finis se déplacera vers, en se basant sur l'état actuel et les inputs (Breen, 2005).

Le schéma de la première vue est présenté par un 5-uplet contenant l'activité concernée avec ses entrées/sorties, les activités sont les suivantes dans le processus de gestion d'un équipement physique :

< *ActivityInPut.Id, Activity, ActivityOutPut.Id, NextActivity, RelatedPhysicalEquipment* >.

Tableau 5-3 . Vue de premier niveau du processus d'apprentissage (vue physique)

ActivityInPut.Id	Activity	ActivityOutPut	NextActivity.Id	RelatedPhysicalEquipment
234	A982P22	536	A1236P22	P456
536	A1236P22	589	A2356P22	P456
789	A254P11	465	A269P11	Pu342
865	A11P18	965	A36P18	I231
965	A36P18	136	A39P18	I231
136	A39P18	245	A85P18	I231

Le tableau 5-3 ne fournit aucune information exploitable telle quelle. Nous allons donc raisonner au niveau des instances de concepts modèle et non au niveau des instances des concepts physique, grâce à l'ontologie IMAMO.

En ce qui concerne le deuxième niveau de vue, le schéma de données est principalement composé par le 8-uplet suivant :

< *ActivityInPut.Concept, ActivityInPut.Val, Step, ActivityOutPut.Concept, ActivityOutPut.Val, Constraint, NextStep, ConcernedEquipmentModel*>.

Cette deuxième vue est générée à partir de la première.
Step et *NextStep* dans cette deuxième vue correspondent respectivement aux *Activity* et *Next Activity* dans la première vue. *ActivityInPut.*
Concept et *ActivityInPut.Val* présentent les valeurs des propriétés (attributs) des instances du concept *ActivityInPut.*
La même chose est appliquée pour les instances du concept *ActivityOutPut.* En ce qui concerne *Constraint*, ceci est généré à partir de la *Transition* qui correspond au concept *ActivityInPutOutPut*.
ConcernedEquipmentModel présente le concept "*Equipment Model"* faisant référence au concept *"Physical Equipment"*.

Tableau 5-4 Vue de deuxième niveau du processus d'apprentissage

ActivityInPut.Concept	ActivityInPut.Val	Step	ActivityOutPut.Concept	ActivityOutPut.Val	Constraint	NextStep	Concerned Equipment Model
Alarm.Measure.Value	46	Validate Alarm	Alarm.validated	True	Alarm.Measure.Value>42	Set Emergency Level	Pusher
Alarm.validated	True	Set Emergency Level	Alarm.Emergency	Very High	Alarm.Measure.Value>42 And Alarm.Validate=True	Request exepertise	Pusher
Alarm	44	Validate	Alarm.validated	False	Alarm.Value>4	Close	Puller

		Alarm			2	Work request	
Measure	41	Control Data	Condition	Not Violated	Measure.Value> 42	Control Data	Indexer
Measure	42	Control Data	Condition	Not Violated	Measure.Value> 42	Control Data	Indexer
Measure	43	Control Data	Condition	Violated	Measure.Value> 42	Create Alarm	Indexer

4.2- Apprentissage

4.2.1- Construction de l'ensemble d'apprentissage

A partir de la deuxième vue, nous construisons l'ensemble d'apprentissage de C4.5. Cet ensemble est constitué par une collection de séquences de données. Chaque séquence a un *tuple* de valeurs pour un ensemble fixe d'attributs (ou variables indépendantes) A = {A1, A2,… An} et un attribut de classe (ou variable dépendante). Un attribut Aa est décrit comme continu ou discret selon ses valeurs qui sont numériques ou nominales (Kohavi & Quinlan, 2002).

Dans notre étude de cas nous avons pris comme variable à expliquer (variable de classe) l'attribut ActivityOutPut.Val et comme ensemble de variables indépendantes (variables explicatives) *{ActivityInPut.Concept, ActivityInPut.Value, ActivityOutPut.Concept, constraint, ConcernedEquipmentModel}.*

La Figure 5-10 montre un exemple de l'ensemble d'apprentissage soumis à C4.5 avec un validateAlarm.names définissant les attributs et un fichier validateAlarm.data recensant les instances.

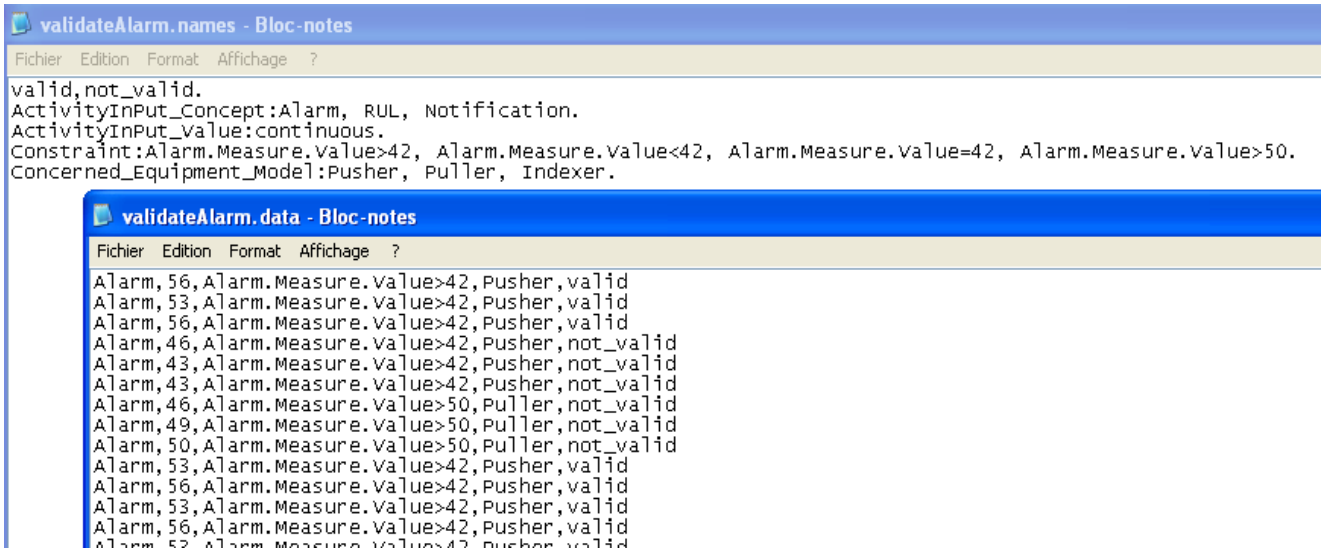


Figure 5-10 Exemple de fichiers C4.5 pour le STEP « validate.Alarm »

4.2.2- Lancement de C.4.5 et validation

Le lancement automatique de l'algorithme C4.5 peut se faire soit périodiquement (par exemple tous les mois), ou en fonction d'un seuil identifié sur le nombre d'instances dans la base de connaissances (par exemple l'algorithme sera lancé tous les 100 nouveaux instances d'un concept). Notre choix s'est porté sur la deuxième alternative car le nombre d'instances ne varie pas de façon uniforme dans le temps. En effet, nous pouvons avoir par exemple seulement 9 instances pendant un mois et 100 instances en deux jours. Ce nombre est vérifié par le composant raisonneur à chaque fois qu'une nouvelle instance est ajoutée à la base de connaissances. La Figure 5.11 montre un exemple d'arbre de décision obtenu suite à l'exécution de C4.5.

La validation des résultats d'apprentissage se fera suivant quatre indicateurs à savoir : les instances correctement classés (*Correctly Classified Instances), Kappa statistic*, l'erreur relative absolue (*Relative absolute error)* et l'erreur relative quadratique (*relative squared error*). Kappa *statistic* est une mesure de chance corrigée d'un accord entre les classes estimées et les classes vraies. Il est calculé en prenant l'accord attendu par chance loin de l'accord observé et en divisant par le maximum d'accord possible. Les taux d'erreur sont utilisés pour la prévision numérique plutôt que la classification. En prévision numérique, les prévisions ne sont pas seulement bonnes ou mauvaises, l'erreur a une grandeur, et ces mesures le reflètent (Bouckaert, 2010).

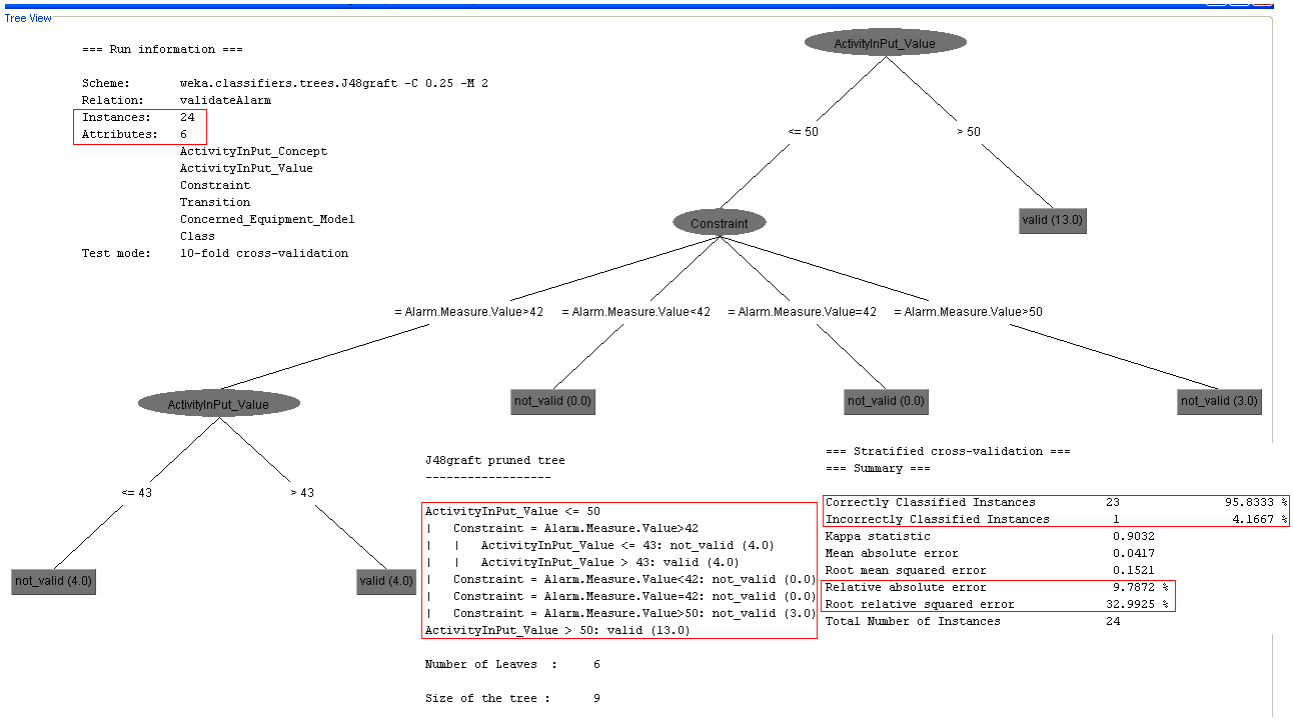


Figure 5-11 Résultats de la deuxième exécution de l'algorithme d'apprentissage

Les décisions se font automatiquement, nous avons adopté une politique de validation en définissant des seuils de confiance élevés comme configuration de base (Tableau 5), mais qui peuvent être modifiés par l'administrateur de la plateforme. En outre, nous notons que tous les seuils doivent être respectés, pour valider les résultats d'apprentissage.

Tableau 5-5 Seuils de validation des indicateurs

Indicator	Threshold
Correctly Classified Instances	> = 90%
Kappa statistic	> = 0.5
Relative absolute error	< = 10%
relative squared error	< 50%

4.3- L'adaptation des règles et mise à jour de la base de connaissances

L'adaptation, présente la troisième activité dans le processus d'apprentissage. L'objectif de cette activité est d'adapter les règles génèrées de l'activité d'apprentissage et de les enregistre comme de nouvelles instances dans la base de connaissances. Le fonctionnement de cette activité suit le processus présenté dans la Figure 5.12.

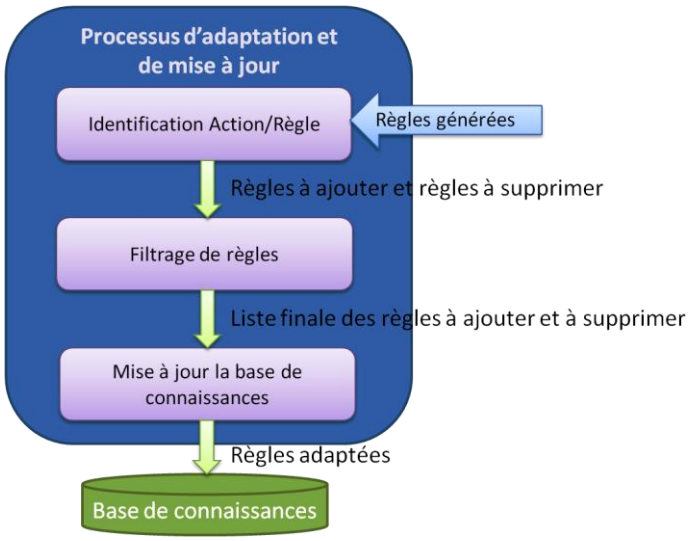


Figure 5-12 Processus d'adaptation et de mise à jour

La première activité dans ce processus porte sur l'identification des actions à faire par rapport à chaque règle générée (règles à ignorer, règles à supprimer, règles à ajouter). La deuxième activité porte sur le filtrage des règles à ajouter afin d'identifier les possibilités de subsomption. Après ceci, la dernière activité consiste à mettre à jour la base de connaissance par l'ajout des nouvelles règles, la suppression ou la modification des règles déjà existantes. En plus, nous rappelons que cette mise à jour concerne principalement les instances des concepts « Step», « Transition » et « Constraint ». La modification au niveau de cette vue de l'ontologie provoque un changement dans les modèles de processus (process pattern).

4.3.1- Identification actions par rapport aux règles générées

La Figure 5-13 fournit un exemple de résultat d'apprentissage des règles. Le *raisonneur* doit traduire ces règles sous une forme respectant la vue du processus de l'ontologie IMAMO. Le résultat d'apprentissage est composé par des règles définies et exprimées par les attributs des fichiers C4.5. Par conséquent,

le *raisonneur* prend avantage du fait que la structure de tous les fichiers de données est la même pour toutes les instances du concept *Step* et que les attributs présentent les concepts de l'ontologie. Ainsi, les règles qui en résultent sont exprimées en utilisant les concepts de l'ontologie et de leurs instances ou des termes connexes. L'exemple suivant fournit des règles sur les transitions possibles et leurs contraintes relatives, liées à «Valider alarme » une instance de concept *Step*.

```
J48graft pruned tree
------------------

ActivityInPut_Value <= 50
|   Constraint = Alarm.Measure.Value>42
|   |   ActivityInPut_Value <= 43: not_valid (16.0)
|   |   ActivityInPut_Value > 43: valid (4.0)
|   Constraint = Alarm.Measure.Value<42: not_valid (0.0)
|   Constraint = Alarm.Measure.Value=42: not_valid (0.0)
|   Constraint = Alarm.Measure.Value>50: not_valid (14.0)
ActivityInPut_Value > 50: valid (48.0)

Number of Leaves  :     6

Size of the tree :      9
```

Figure 5-13 Exemple de règles résultantes de l'apprentissage

Le résultat obtenu doit être analysé en utilisant la vue du second niveau liée à l'étape d'analyse dans l'activité d'apprentissage. Cette vue permet de comprendre le niveau de la relation entre les attributs utilisés dans les règles et les concepts. Par exemple, concernant la vue de second niveau, dans le cas du *Step «Valider alarme »*, *ActivityInPutValue* présente la valeur de *Alarm.Measure.Value*.

Afin d'interpréter les règles d'apprentissage générées, le *raisonneur* fait un *mapping* entre ces règles, la vue de deuxième niveau et la base de connaissance. Il construit un tableau d'interprétation ayant comme schéma *le 4-uplet (Step, contrainte classifié, Transition associée apprise, Contrainte existante).* Le tableau comportera donc, le *Step* sur lequel l'apprentissage était effectué, la contrainte classifiée associée à cet apprentissage, la valeur de la transition résultante de cette contrainte et finalement la contrainte déjà associée au *Step* dans la base de connaissance. La structure de ce tableau permettra de comparer les contraintes existantes et les contraintes classifiées par l'apprentissage, et permettra aussi la création dans la base de connaissances des relations entre ces nouvelles règles et les valeurs des transitions associées. Le tableau 6 présente le tableau d'interprétation du résultat d'apprentissage au sujet du *Step «Valider alarme »*.

Tableau 5-6 Table d'interprétation

Step	Extracted Constraint	Learned Transition	Existed constraint
Validate-Alarm	Alarm.Measure.Value >50	Alarm-Valid	Alarm.Measure.Value >50
Validate-Alarm	Alarm.Measure.Value <=43	Alarm-Not-Valid	Alarm.Measure.Value >42
Validate- Alarm	Alarm.Measure.Value >43	Alarm-Valid	Alarm.Measure.Value >42
Validate-Alarm	Null	Null	Alarm.Measure.Value <42
Validate-Alarm	Null	Null	Alarm.Measure.Value =42
Validate-Alarm	Alarm.Measure.Value <=50	Alarm-Not-Valid	Alarm.Measure.Value >50

A partir de ce tableau, grâce à la comparaison des règles apprises et des règles existantes, le *raisonneur* identifie les actions à faire par rapport à chaque règle apprise, soit une action d'ignorance ou de modification.

Par exemple dans les règles comparées dans le tableau 6, nous pouvons identifier les actions suivantes :

- *Ignorer les règles* contradictoires
    - La contrainte *Alarm.Measure.Value <=50* extraite lors de l'apprentissage est contradictoire à la contrainte existante Alarm.Measure.Value >50.
- *Modifier les règles existantes*:
    - Alarm.Measure.Value >42, Alarm.Measure.Value <42, Alarm.Measure.Value =42 seront remplacées par les règles Alarm.Measure.Value <=43 and Alarm.Measure.Value >43.

A partir de ces actions identifiées et les valeurs des transitions apprises, le raisonneur définit les règles et les relations à mettre en place dans la base de connaissances. A partir des actions identifiées pour notre cas d'étude (tableau 6), deux actions seront définit :

- Eliminer les relations avec les contraintes existantes à modifier (entre *Transition et Constraint*)
    - Alarm.Measure.Value >42
    - Alarm.Measure.Value <42
    - Alarm.Measure.Value =42
- Ajouter les nouvelles relations pour les règles apprises
    - Règle 1: Alarm.Measure.Value >43 (autogéré) (has Transition ← Valid_Alarm)
    - Règle 2: Si Alarm.Measure.Value <=43 (autogéré) (has Transition ← Not_Valid_Alarm)
    - Règle 3: Si Alarm.Measure.Value >50 (autogéré) (has Transition ← Valid_Alarm)

4.3.2- Filtrage des règles

Après ce *mapping* règles/actions, une activité de filtrage des règles à ajouter est lancée. Cette étape consiste à vérifier s'il y a des règles contradictoires ou s'il y a des règles plus générales que d'autres entre ces nouvelles règles (vérifier les possibilités de subsomption). Par exemple, la règle 1 est plus générique que la règle 3, d'où la règle la plus générale sera adoptée. Par conséquent, à la fin de cette activité deux règles sont validées à savoir règle 1 et la règle 2 ainsi que les règles qui seront modifiées.

4.3.3- Mise à jour de la base de connaissances

Comme défini dans l'ontologie, nous dissocions les modèles abstraits des modèles physiques. Nous avons un modèle abstrait relatif aux processus (représenté par les concepts S*tep, Transition et Constraint)* et un modèle opérationnel (représenté par les concepts*Activity, et ActivityInPutOutPut)* instanciés lors de l'exécution de la plateforme. La mise à jour de la base de connaissances concerne les instances du modèle abstrait de processus. Un exemple de mise à jour est illustré dans les Figures 5-14 et 5-15 fournissant un aperçu en amont et en aval de la base de connaissances ainsi que les opérations effectuées lors de sa mise à jour.

```
(DEFRELATION Confirmed-Self-Managed((?C Constraint) (?Self-Managed Boolean)))
(Assert (Step Validate-Alarm))
(Assert (Transition Alarm-Not-Valid))
(Assert (Transition Alarm-Valid))
(Assert (Constraint Alarm.Measure.Value>42))
(Assert (Constraint Alarm.Measure.Value<42))
(Assert (Constraint Alarm.Measure.Value=42))
(Assert (has-constraint Alarm-Not-Valid  Alarm.Measure.Value<42))
(Assert (has-constraint Alarm-Valid  Alarm.Measure.Value>42))
(Assert (has-constraint Alarm-Not-Valid  Alarm.Measure.Value=42))
(Assert (Confirmed -Self-Managed  Alarm.Measure.Value=42 False))
(Assert (Confirmed -Self-Managed  Alarm.Measure.Value<42 False))
(Assert (Confirmed -Self-Managed  Alarm.Measure.Value>42 False))
```

Figure 5-14 Base de connaissances initiale

Selon les règles apprises, le *raisonneur* commence par rétracter les instances à supprimer, puis ajouter les nouvelles instances. La base de connaissances mise à jour dans la Figure 5.19 montre les relations rétractées entre les transitions `Alarm-Valid` et `Alarm-Not-Valid` avec certaines contraintes, l'instanciation de nouvelles contraintes et de leurs relations avec les transitions `Alarm-Valid` et `Alarm-Not-Valid` et enfin l'affectation de la valeur « `True` » à l'attribut `Confirmed-Self-Managed` de ces contraintes. La Figure 5-16 présente la nouvelle base de connaissances résultante après la mise à jour.

Modifier régles existantes

```
(Retract (has-constraint Alarm-Valid  Alarm.Measure.Value>42))
(Retract (has-constraint Alarm-Not-Valid  Alarm.Measure.Value<42))
(Retract (has-constraint Alarm-Not-Valid  Alarm.Measure.Value=42))
(Assert (Constraint Alarm.Measure.Value>43))
(Assert (Constraint Alarm.Measure.Value<=43))
```

Ajouter nouvelles régles

```
(Assert (has-constraint Alarm-Valid  Alarm.Measure.Value>43))
(Assert (has-constraint Alarm-Not-Valid  Alarm.Measure.Value<=43))
(Assert (Confirmed-Self-Managed  Alarm.Measure.Value<=43 True))
(Assert (Confirmed-Self-Managed  Alarm.Measure.Value>43 True))
```

Figure 5-15 Base de connaissances mise à jour par le *GeP*

Base de connaissance après mise à jour :

```
(Assert (Step Validate-Alarm))
(Assert (Transition Alarm-Not-Valid))
(Assert (Transition Alarm-Valid))
(Assert (Transition Alarm.Measure))
(Assert (Constraint Alarm.Measure.Value>43))
(Assert (Constraint Alarm.Measure.Vale<=43))
(Assert (has-constraint Alarm-Valid  Alarm.Measure.Value>43))
(Assert (has-constraint Alarm-Not-Valid Alarm.Measure.Value<=43))
(Assert (Confirmed-Self-Managed Alarm.Measure.Value<=43 True))
(Assert (Confirmed -Self-Managed Alarm.Measure.Value>43 True))
```

Figure 5-16 Nouvelle base de connaissances

Ainsi, à l'issue de cette mise à jour de la base de connaissances, le modèle processus est modifié. La nouvelle version du processus de gestion dans la Figure 5-17 est bien évoluée par rapport au premier processus illustré dans la Figure 5-8.

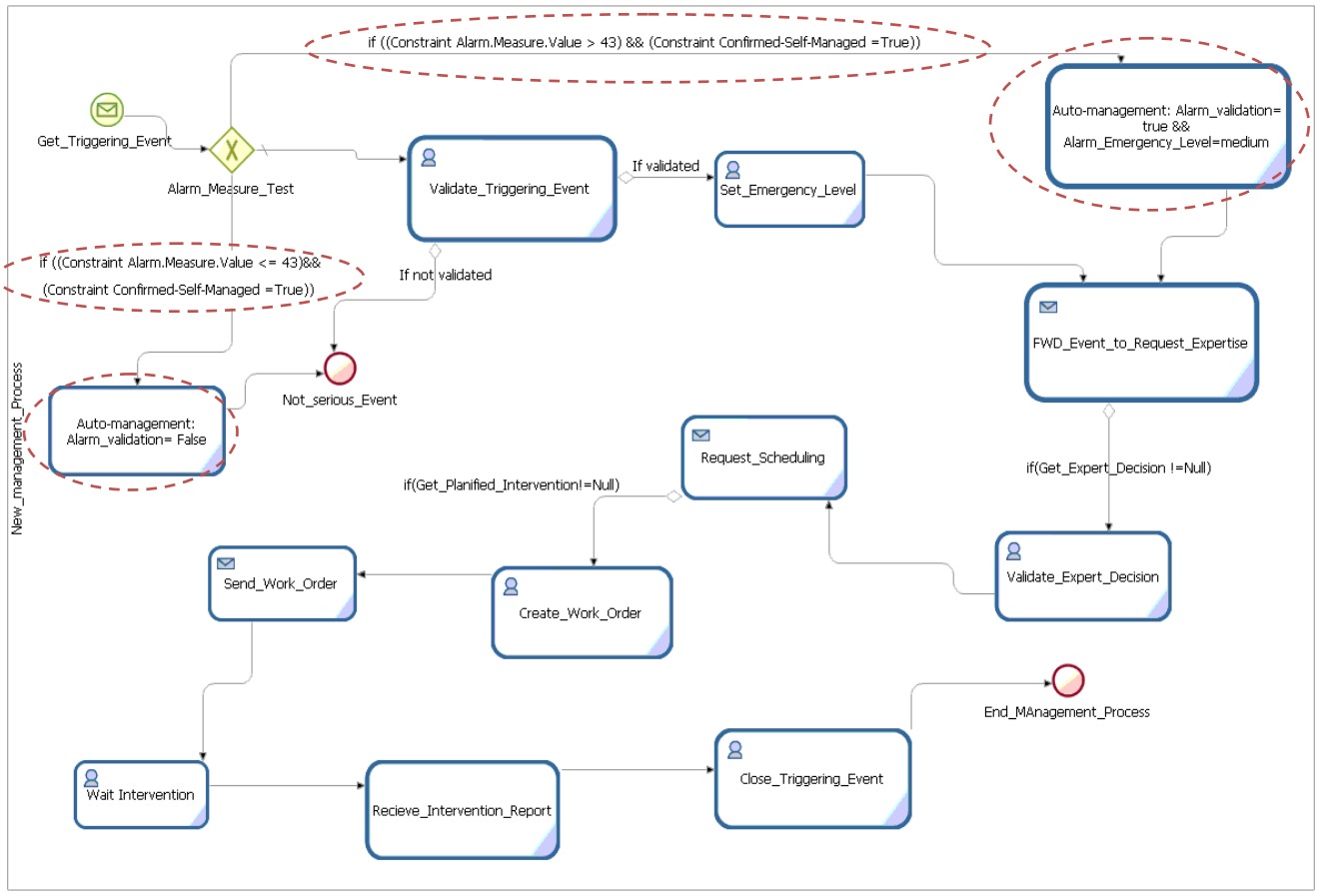


Figure 5-17La nouvelle version du processus de gestion

5. Exploitation des règles issues de l'apprentissage

Dans cette section nous présentons deux cas d'exploitation des règles issues de l'apprentissage. La première porte sur la fonctionnalité d'autogestion et la deuxième sur un service de diagnostic dynamique. L'exploitation de ces règles ajoutées dans la base de connaissances peut se faire de deux manières différentes, soit par autogestion des processus dans la plateforme, soit par les services fournis par les applications qu'elle intègre. La différence entre un service dynamique et la fonctionnalité d'autogestion est que cette dernière porte sur l'autogestion (exploitation des règles d'apprentissage) de l'activité effectuée par l'utilisateur humain et que le premier porte sur l'exploitation du service lui-même des règles d'apprentissage sur les activités effectuées par les applications.

5.1- Fonctionnalité d'autogestion

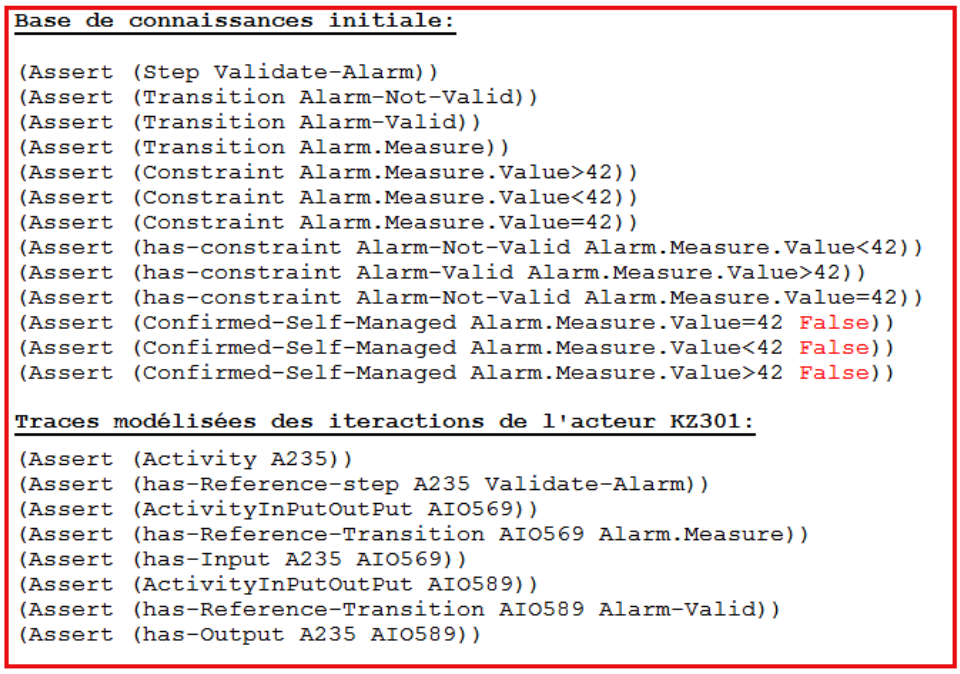
La fonctionnalité d'autogestion dans la plateforme de s-maintenance est assurée par le composant *GeP (gestionnaire de processus).* Elle permet d'exploiter les connaissances dynamiques apprises pour gérer de façon automatique les activités sur les quelles la fonctionnalité d'auto-apprentissage a été mise. L'exécution de ces activités en prenant appui sur ces règles de connaissances se fera sans intervention d'opérateurs.

En effet, la Figure 5-14 contient des éléments de la base de connaissances, *Confirmed-Self-Managed* est un attribut 46 booléen du concept «Constraint ». Sachant qu'une règle apprise porte sur une contrainte de transition entre activité, cet attribut est mis à «true» lors de la mise à jour de la base de connaissances.

La Figure 5-18 montre une vue de trace d'exécution extraite de la base de connaissance de la plateforme concernant quelques activités avant la mise à jour de la base de connaissance suite à l'auto-apprentissage sur ces activités. Dans cette phase, l'exécution et l'instanciation des ces activités sont assurées par des acteurs de maintenance utilisant la plateforme.

Après un auto-apprentissage validé sur une instance particulière du concept Step, les activités faisant références à ce Step (instances du concept Activity), ainsi que les entrées et sorties de ces activités (les instances du concept AcitivityInPutOutPut) seront autogérées (c.à.d. lancement automatiquement de l'exécution et affectation automatique des valeurs) par le composant *GeP*.

Dans la Figure 5-19, nous fournissons une vue extraite de la base de connaissances de la plateforme sur des activités et des transitions gérées automatiquement par le *GeP* avec une affectation automatique des valeurs de sorties de l'activité grâce aux règles apprises. L'exemple dans cette vue concerne une activité faisant référence au *Step « Validate-Alarm »* dans le processus de management illustré dans la Figure 5-17.



```
Base de connaissances initiale:

(Assert (Step Validate-Alarm))
(Assert (Transition Alarm-Not-Valid))
(Assert (Transition Alarm-Valid))
(Assert (Transition Alarm.Measure))
(Assert (Constraint Alarm.Measure.Value>42))
(Assert (Constraint Alarm.Measure.Value<42))
(Assert (Constraint Alarm.Measure.Value=42))
(Assert (has-constraint Alarm-Not-Valid Alarm.Measure.Value<42))
(Assert (has-constraint Alarm-Valid Alarm.Measure.Value>42))
(Assert (has-constraint Alarm-Not-Valid Alarm.Measure.Value=42))
(Assert (Confirmed-Self-Managed Alarm.Measure.Value=42 False))
(Assert (Confirmed-Self-Managed Alarm.Measure.Value<42 False))
(Assert (Confirmed-Self-Managed Alarm.Measure.Value>42 False))

Traces modélisées des iteractions de l'acteur KZ301:

(Assert (Activity A235))
(Assert (has-Reference-step A235 Validate-Alarm))
(Assert (ActivityInPutOutPut AIO569))
(Assert (has-Reference-Transition AIO569 Alarm.Measure))
(Assert (has-Input A235 AIO569))
(Assert (ActivityInPutOutPut AIO589))
(Assert (has-Reference-Transition AIO589 Alarm-Valid))
(Assert (has-Output A235 AIO589))
```

Figure 5-18 Une vue de la base de connaissances avant l'auto-apprentissage

[46] Nous rappelons que dans PowerLoom, les attributs des concepts sont présentés comme des relations entre le concept et le nom de l'attribut et son type.

```
Base de connaissance après mise à jour :

(Assert (Step Validate-Alarm))
(Assert (Transition Alarm-Not-Valid))
(Assert (Transition Alarm-Valid))
(Assert (Transition Alarm.Measure))
(Assert (Constraint Alarm.Measure.Value>43))
(Assert (Constraint Alarm.Measure.Vale<=43))
(Assert (has-constraint Alarm-Valid  Alarm.Measure.Value>43))
(Assert (has-constraint Alarm-Not-Valid Alarm.Measure.Value<=43))
(Assert (Confirmed-Self-Managed Alarm.Measure.Value<=43 True))
(Assert (Confirmed -Self-Managed Alarm.Measure.Value>43 True))

Traces modélisées des interactions du GEP :
(Assert (Activity A425))
(Assert (has-Reference-step A425 Validate-Alarm))
(Assert (ActivityInPutOutPut AIO692))
(Assert (has-Reference-Transition AIO692 Alarm.Measure))
(Assert (has-Input A425 AIO692))
(Assert (ActivityInPutOutPut AIO891))
(Assert (has-Reference-Transition AI891 Alarm-Valid))
(Assert (has-Output A425 AIO891))
```

Figure 5-19 Vue de la base de connaissances après auto-apprentissage et avec autogestion

5.2- Service dynamique

Nous illustrons sur les Figures suivantes le cas du service de diagnostic qui évolue dynamiquement avec la dynamicité de la base de connaissance à chaque fois une nouvelle règle apprise concerne l'une des activités gérées par lui est ajoutée dans la base. La Figure 5-20 illustre le processus de diagnostic initial définit dans la base de connaissance.

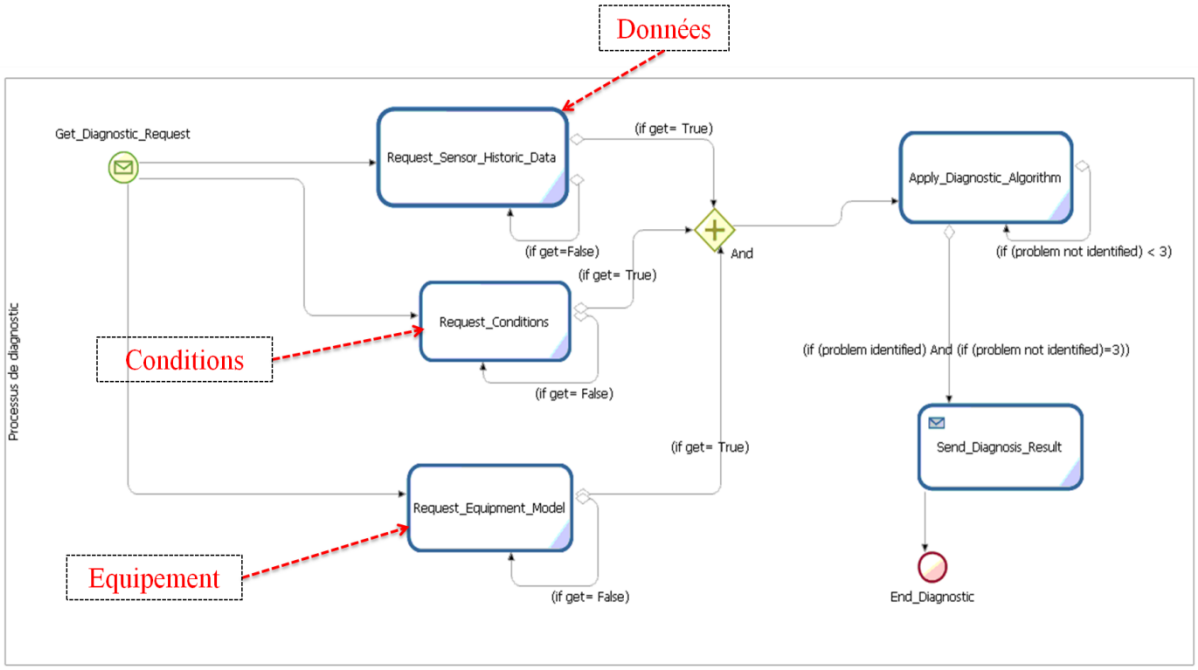


Figure 5-20 Processus initial du service de diagnostic

Ainsi, après un auto-apprentissage sur l'étape (activité modèle) *« Apply_Diagnostic_Algorithm »*, une nouvelle règle sur les entrées (Données, Condition, Modèle équipement) et la sortie (problème identifié) est rajoutée dans la base de connaissances.

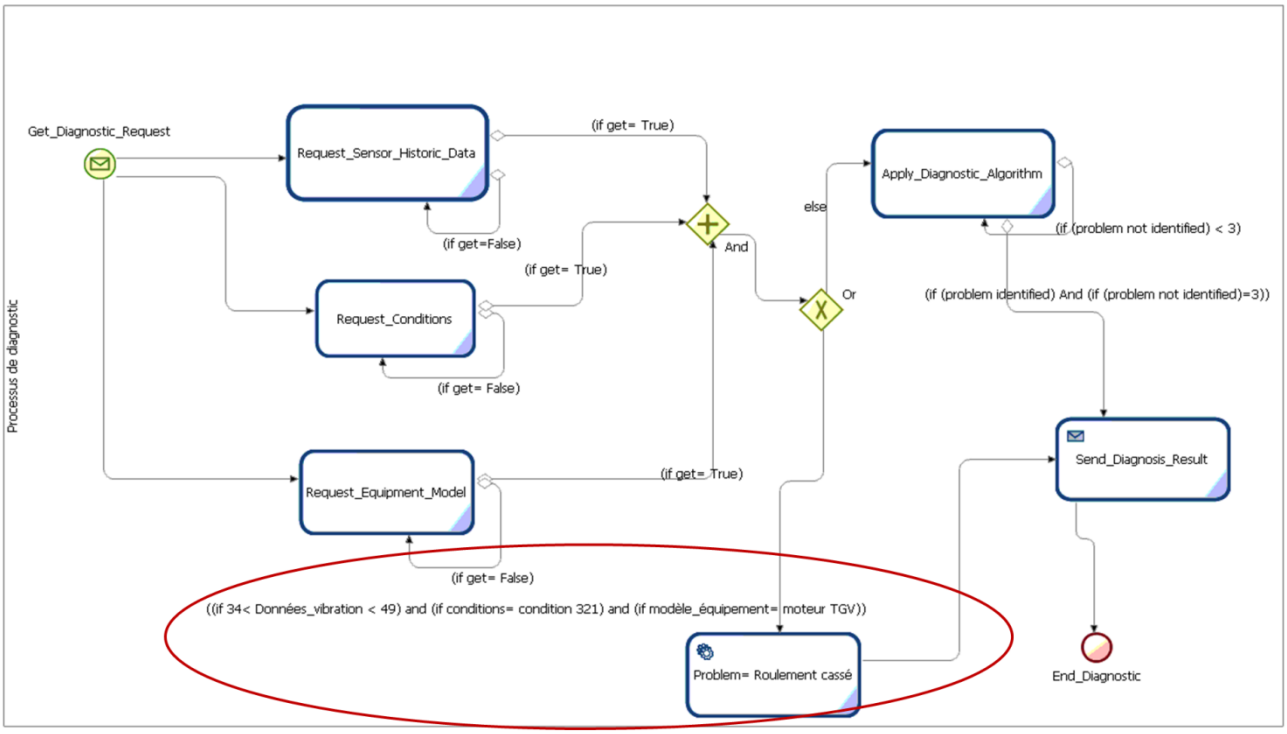


Figure 5-21 Nouveau processus du service de diagnostic après un premier auto-apprentissage

Par exemple, l'apprentissage sur les entrées sorties du *step « Apply_Diagnostic_Algorithm »* a permis de déduire que pour les moteurs de TGV (modèle d'équipement) , quand les données du capteur de vibration sont entre 34 et 49 et que la condition associée est la condition 321 donc le problème identifié est « roulement cassé ». Cette nouvelle règle est ajoutée donc à la base de connaissances. Lors des nouvelles exécutions du service de diagnostic, ce dernier suit le nouveau processus issu de cet apprentissage (voir Figure 5-21).

Ainsi, à chaque fois que de nouvelles règles sont apprises, le service de diagnostic suit le nouveau processus résultant de ce dernier apprentissage. La dyanmicité du service est toujours en évolution avec l'avancement de l'apprentissage et de l'exécution.

La Figure 5.22 fournit une autre version du processus du service de diagnostic suite à une nouvelle règle d'apprentissage portant sur les actionneurs. En effet, pour les équipements ayant comme modèle d'équipement actionneur, si les données du capteur de fin de course sont égales à faux et si la condition associée est la condition 2574 alors le problème identifié est « vieillissement des joints des vannes ».

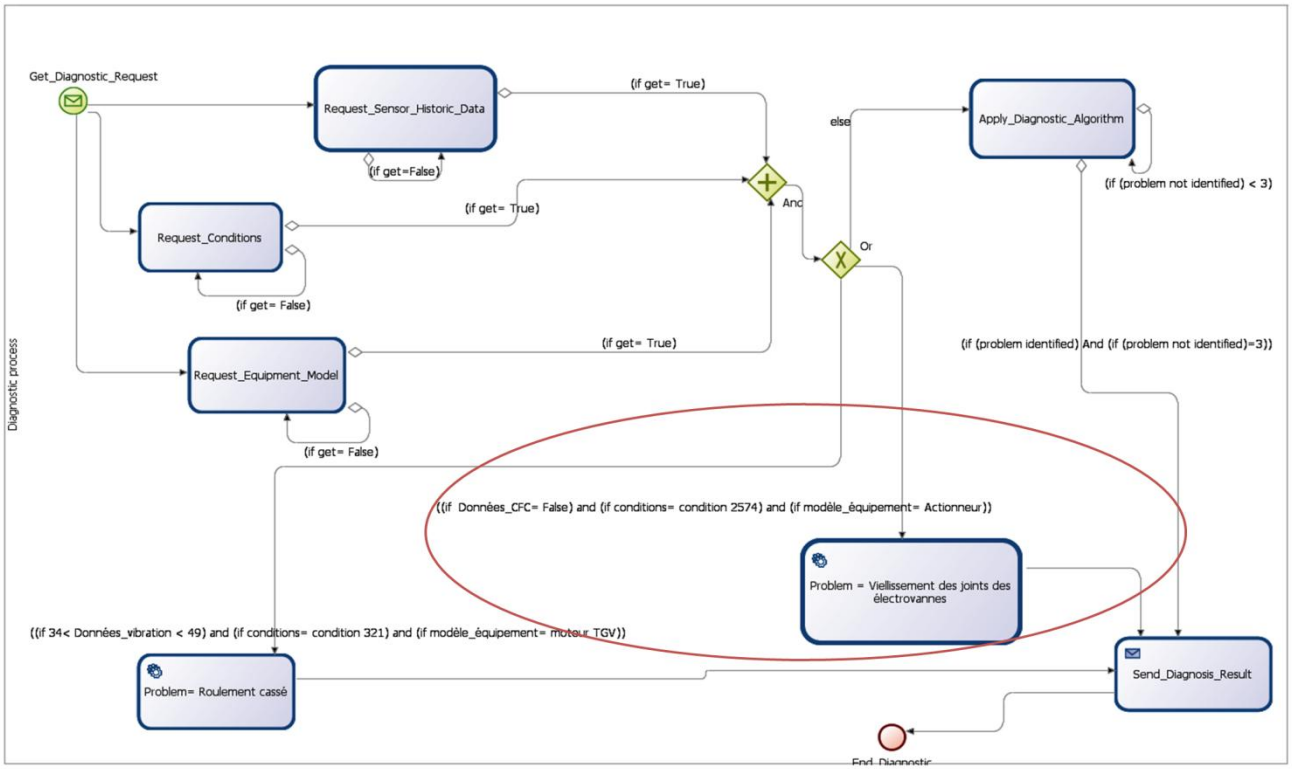


Figure 5-22 Nouveau processus du service de diagnostic issu d'un nouvel auto-apprentissage

Après avoir appliqué ces fonctionnalités sur un cas d'étude, et montrer la faisabilité de ce système à base de traces, nous étudierons l'impact de celles-ci sur les performances de la fonction de maintenance.

6. Impact de ces fonctionnalités sur les performances de la maintenance

El Aoufir et al, définissent l'efficience de la maintenance par son aptitude à améliorer les objectifs fixés (disponibilité, sécurité, qualité, …) tout en réduisant le coût global de la maintenance. Ils affirment que la mesure de la performance du processus maintenance est liée à la définition et à la mesure d'indicateurs de performance techniques (exemple : la disponibilité) et d'indicateurs économiques (les coûts directs et indirects de la maintenance) (EL AOUFIR & BOUAMI, 2005).

L'objectif principal des systèmes de maintenance est de faire évoluer cette efficience en améliorant la mesure de performance du processus de maintenance.

Dans ce contexte, nous allons faire une étude sur l'impact des fonctionnalités de la plateforme de s-maintenance sur les performances techniques et économiques de la maintenance. Cette étude portera sur deux types d'indicateurs de temps et de coûts.

6.1- Performance technique : indicateurs de temps

Nous nous intéresserons à l'indicateur de temps global du processus MTTR permettant de calculer le temps global du processus de maintenance (Stanley, 2011). Il existe d'autres indicateurs, comme le MTBF[47] facilement calculable dans la plateforme de maintenance, mais que nous n'utiliserons pas dans notre cas de simulation de fonctionnement de la plateforme.

Le MTTR *(mean time to repair)* est appelé aussi indice de maintenabilité. La maintenabilité s'entend, pour une entité utilisée dans des conditions données, comme la probabilité pour qu'une activité donnée de maintenance puisse être effectuée sur un intervalle de temps donné, lorsque la maintenance est assurée dans des conditions données et avec l'utilisation de procédures et moyens prescrits (Stanley, 2011).

Le MTTR est calculé en additionnant les temps actifs de maintenance (comportant les temps de localisation de défaillance, de diagnostic, d'intervention, de contrôles et d'essais) ainsi que les temps annexes de maintenance (comportent les temps de détection, d'appels à la maintenance, d'arrivée de la maintenance, de préparation de la logistique d'intervention, etc.), le tout divisé par le nombre d'interventions.

De plus, il est à noter que les indicateurs de maintenance se basent généralement sur les données réelles et non pas sur des estimations. La plateforme permet d'effectuer des calculs réels ainsi que des calculs d'indicateurs estimés (par exemple grâce à l'attribut « Estimated-Execution-Duration » du concept « Step ») ce qui permettra de comparer ces différentes valeurs entre elles.

Par conséquent, le temps estimé d'exécution d'un processus est égal au temps d'exécution estimé d'un « process pattern » que nous notons TexPp (Temps exécution Process pattern).

$$TexPp(pp) = \sum (pp.Act_i.estimated\_excution\_duration))$$

D'autre part, l'exécution d'un processus de maintenance dure un certain temps, avec un début (period.start) et une fin (period.end). Un processus peut être démarré par un utilisateur ou bien par un autre processus. Ce temps d'exécution représente la somme des temps d'exécutions réels des activités (calculer grâce au concept « period » et ses attributs (start et end)) appartenant à chaque processus. On le note TexP (Temps exécution processus) :

$$TexP(p) = (p.Period.End - p.Period.Start)$$
$$= \sum (TexA(\ p.Act_i\ )) = \sum (p.Act_i.Period.End - p.Act_i.Period.Start)$$

Ainsi, pour calculer les temps estimés et réels de maintenance sur une période donnée, il suffit de multiplier les deux indicateurs précédents par le nombre de processus instanciés tout au long de cette période.

Par conséquent, le temps d'exécution estimé (Tee) pour une période A donnée se calcule par :

$$Tee\ (A) = K * TexPp(MT)$$

[47] Le MTBF (Mean Time Between Failures), appelé aussi indice de fiabilité, désigne le temps moyen entre défaillances consécutives d'un type d'équipement.

K présente le nombre de processus instanciés et faisant référence aux instances du concept« Maintenance Type » (MT) qui est un sous concept de « process pattern »[48].

Ainsi le Temps d'exécution réel (Ter) est noté par:

$$Ter\ (A) = H * TexP(p)$$

H est le nombre de processus instanciés du processus (p) instanciés et faisant référence aux instances du concept« Maintenance Type » durant la période l'année A.

Les deux temps calculés précédemment permettent d'évaluer l'erreur entre le temps estimé et le temps réel pour une période donnée. En fonction de ce taux d'erreur, nous pouvons remettre en cause le calcul d'estimation de temps. Cet indicateur que nous appelons erreur d'estimation de temps d'exécution (EeTe) est noté par:

$$EeTe = Ter - Tee$$

6.2- Performance économique : indicateurs de coût

En ce qui concerne les performances économiques, elles sont liées directement aux coûts de la maintenance. La norme NF X60-0201 définit les coûts de maintenance comme étant *les coûts directement imputables à la maintenance*. Elle précise que *ces coûts peuvent s'analyser par nature (personnel, outillages et équipements de maintenance, produits et matières consommées, sous-traitance, autres) et par destination (préparation, documents techniques, interventions, suivi et gestion, magasinage, formation, autres, ...).* Ces coûts sont généralement appelés coûts directs.

La Figure 5-23 montre qu'il existe des coûts indirects en opposition avec les coûts directs et qui se répercutent sur les coûts cumulés de maintenance. Ces coûts indirects sont composés de :

- coûts d'indisponibilité qui incluent les coûts de pertes de production, la non qualité, les surcoûts de production, les pénalités contractuelles, … consécutifs à une défaillance.
- coûts de défaillance qui intègrent les coûts de maintenance corrective et les coûts d'indisponibilité consécutifs à la défaillance.

En effet, les coûts directs d'un processus dépendent des coûts des activités appartenant à celui-ci et les coûts indirects dépendent directement de son temps d'exécution.

[48] Remarque : Les processus complexes sont composés par des activités qui peuvent faire références à des « steps » qui font références à d'autres processus (exemple un processus de maintenance conditionnelle fait appel à un processus de diagnostique et un processus d'intervention. Par conséquent, pour éviter toutes erreurs de calcule, nous considérons pour tous les calcules dans cette section que les processus faisant références à des « process pattern » de type « maintenance type ».

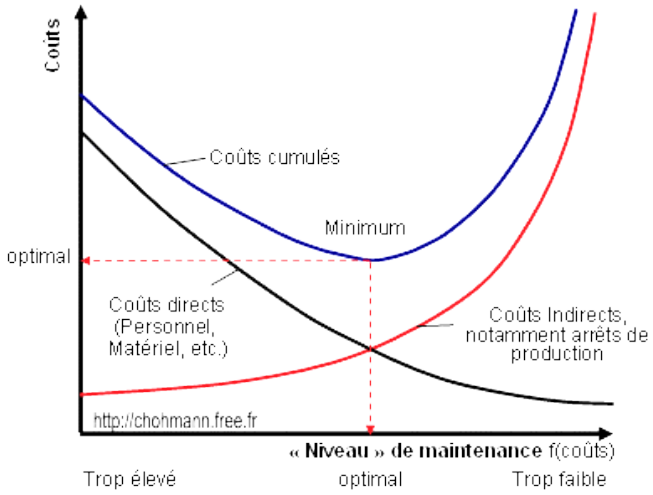


Figure 5-23 Coûts de la maintenance[49]

Lorsque la plateforme a appris une règle permettant de résoudre automatiquement un problème aucun coût ne sera occasionné pour cette activité, au contraire d'une activité impliquant un utilisateur. Nous partons de l'hypothèse suivante pour estimer les coûts directs et indirects d'une activité. Nous classons donc ces activités en deux catégories :

- Activité à coût : une activité ayant une sortie « AcitivityInputOutPut » faisant référence à une transition ayant « confirmed-self-managment » égale à « false ». Le coût cumulé estimé de cette activité est calculé grâce aux valeurs des attributs « estimated-execution-duration » et « estimated-coast » du « step » sur lequel l'activité fait référence.
- Activité sans coût : une activité ayant une sortie « AcitivityInputOutPut » faisant référence à une transition ayant « confirmed-self-managment » égale à « true ». le cout cumulé de cette activité sera égal à zéro grâce à l'automatisation de cette activité (vu la non implication d'utilisateur dans son exécution et le temps négligé de cette exécution).

De la même façon des temps d'exécution, nous allons calculer les coûts réels ainsi que les coûts fictifs estimés des processus de maintenance pour les futures prévisions budgétaires.

Par conséquent, le coût cumulé estimé d'un processus de maintenance est :

$$Ce(p) = \text{cout directe } (p) + \text{cout indirect } (p) = \sum (Cact(\ p.Act_i\ )) + \ (k * (TexP(p)))$$

Sachant que Cact ($p.Act_i$) prèsente le coût d'une activité *i* dans un processus *p*. De plus, la constante k représente le coût d'arrêt de production par unité de temps. En d'autres termes, coût de la perte à cause de l'arrêt de la production.

Ainsi, le coût annuel estimé est noté par :

$$CAe(A) = NI(p) * Ce(p)$$

[49] http://chohmann.free.fr/

NI(p) présente le nombre d'instances du processus « p » durant l'année A.

Le cout réel approximatif des processus de maintenance (que les processus faisant référence à des « process pattern » de type « maintenance type ») durant une année CMr (Coût Manitenance Réel) est égal à :

$$CMR(A)= \sum NI_j * Ce(p_j)$$

Dans cette équation, NI_j présente le nombre d'instances du processus p_j durant l'année A.

Ainsi, le gain annuel de maintenance que l'entreprise peut obtenir présente la différence entre le coût estimé et le coût réel durant une année. Ce gain est présenté par l'équation suivante :

$$Gain\ (A)= Ce(A) - CMR(A)$$

Cet indicateur est utile pour évaluer les performances économiques de la maintenance ainsi que pour ajuster les futurs de m dudgets aintenance.

6.3- Simulation par étude de cas

Dans cette section, nous présentons une étude de cas de l'entreprise fictive TEM (Technlogies, Equipmente and Maintenance), relative à la gestion de la maintenance via la plateforme de s-maintenance. Nous évaluons l'impact de cette gestion sur les performances économiques et techniques de cette entreprise.

TEM est spécialisée dans la fabrication de machines destinées au décolletage de pièces à partir de barres ou de lopins. L'entreprise propose aujourd'hui une large palette de produits. Les gammes principales se composent de tours automatiques mono-broches à poupée mobile, des tours multibroches et de tours multibroches à cames. En effet, en complément de ses produits industriels, TEM fournit aussi à ses clients des services de maintenance avec les machines qu'elle vend. Pour assurer ses services de maintenance, TEM utilise la plateforme de s-maintenance Tem@web.

Le nombre de machine géré par la plateforme Tem@web s'élève à plus de 2000 machines dispersées un peu partout dans le monde. Nous étudierons un échantillon de 1000 machines de la même gamme produites par cette entreprise.

Dans ce contexte, nous posons un ensemble d'hypothèses permettant de simuler sur une période donnée, des données reflétant l'activité de la plateforme de s-maintenance Tem@web dans l'entreprise TEM (voir tableau 5-7).

Hypothèses sur le nombre de processus instanciés :

Le nombre de processus instanciés dans la plateforme dépend du nombre de défaillances enregistrées de ces machines qui est en lien direct avec les événements déclencheurs.

Afin de mettre en place des hypothèses sur les nombre de défaillances, nous avons utilisé la courbe en baignoire pour simuler un nombre de défaillances sur chaque période du cycle de fonctionnement de chaque équipement à savoir la jeunesse, la maturité et la vieillesse (voir Figure 5-24).

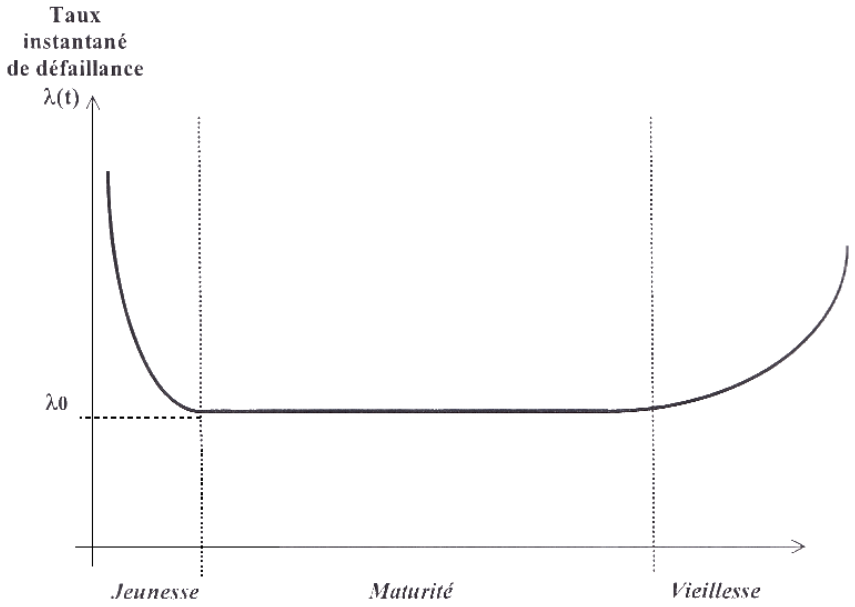


Figure 5-24 Courbe en baignoire

Selon la décomposition de la courbe en baignoire, sur les 1000 machines gérées par la plateforme Tem@web pour une période de 10 ans, nous prenons comme hypothèses : le nombre d'événements déclencheurs par mois s'élève à

- 20% du nombre de machine sur la phase de jeunesse qui dure une année, soit de 2400 par ans.

- 10% du nombre de machine sur la phase de maturité qui dure cinq années, soit de 1200 par ans.

- 35% du nombre de machine sur la phase de vieillesse qui dure quatre années, soit de 4200 par ans.

Chaque événement déclencheur induit l'instanciation d'un processus. Le nombre de processus de maintenance instanciés par an est égal au nombre d'événements instanciés.

Hypothèses sur les processus et leurs coûts

- Il faut mentionner que les processus instanciés ne sont pas tous du même type (ne font pas référence au même « process pattern» et plus précisément au même « maintenance type »). Les coûts de chaque pattern doivent être calculés indépendamment les uns des autres.

- Pour simplifier la simulation, nous allons considérer un coût moyen direct d'un processus (CMdP). Ce coût est calculé par la sommation des coûts estimés de toutes les instances du concept « maintenance type » divisé par le nombre d'instance de ce concept.

$$CMdP = \sum Ce(\ MaintenanceType_i\ ) / Maintenance\_Type.Nombre\_Instances$$

Hypothèses sur les activités :

- Le coût d'un processus dépend des coûts des activités composant ce processus. Afin de simplifier les calculs, nous faisons le calcule du coût moyen des activités dans un processus (CMA).

$$CMA\ (p) = (\sum (Coût(\ p.Actvité_i\ ))) / p.nombre\_Activité$$

- Soulignons que le nombre d'activités dans un processus varie d'un processus à un autre et dépend principalement du nombre de « step » dans un « process pattern ». Pour cela nous considérons une moyenne du nombre d'activités par processus. Cette moyenne consiste à diviser la somme des « step » appartenant à chaque instance de « maintenance type » par le nombre de ces instances.

$$NAP= \sum (nombre\_step.MT_i) / Maintenance\_Type.Nombre\_Instances$$

Hypothèses sur l'apprentissage :

- En ce qui concerne l'apprentissage, nous soulignons que toutes les activités ne peuvent pas être considérées par les services d'auto-apprentissage et d'autogestion. Il existe des activités qui nécessitent des intervenants humains pour des opérations manuelles (par exemple remplacement d'un composant). A cet effet, les activités prises en compte par les deux services sont généralement les activités de prise de décision qui nécessitent des compétences et connaissances de raisonnement.

- D'autre part, nous émettons l'hypothèse que 60% des activités d'un processus est susceptible d'être automatisé.

- De même l'estimation du taux d'apprentissage sur les activités pendant la période de gestion de 10 ans des 1000 machines. Cette période sera décrite à partir de la courbe en baignoire. Le taux d'apprentissage varie d'une phase à l'autre. La Figure 5-25 présente le taux d'apprentissage sur une période de 10 ans, qui varie suivant le nombre de processus instanciés le long de chaque phase. Ainsi, nous pouvons le constater sur la courbe présentée sur la Figure 5-25, l'existence de périodes où l'apprentissage est égal à 0. Ce qui signifie que le processus d'auto-apprentissage pendant cette période n'a inféré aucune règle. Ceci peut être considéré comme une situation qui n'a rien d'inquiétant. En fait, cela se produit quand le système se stabilise et atteint un niveau de maturité suite aux apprentissages effectués pendant les périodes précédentes.

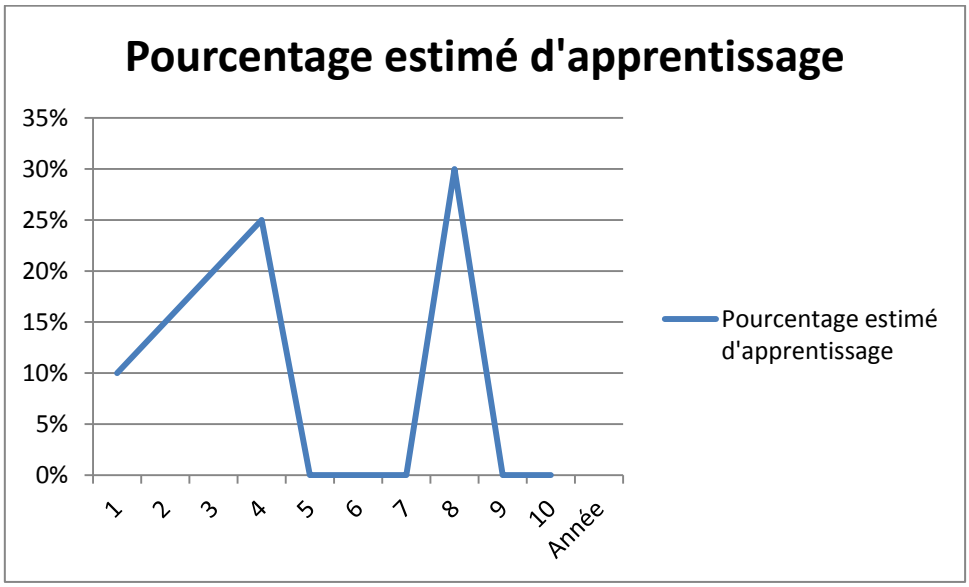


Figure 5-25 Hypothèses d'apprentissage sur une période de 10 ans

Le taux d'apprentissage d'une année se répercutera sur l'année suivante, ce qui explique que le temps réel d'excution lors de la première anné est identique au temps d'exécution estimé.

Simulation des calculs

Avant d'effectuer les calculs de simulation résumés dans le tableau 5-7, nous présentons un ensemble d'hypothèses de départ sur les coûts à utiliser :

- Nombre moyen d'activités dans un processus (NAP) = 10 activités
- Coût moyen d'une activité CMA (p) = 10 Unités de Coût (UC)
- Temps d'exécution estimé d'une activité = 5 Unités de Temps (UT)
- Coût moyen d'un processus = 100 Unités de Coût (UC)
- Temps moyen estimé d'un processus = 50 Unités de Temps (UT)
- Le coût indirect d'un processus = coût d'arrêt de production = (k * (TexP(p), dans notre cas k=1 Unité de Coût).

Il est à noter, que les coûts réels sont calculés à travers les fonctionnalités d'auto-X de la plateforme qui tiennent compte de la présence des règles de décision extraites et enregistrés dans la base de connaissances. Ces règles ont pour objectif la simplification des fontionnalités d'auto-X.

Les coûts estimés ne tiennent pas compte de ces fonctionnalités mais ils sont estimés par rapport à toutes les activités des processus préalablement définis.

Tableau 5-7 Données des performances de maintenance simulées pour TEM sur 10 ans

		1ère année	2ème année	3ème année	4ème année	5ème année	6ème année	7ème année	8ème année	9ème année	10ème année
1	Nombre de processus instanciés	2400	1200	1200	1200	1200	1200	4200	4200	4200	4200
2	Taux d'apprentissage	10%	15%	20%	25%	0	0	0	30%	0	0
3	Nombre moyen d'activité par processus	10	9.4	8.59	7.7	6.8	6.8	6.8	6.8	6	6
4	Temps d'exécution estimé (Tee) (en UT)	120000	60000	60000	60000	60000	60000	210000	210000	210000	210000
5	Temps d'exécution réel (en UT)	120000	56400	51540	46200	40800	40800	142800	142800	126000	126000
6	Gain en temps d'exécution (en UT)	0	3600	8460	13800	19200	19200	67200	67200	84000	84000
7	Coût estimé d'un processus (en UC)	100	100	100	100	100	100	100	100	100	100
8	Coût estimé des processus sans auto-X (en K UC)	360 K	180 K	180 K	180 K	180 K	180 K	630 K	630 K	630 K	630 K
9	cout réel approximatif avec auto-X (en K UC)	360 K	169K	155K	139K	122 K	122 K	429 K	429 K	378 K	378 K
10	Gain/année (en UC)	0	11 K	25 K	41 K	58 K	58 K	201 K	201 K	252 K	252 K
11	Cout cumulé estimé (en K UC)	360 K	540 K	720 K	900 K	1080 K	1260 K	1890 K	2520 K	3150 K	3780 K
12	Cout réel cumulé (en K UC)	360 K	529 K	684 K	822 K	945 K	1067 K	1496 K	1924 K	2302 K	2680 K
13	Gain cumulé (en K UC)	0	11 K	36 K	77 K	135 K	193 K	394 K	595 K	847 K	1099 K

Synthèse et interprétation

Les résultats de cette simulation montrent que le gain pourrait atteindre les 30% des coûts estimés sans la prise en considération des fonctionnalités auto-X de la plateforme. Ce gain possible est nettement identifiable au niveau des coûts de maintenance. Par ailleurs, nous constatons que le gain en disponibilité pourrait aussi être très élevé grâce au gain en temps. Ainsi, nous représentons la variation des temps et des coûts estimés et réels respectivement aux Figures 5-26 (A et B).

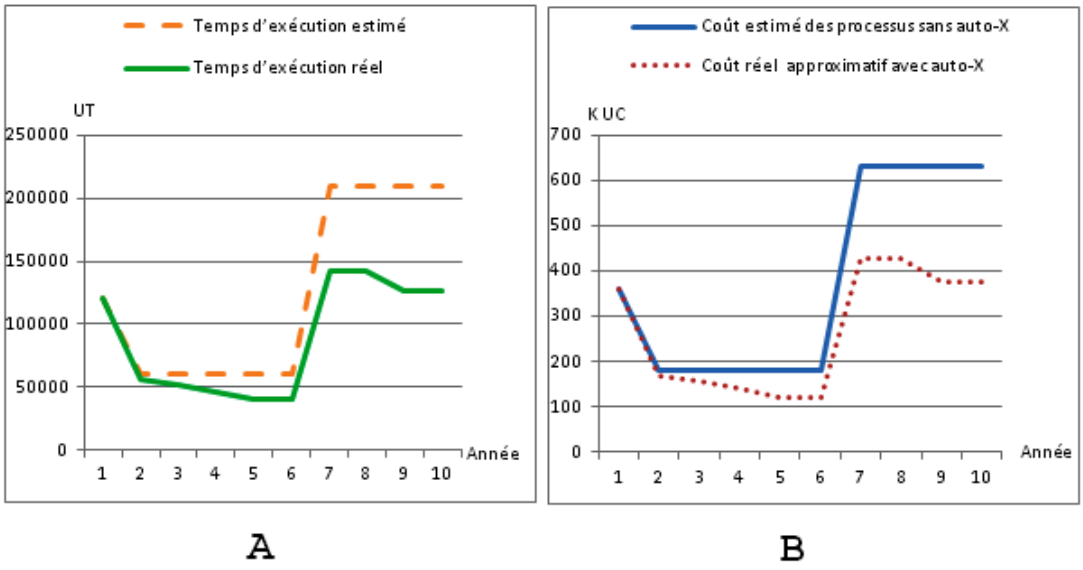


Figure 5-26 Variation des coûts et des temps estimés et réels sur 10 ans

Nous remarquons qu'il y a une baisse importante du coût et du temps à partir de la sixième année. Ce qui signifie que la plateforme de s-maintenance permet de rentabiliser les coûts de maintenance ; c'est-à-dire plus la plateforme apprend plus on obtient une réduction de coûts.

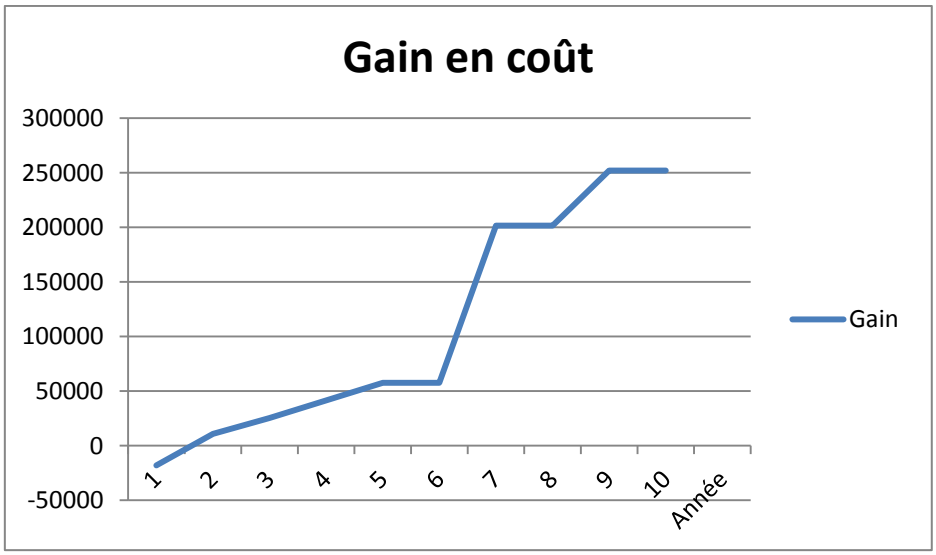


Figure 5-27 Evolution des gains sur 10 ans

En outre, nous remarquons que le gain en temps ou en coût dépend du nombre de processus instanciés, en l'occurrence les phases de jeunesse et de vieillissement des équipements (le nombre de défaillance). De même, le gain (réduction des coûts) évoluerait dans le temps en passant d'une année à une autre (voir Figure 5-27).

D'autre part, dans la pratique, nous devrions tenir compte de la fiabilité du système de surveillance et du dispositif de commutation, ainsi que des applications intégrées (par exemple, diagnostic, pronostic).

7. Conclusion

La dynamique des connaissances des services de la plateforme de s-maintenance prennent appui sur des fonctionnalités d'auto-X (auto-traçabilité, auto-apprentissage et autogestion). La mise en place de ses fonctionnalités s'est faite à l'aide de l'ingénierie des traces qui permet la réutilisation, l'analyse et l'exploitation des interactions faites dans un système.

Partant d'un système à base de traces (SBT) défini pour les environnements d'apprentissages interactifs initialement composé de trois modules interdépendants de base à savoir le système de collecte, le système de transformation et le système de visualisation, nous avons proposé un système à base de traces adapté à la s-maintenance.

Ce SBT prend appui sur l'ontologie du domaine de maintenance (IMAMO) et est composé de deux principaux systèmes gérés par les composants de la plateforme : un système d'auto-traçabilité composé d'un système de collecte et d'un système de transformation ; et un système d'auto-apprentissage.

Le système de collecte qui est géré par le composant coordinateur, enregistre dans des fichiers logs les interactions faites via la plateforme que ce soit entre la plateforme et les utilisateurs ou les interactions entre les composants de la plateforme (activités exécutées).

Le système de transformation géré par le composant *GTM (gestionnaire de traces modélisées)* fait un mapping entre les concepts de l'ontologie (la vue portant sur les processus) et le contenu des fichiers logs des interactions (traces non structurées). Ce mapping a pour objectif d'extraire des traces modélisées afin d'alimenter la base de connaissances et faciliter la tache de leurs interprétation.

Le troisième système de ce SBT est le système d'auto-apprentissage géré par le composant raisonneur. Ce système assure la fonctionnalité d'auto-apprentissage par l'analyse des traces modélisées et l'extraction de nouvelles règles concernant l'exécution et le déroulement des activités des processus gérés par la plateforme. Ce système prend appui sur une méthode à base de vue inspirée des tables de transitions d'état et un algorithme de classifications. Ainsi, l'ajout de ces règles apprises dans la base de connaissances au fur et mesure de l'exécution de la plateforme assure par conséquent l'évolution dynamique souhaitée.

Quant à l'exploitation des connaissances générées par ce SBT, nous avons illustré sur un cas d'exploitation la fonctionnalité d'autogestion du processus de maintenance, et sur un deuxième cas d'exploitation un service de diagnostic dynamique.

D’autre part, l’impact de la plateforme sur les performances de la maintenance a été évalué sur une simulation dédiée au cas d’utilisation d’une entreprise fictive TEM. Cette simulation a portée sur la gestion de 1000 machines par la plateforme de s-maintenance durant une période de 10 ans. La simulation basée sur un ensemble d’indicateurs de coûts et de temps définis, a mis en évidence une réduction en temps de résolution de problème et en coûts de maintenance. En effet, grâce à l’exploitation des fonctionnalités auto-X dans la plateforme, l’entreprise obtiendrait un gain qui pourrait atteindre 30% des coûts estimés ainsi qu’un gain de disponibilité conséquent grâce à la réduction des temps de traitement de la maintenance. De plus, nous avons pu remarquer que le coût pourrait diminuer au cours du temps, grâce à l’expérience de la plateforme et des règles de connaissances extraites.

Conclusion générale

1. Conclusion

La fonction de maintenance tient une position stratégique dans l'organisation de l'entreprise et répond à des besoins permettant de maîtriser techniquement et économiquement le maintien en condition opérationnel des équipements industriels. Vu l'importance grandissante qu'a prise cette fonction, un accent particulier a été mis sur le développement de systèmes informatiques d'aide à la gestion de la maintenance industrielle. Toutefois, les services, les fonctionnalités et les indicateurs fournis par ces systèmes d'aide ne s'adaptent pas à l'évolution des besoins des utilisateurs et nécessitent une modification voire une mutation en un autre type de système informatique d'aide. Ce qui nous a amené à proposer un nouveau concept de système de maintenance le concept de s-maintenance, concrétisé par une plateforme informatique intelligente orientée connaissance répondant aux besoins évolutifs des utilisateurs.

Pour réaliser cet objectif nous avons organisé notre travail en deux parties, une partie concernant la spécification de cette génération de système et une deuxième dédiée à l'élaboration de celui-ci.

En ce qui concerne la spécification, nous avons tout d'abord étudié les enjeux auxquels doit faire face les systèmes informatiques de maintenance de demain en prenant appui sur les projets de référence dans le domaine comme TATEM et les projets de IMS-Center. Les principales caractéristiques de ces systèmes sont le partage la réutilisation des connaissances, la standardisation, le traitement intelligent et les fonctionnements autonomes.

Ainsi, nous avons été amenés à étudier l'évolution des systèmes informatiques de maintenance et nous avons recensé cinq générations de maintenance dont la quatrième est la e-maintenance présentant les systèmes actuels les plus performants et une cinquième génération que nous avons proposé dans ce travail la s-maintenance.

Afin de situer notre nouvelle génération, par rapport à la e-maintenance, nous avons été amenés à faire un état de l'art sur le concept de e-maintenance. Force de constater que la plupart des travaux dans le domaine ne différencient pas le concept, la plateforme et l'utilisation des technologies.

Par conséquent, nous avons proposé deux définitions, une pour le concept de e-maintenance et l'autre pour le modèle de plateforme associée. Puis nous avons donc défini le concept de s-maintenance comme la réalisation de la maintenance basée sur le partage et la réutilisation des connaissances expertes du domaine en fournissant des services adaptatifs et autonomes. Par contre une plateforme de s-maintenance formalise ces connaissance et les partage entre les différentes applications intégrées dans le système, ce qui garantit une interopérabilité technique et sémantique tout en assurant des fonctionnalités auto-X (auto-traçabilité, auto-apprentissage et autogestion) et fournissant des services à la demande.

Une étude sur les plateformes de e-maintenance existantes par rapport aux caractéristiques spécifiques de la s-maintenance, a souligné que ces plateformes ne pouvaient être reprises pour élaborer une plateforme de s-