PETRA: Process Evolution using a TRAce-based system on a Maintenance Platform

Mohamed-Hedi Karray[1], Brigitte Chebel-Morello[2], Noureddine Zerhouni[2]
[1]University of Toulouse (mkarray@enit.fr)
[2] FEMTO-ST Institute (brigitte.morello@femto-st.fr; noureddine.zerhouni@femto-st.fr)

Abstract.

To meet increasing needs in the field of maintenance, we studied the dynamic aspect of process and services on a maintenance platform, a major challenge in process mining and knowledge engineering. Hence, we propose a dynamic experience feedback approach to exploit maintenance process behaviors in real execution of the maintenance platform. An active learning process exploiting event log is introduced by taking into account the dynamic aspect of knowledge using trace engineering. Our proposal makes explicit the underlying knowledge of platform users by means of a trace-based system called "PETRA". The goal of this system is to extract new knowledge rules about transitions and activities in maintenance processes from previous platform executions as well as its user (i.e. maintenance operators) interactions. While following a Knowledge Traces Discovery process and handling the maintenance ontology IMAMO, "PETRA" is composed of three main subsystems: *tracking, learning* and *knowledge capitalization*. The capitalized rules are shared in the platform knowledge base in order to be reused in future process executions. The feasibility of this method is proven through concrete use cases involving four maintenance processes and their simulation.

Keywords: Trace-Based Systems, process extension, process mining, experience reuse, s-maintenance platform.

I. Introduction

Maintaining the operational condition of industrial equipment, preventing industrial risks and ensuring the safety of persons and assets are just some of the principal challenges for production firms. Maintenance has been recognized as a fundamental function in the company and is transferred from the cost center to the profit center, which has led to massive development of maintenance support systems. From the CMMS[1] to the e-maintenance platform, these systems provide the maintenance staff (agents, experts and managers), with decision-support and a set of services allowing computerized management of core activities for maintenance processes (e.g. intervention, planning, diagnostic, etc.).

However, user needs continue to evolve and cannot be satisfied by services currently provided by the most advanced maintenance support systems on the market such as e-maintenance platforms [Karray et al., 2009]. Despite the presence of an experience feedback function on the platform, the services offered are not completely adapted to user needs. Such adaptation is an essential key to the continuous improvement of the performance of industrial equipment [Chebel-Morello et al., 2013]. In particular, capitalization and sharing of knowledge resulting from experience feedback are indeed constructed on the basis of knowledge and models formalized prior to current user needs [Jabrouni et al., 2011] [Yee fan Tang et al., 2007]. The structure of the model of experience feedback (such as the domain ontology in [Lejarraga et al., 2011] and [Kamsu Fogem et al., 2008] must evolve to respond to the current problem because these methods are static. It offers neither dynamic nor reactive services because these are based on knowledge formalized in the design phase of the system and are not updated due to the lack of exploitation of dynamic experience feedback and analysis of the real behavior of these services. Indeed, Weber et al. affirm that most lesson-learned systems (i.e. experience feedback systems) are passives, stand-alone systems [Weber et al., 1999].

Our challenge is to develop a system that can dynamically adapt to changes in user needs through active lessons. We thus propose to develop a knowledge-oriented maintenance platform [Chebel-Morello et al.,

[1] Computerized Maintenance Management System

2010] called an s-maintenance platform ('s' as in semantics), offering services that evolve along with user needs by means of the reuse of experience.

In this challenge there are different issues to be resolved. Weber introduces architecture for active lesson delivery systems [Weber et al., 1999]. The most relevant past experience is presented to users in a conversational case based plan-authoring system. Craw [Craw, 2009] proposes an agile CBR that "transforms traditional CBR into a dynamic, knowledge-rich, self-organizing, cooperative problem-solving methodology". Cordier suggests Trace-Based Reasoning which works the same way [Cordier et al., 2010]. Trace engineering provides the dynamic aspect of knowledge. Traces of interaction between the user and the computer are considered as knowledge containers of user experiences implicitly stored [Mille, 2006], [Cordier et al.,2009].

Hence, in accordance with the work of Cordier, we are oriented towards an active learning process exploiting event logs to update maintenance process behavior.

Our objective is to propose a dynamic experience feedback method, to exploit process behavior in real execution of the maintenance platform. Feedback on activities will help to make the existing knowledge explicit on the platform, formalizing and sharing collective knowledge. In fact, trace engineering provides a new form of experience reuse and appears to be the means most suited to providing this feedback through the dynamic aspect of knowledge included on the s-maintenance platform.

For this purpose, we have developed a trace-based system called PETRA (Process Evolution using a TRAce-based system) for tracing and analyzing activities conducted via a maintenance platform. The ultimate aim of this system is to extend the behavior of modeled maintenance processes on the platform by analyzing their real behavior.

In agreement with Laflaquière et al. [2008], PETRA will transform execution logs into modeled traces. For a trace model, we use the process view of IMAMO [Karray et al., 2012], a domain maintenance ontology. Also, and in agreement with Rozinat and Van der Alast [2006], we use decision trees as a data mining method, adopting them for the knowledge discovery process in traces inspired from KDD [2] process [Piatetsky-Shapiro & Frawley, 1991] to analyze activity traces. The system outputs are knowledge rules adapted to update the platform's process models and consequently extend them.

In the second section of this paper we provide an overview of event log studies and trace-based systems. We then develop our proposed system, PETRA, as well as its functioning process and specificities. Before concluding and discussing future work, we provide a simulation process applied to prove the feasibility of our proposed approach and to show how this system can extend process behaviors

II. Overview: Trace-based systems

1- Observed event logs

Many studies of the observed event logs have been undertaken in different contexts and with different aims. Their purposes are not similar and they do not use comparable tools. There are many types of application in this field, such as (i) software maintenance: the study by Almeida Maia and Lafeta exploits execution traces, with the aim of helping software maintenance activities to facilitate feature location [Marcelo de Almeida et al., 2013]; (ii) assistance systems in a web-based learning environment [Rech et al., 2007] in which algorithms are proposed to assist users; (iii) assistants to digital applications which represents the system; (iv) social webs, in which a digital trace of the web is exploited to mine knowledge for web processes: Champin et al. [2009] looks at the experiential knowledge determined in the log, which drives its search recommendations in web documents; and (v) web mining, process mining, and data streams.

[2] Knowledge Discovery in Databases

Gaber et al. [2005] have reviewed stream mining, the process of extracting knowledge from uninterrupted records, by interpreting continuous records such as digital traces, which is the case of log files. A Trace-Based System (TBS) may be used to improve the results of the mining process [Champin et al., 2012]. Mathern et al. discuss how the exploitation of modeled traces can improve the mining process [Mathern et al., 2012]. Also, as detailed by Fayyad [1996] and Van der Aalst [2009], TBS can be used to facilitate activity analysis and modeling.

According to Van der Alast, the goal of process mining is to discover, monitor and improve real processes (i.e., not assumed processes) by extracting knowledge from event logs [Van der Alast, 2010]. It is noteworthy that each record in a log file is caused by a given event in the execution of the system, such as user interaction, function call, input or output procedure, etc. Records in log files are continued and are often parameterized, i.e. they show current values of variables and return values of function calls or any other state information [Andrews, 1998].

Execution traces furnish important knowledge that will allow the maintenance platform to improve the quality of its services independently of users, by providing a dynamic service that changes their behavior according to their previous executions (i.e. experiences and interactions). This will minimize the costs of experts, and will explain the knowledge exchanged on the s-maintenance platform.

It is to be noted that, in most works on process mining, topics are oriented towards process reconstruction and discovery [Cook et al., 98], [Agrawal et al., 98], [Van der Alast, 2004]. However, in this study we will focus only on process extension by focusing on each elementary activity and not on the entire process. Process extension considers the presence of an a-priori model. The latter is extended with a new aspect or perspective, i.e., the goal is not to check conformity but to enrich the model [Van der Alast, 2004] as in the case of process model extension with performance data, i.e., some of the decision mining algorithms described by Rozinat and Van der Alast [2006] which extend a given process model with conditions for each decision.

2- Definitions of traces

Before reviewing trace-based systems, let us define what is meant by “trace”. Li et al [2013] provide semiological and etymological definitions of trace from several viewpoints. In computer science, a trace typically concerns the interactive activities between the system and the different users, called actors. Few definitions are presented in different works such as the MUSETTE approach [Champin et al., 2003], TBMS[3] [Laflaquière et al., 2006], KTBS[4] [Clauzel et al., 2009], TRAILS[5] [Walker, 2006], etc. Each definition is adapted to the studied domain that is, in most cases, the learning environment.

According to Jermann et al. [2001], a trace is an observation or a recording of a learner’s interaction with the system to be used for analysis. Similarly, Pernin defines a trace as an indicator of actor activity in a learning situation, whether instrumented or not [Pernin, 2005].

Also, Champin et al. [2004] define the trace as a sequence of states and transitions representing user activities. As a part of the TRAILS project, traces of use in hypermedia are considered as a sequence of actions and are used to identify the overall objective of the user. As for Choquet & Iksal. [2007], data providing information on a learning session are considered as a trace of any broadening of the definition given by the TRAILS project.

Clauzel et al. define an interaction trace as: ‘‘histories of user actions collected in real time from their interactions with the software’’ [Clauzel et al., 2009]. More directly, Zarka et al. define a trace of interaction as ‘‘a story of the user’s actions, step by step’’ [Zarka et al., 2011]. From a different viewpoint, Settouti et

[3] Trace-Based Management System
[4] Kernel Trace-Based System
[5] Personalized and Collaborative Trails of Digital and Non-Digital Learning Objects

al. [2009] define a numerical trace as a “trace of the activity by a user who uses a tool to carry out this activity, saved in a numerical medium”.

Settouti et al. formally define a trace as "a collection of observed elements that can be temporally located". An observed element may be considered as any part of the user's environment (an entity, action, event, etc.) that makes sense to the execution of the observed activity [Settouti, Prié, Marty, & Mille, 2007].

Within the scope of a web-based collaborative working environment, Li et al. consider that a trace not only records the interactions between user and system, but also reflects the potential relationships among collaborators. From this point, they distinguish different types of traces and focus on the definition of a Collaborative Trace (CT) defined as follows: ‘‘A Collaborative Trace is a set of traces that are produced by a user belonging to a group and is aimed at that group’’ [Li et al., 2013].

To summarize, a trace of interaction is a sequence of spatiotemporal objects containing and representing a record of the actor’s activity in a numerical environment in a studied domain.

The actor can be human or software. Concerning the numerical environment, it can be a learning platform, a social network, a maintenance platform or any information system. Hence, the studied domain depends on the application, business process [Abecker et al., 2000], maintenance strategy [Endrenyi et al., 2001], human learning environment, etc. Objects can be modeled in different ways such as a state transition diagram, graph model, petri net model, etc.

Also, a trace is a part of the experience of interaction between a system and its users.

In our research, in keeping with our context, we define a trace as the *“Trace”* of an activity executed by the user, or by the platform of a service (function) integrated into the s-maintenance platform, to carry out a part of the computerized maintenance process.

3- Trace-based systems

It is of note that trace engineering concerns the management of traces with the goal of reusing them. Indeed, traces of interactions can be reused for two purposes, namely assistance and analysis.

Laflaquière and his colleges thus proposed Trace-Based Management Systems (TBMSs) devoted to the management of modeled traces [Laflaquière et al., 2006] in order to analyze and model personal interactive traces. A general framework was introduced to support trace-based system creation and experience reuse. To extend this effort, they have recently built a prototype platform to represent the activities as a set of observed elements: a **k**ernel for **t**race-**b**ased **s**ystems (kTBS).

Therefore, treatments to be applied to traces from their collections are formalized using Trace Based System (TBS) [Cram et al., 2007]. Any system reusing traces is a TBS. The latter is based on three main phases: collection—often followed by a pretreatment step, analysis and exploitation [Bousbia, 2010]. To facilitate the establishment of a TBS, Settouti et al. have proposed an architecture composed of several interconnected modules [Setoutti et al., 2006] [Laflaquière et al., 2006].

The collection system captures the interactions through tracing sources and creates a first trace. The collected traces are structured by the system in a hierarchical structure of classes called an observed trace model [Cram et al., 2007]. The transformation system is the core of the TBS, allowing generation of new knowledge from the collected traces. The choice of transformation model to be applied depends on the application of this trace. The set of traces collected and processed is then accessible through both a query and a visualization system to allow their exploitation, analysis and interpretation [Settouti et al., 2006].

The traceability process is the collection of all traces using a recordable environment saving the activities of maintenance operators. The registration of these marks may refer to different formats such as log files, tracks or trails [Cram et al., 2007] [Choqueet & Iksal., 2007].

According to [Bousbia, 2010] the collection phase provides data labeled "raw traces" or "primitive tracks" not easily exploitable as such. Creating traces from logs is a complex process that requires many operations (filtering, recomposing sessions, etc.). Indeed, logs can contain large volumes of information, sometimes without the context in which the information was generated. The stored data are unstructured and mostly unsuitable for the desired objective [Cram, Jouvin, & Mille, 2007].

To implement the reuse of interaction experience, we must be able to model traces [Cram, Jouvin, & Mille, 2007].

According to [Cram et al., 2007], each observed element is an instance of an observation class belonging to a hierarchical structure of observation classes called the trace model. In addition, the observed elements are interconnected via instance relations defined within the trace model. The trace model defines the vocabulary of the trace, i.e. the observed element types that can appear in the trace, their properties, and the types of relations that can be established between them, together in a way to express the temporality of each event (sequentiality, absolute time, relative time) [Laflaquière et al., 2010]. The trace model can be linked to an ontology defining the user/system interaction. The association of a digital trace and its model is called a modeled trace.

In a learning environment, Laflaquière et al. proposed a trace model composed of actions and observed entity types. The trace is composed of observed elements that are temporally situated within the time of observed interaction [Laflaquière et al., 2010].

To model an activity based on its traces, Georgeon et al. proposed a trace graph structure in two parts: a sequence and an ontology [Georgeon et al, 2012]. The sequence includes event instances with relation instances found between the event instances. The ontology includes event classes with relations between the event classes.

In their Web-based Collaborative Working Environment (WCWE), Li and his colleges propose a trace model when any finished interactions, or actions that rely on the functionalities in the shared space, can be characterized by the modeled traces. In this model, the trace is composed of three basic items: (i) ‘‘Emitters’’ that leave the trace (the subject); (ii) ‘‘Receivers’’ that receive the trace or the object of the trace; and (iii) ‘‘A property and a corresponding value’’, i.e., an original trace that can generally be considered as an information set having several properties with values.

Compared with the trace model proposed by [Clauzel et al., 2011], the model of Li et al. focuses on the interactions among the actors themselves, in the context of classification and comparison of different types of traces. The trace model used in the ITHACA[6] project is also applied to the kTBS platform [Clauzel et al., 2009]. Lafifi et al. proposed a different trace model concentrated on the architecture of the collaborative learning system [Lafifi et al., 2010]. From a different viewpoint, the model introduced by Sehaba [2011] deals with the transformation process for the adaptation of shared traces in accordance with the user’s profile.

Given the importance of the ontology model, along with the different contexts in which maintenance takes place, and with respect to the learning of visualization systems, we thus adopt a trace model from the maintenance domain ontology IMAMO [Karray et al., 2012], called a process model. This model focuses on the actors’ activities and will be explained below. Compared to the other models described above, the process model includes different levels of views concerning concepts, instances and values of properties. It is also devoted to management of different elements involved in the maintenance process such as user profiles, activity patterns and the inputs and outputs of each maintenance task accomplished through the maintenance platform.

III. PETRA architecture and functioning

[6] Interactive Traces for Human Awareness and Collaborative Annotation

Let us recall that that our goal is the further evolution of behavior of the maintenance processes already designed during the development phase of the s-maintenance platform. The objective is to learn more about the execution of the executed activities corresponding to process steps, to analyze their executions and then to extract knowledge rules concerning transitions between these steps. Hence, it will be possible to change their behavior by adding new transitions and/or steps to the process pattern. Figure 1 illustrates possible evolutions of the predefined processes after learning takes place in the second step. Thus, after repeated learning events, we can see the onset of the transition TR13 between steps 1 and 3 during the constraints (Cos1 & Cos2). We also observe the addition of step 5 and the transitions TR15 and TR54 respectively relating Step 5 to steps 1 and 4 while respecting the constraints Cos15 and Cos54.

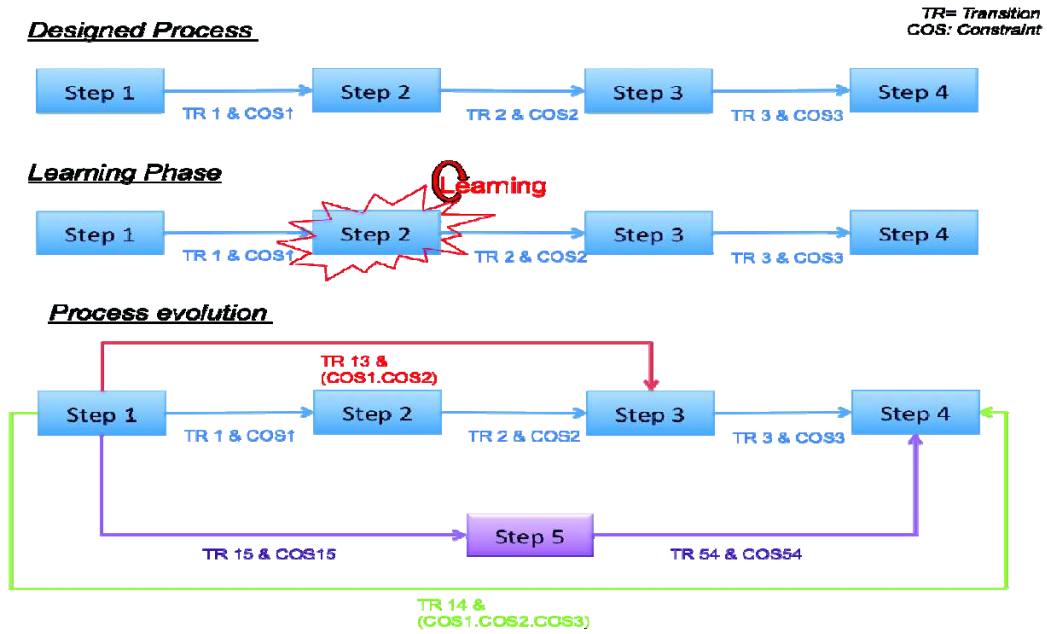


Figure 1 - Schematic illustration of our objective

Thus, we propose PETRA, a trace-based system (TBS) not limited to visualization and display, but with the ambition to interpret these traces through the knowledge base of the platform and its intelligent modules.

To achieve this goal, the architecture of this TBS will consist of three main systems, one for *tracking*, another for *learning* and a third for knowledge capitalization. *Tracking* is composed of two subsystems, one for trace collecting and a second for transformation (processing).

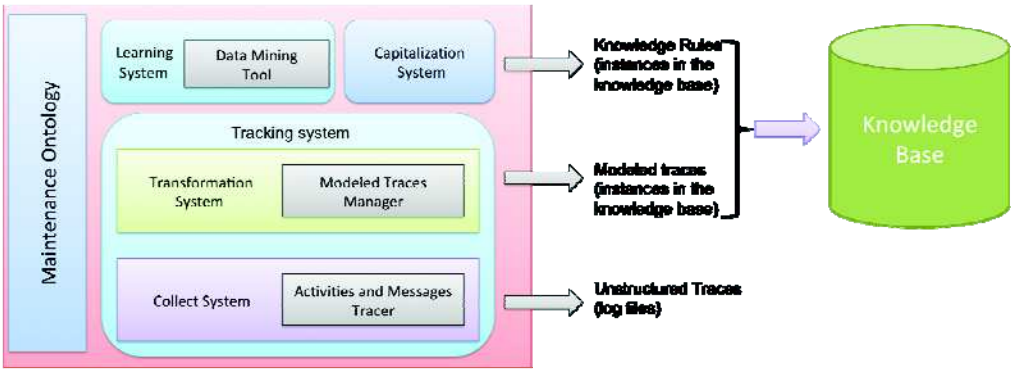


Figure 2 - PETRA architecture

As shown in Figure 2, the TBS that we call PETRA is based on the maintenance domain ontology IMAMO [Karray et al., 2012] so as to model traces as well as to interpret and extract knowledge.

The collection system produces traces of activity interactions and provides log files containing unstructured traces.

The transformation system changes unstructured traces into modeled ones by using the ontology. It then saves the modeled traces as new instances in the knowledge base.

The learning system allows the interpretation and the inference of knowledge rules involving process activities managed by the platform, from the modeled traces already stored in the knowledge base.

From the generated rules, the capitalization system validates the trusted ones and then adapts them to the structure of the knowledge base before updating it.

As shown in Figure 3, PETRA follows the KDT (Knowledge Discovery from Traces) process inspired and readapted from the KDD (Knowledge Discovery in Databases) process in the data-mining domain.

Frawley et al. define KDD as “the nontrivial extraction of implicit, previously unknown, and potentially useful information from data" [Frawley et al., 1992]. It is a formal description process of the knowledge-discovery life cycle. It is to be noted that the KDD process is a user-(analyst-)guided process. Mathern et al. propose to adapt this knowledge-discovery cycle by considering data as activity traces [Mathern et al., 2012]. They affirm that data is obtained via two steps: Firstly, a behavior or process occurs in the real world (a process is composed of activities) and secondly, these activities are observed and data is collected. The record of an activity is a trace of that activity.

Mathern et al. consider that knowledge is built rather than discovered [Mathern et al., 2012]. Hence human involvement in making sense out of traces is paramount.

In our case, we work to minimize user intervention by exploiting the trace model of the IMAMO ontology and we attempt to limit user intervention in the validation phase of the interpretations made by the system.

Figure 3 shows different inputs and outputs of various steps involved in the KDT process. More details about these steps are provided in the following sections.

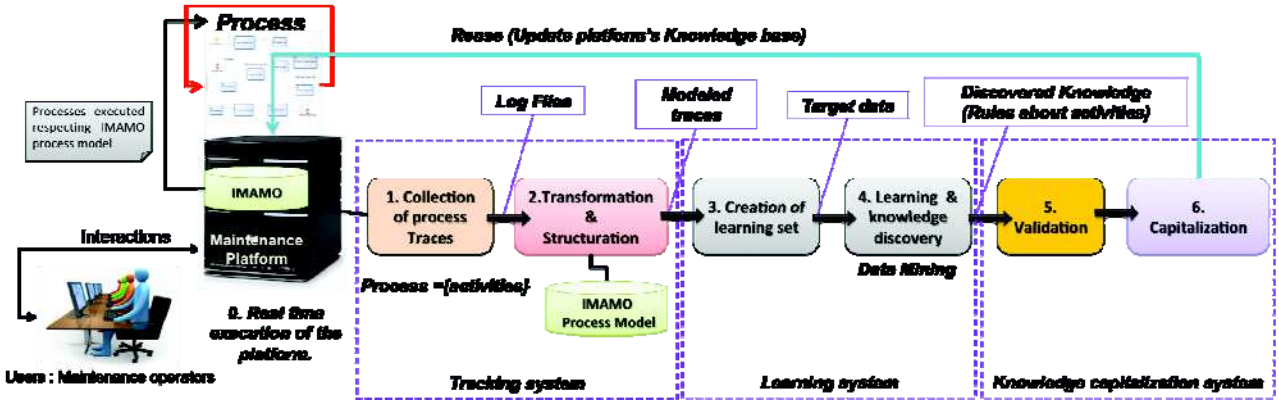


Figure 3 - Closed loop of PETRA functioning

Step 0- Real time execution: While process executions are managed by the maintenance platform and not by PETRA which only collects information about these executions, we can say that this step is considered as PETRA’s input and not as a functioning step. In fact, the execution of processes within the s-maintenance platform follows the knowledge about process executions (composed of activities) stored in the knowledge base defined by the IMAMO ontology [Karray et al, 2011]. The process executions include interactions between users and the s-maintenance platform.

Step 1- Collection of process Traces: During interactions between maintenance operators and platform (activity execution), the tracking system collects interaction traces and stores them in log files.

Step 2 – Transformation & structuration: In this step, the transformation system retrieves log files and parses them by creating a map between file content and the trace model (based on the maintenance ontology

IMAMO) in order to extract modeled traces. Finally, it stores these traces as new instances in the knowledge base. This corresponds to the preprocessing step in the KDD process.

Step 3 – Creation of learning set: This step is included in the learning system. The system retrieves the modeled traces and groups them in an abstract way to be used in the learning phase. This step corresponds to the definition of the target data in the KDD process.

Step 4 - Learning step: The learning system retrieves the modeled traces of activities and adapts them in order to apply the data mining method. The result is presented as knowledge rules concerning these activities. This step corresponds to the data-mining step in the KDD process.

Step 5 - Validation: In this step, the learning system applies some indicators and thresholds to validate the rules generated in the previous step. We note that this step is guided by experts.

Step 6 – Capitalization: The learning system adapts and formalizes the validated rules in order to reflect the structure of the knowledge base before saving them as new instances. This corresponds to the interpretation and evaluation step in the KDD process generally undertaken by the user. The aim of this step is to share the discovered knowledge with the s-maintenance platform so as to be reusable.

1- Tracking system

A- Process of the tracking system

The tracking system consists of collecting all the traces of a given recordable environment while safeguarding the activities of maintenance operators via the platform. This environment should follow the execution process in the maintenance platform. As already mentioned, log files are a good tool for saving traces, but they are under the constraint of the complexity of their structure and analysis. Therefore, in the traceability process (collection and transformation) corresponding to tracking functioning (see Figure 4), we use both log files and modeled traces. Thus, there is a transformation of the log files content to modeled traces by some modules. The transformation is made via a sequential parsing of the log file and extraction of the traced activities via a mapping with the trace model. Finally, the modeled traces are stored in the knowledge base.

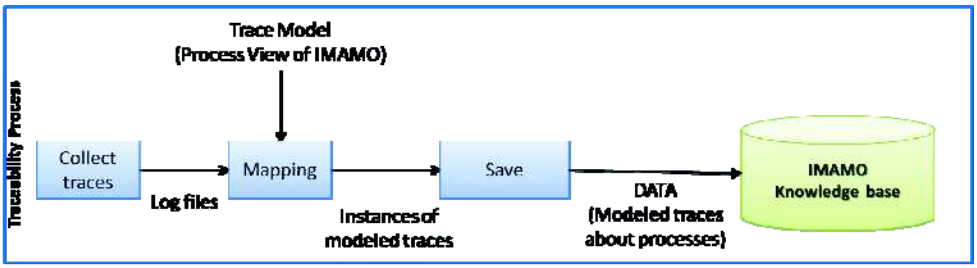


Figure 4 - Traceability process

As we can see in Figure 4, the functioning process of this system is composed of three main activities which are: 1) Collect traces, consisting of saving traces in logs, 2) Mapping, corresponding to the transformation of logs based on the process view of IMAMO, and finally 3) Save, which consists of saving the modeled trace (resulting from the second activity) in the knowledge base.

B- Trace model in IMAMO

Cram et al. argue that the process of interaction between the human and the machine can be classified at several levels (composition relationships may exist between different levels of interactions) [Cram et al., 2007]. Hilbert and Redmiles classify these interactions into six levels of abstraction [Hilbert & Redmiles, 2000] from handling the mouse or keyboard to the traces to be analyzed at the levels of tasks related to the field. The most abstract level (level 6) describes interactions performed on tasks/activities (e.g. making a diagnosis, fixing a bug, etc.). This level operates in PETRA because the goal of traceability and tracking is to

learn from process activities. Hence, the collection of traces will take place on the sixth level. Thus, as shown in Figure 5, the trace model that we adopted includes the activities of the maintenance process managed by the platform users in order to facilitate the processing of traces. In Table 1 we describe the concepts of IMAMO that are exploited in our trace-based system. The adopted model provides all technical, administrative and managerial process views [Karray et al., 2012].

In this model, process is considered as a sequence of related and interdependent activities referring to steps that, in each activity, consume one or more resources to convert inputs into outputs *(ActivityInPutOutPut)*, while respecting a process model *(Process pattern)*. These outputs refer to transitions that, in turn, serve as inputs (referring also to transitions) to the next activity until a known goal is reached (the end of the process).

In this model, each "Process" refers to a "Process Pattern", each "Activity" refers to a "Step" and each "ActivityInPutOutPut" refers to a "transition". Instances of processes and activities present the real interactions made on the maintenance platform and traced by the tracking system. In addition, each activity is performed by one or more actors (users of the platform), taken into account by instances of the concept "Actor". We note that each process is launched by a “Triggering event”. The model supports different types of events such as alarms, notifications of predictive or systematic maintenance and events observed by users. It also takes into account sensor data, their measures, dates and conditions.

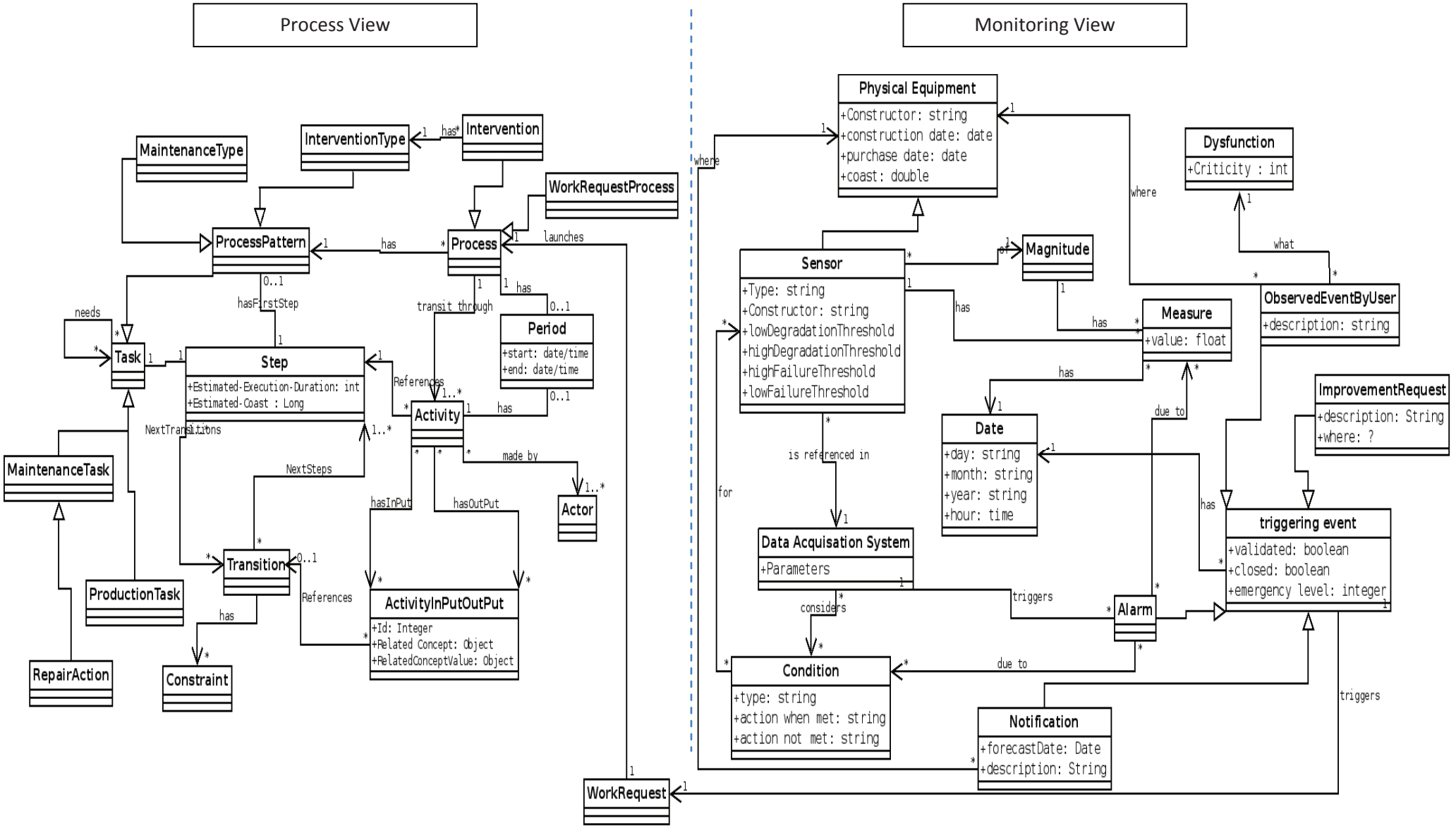


Figure 5 - Process view in IMAMO

Concept Name	Synonyms	Description
Process		Sequence of interdependent and linked activities which, at every step, consume one or more resources (employee time, energy, machines, money) to convert inputs (data, material, parts, etc.) into outputs while respecting a process pattern (i.e. the process pattern presents the general model of the process). These transition outputs then serve as transition inputs for the next step until a known goal is reached.
Process pattern		Pattern that describes a proven, successful approach and/or series of actions. A pattern is a description of a general solution to a common problem or issue from which a detailed solution to a specific problem may be determined.
Maintenance type		Specific type of process pattern concerning the method for carrying out maintenance and the orchestration of processes used in order to achieve the maintenance strategy goals. There are 12 possible types of maintenance which are: 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		Specific type of task concerning one part of maintenance work (e.g. repair, replace, inspect, lubricate, etc).
Intervention type		Specific type of process pattern. This is the principle method of conveying the appropriate activities to all parties involved in an intervention on physical equipment. It presents the generic model of an intervention.
Task		Work assigned or performed as part of one's duties. The task is evaluated by looking at its outcome in terms of completeness, accuracy, tolerance, clarity, error, or quantity.
Repair action		Specific type of maintenance task defined as a physical action taken to restore the required function of faulty physical equipment.
Production task		Specific type of task that terminates in a discrete product or outcome that is observable and measurable.
Constraint		Restrictive condition for the control of transitions between steps.
Activity		Organizational unit for performance of a specific action. An activity is the execution of a task, whether a physical activity or the execution of code. It presents the activity performed by an actor in the real world.
Step		Maneuver undertaken as a part of the progress made towards the progress of a process. It is referenced by an activity.
Work request process		Specific type of process launched automatically or by an actor when receiving a work request. It allows management of the work request until resolution of the original problem triggering the event and the end of the intervention process, including edition of the intervention report.
Transition		Passage from one step to another in the course of a process. This is the connection between two steps in a process. A transition ensures the move from one step to another according to a particular event.
Activity Input Output		Parameters, values and / or input events triggering the launching of an activity. These parameters can be the outputs from other activities. The execution result of an activity (output) is the input from another activity. Activity Input Output is the passing link from one activity to another. Therefore, each Activity Input Output can refer to a Transition.
Actor		Person or a computer system that interacts in the maintenance process.
Role		Prescribed or expected behavior associated with a particular position or status in the maintenance process.
Measure	Measurement	Number or measure or quantity captured by a sensor.

Magnitude		Greatness of size or amount. It presents the property of relative measure.
Data acquisition system		Software system (abbreviated with the acronym DAS or DAQ) that typically converts analog waveforms generally retrieved from sensors into digital values for processing.
Condition		Environmental or functional requirement defined to supervise (monitoring task) specific physical equipment or a place (e.g. site) by the use of sensors and data acquisition systems.
Triggering event		Something that happens to physical equipment at a given time that triggers a specific maintenance process, which is a work request process.
Alarm		Type of triggering event launched from a data acquisition system indicating that there is a measure from a sensor violating some conditions concerning a specific equipment or environment.
Improvement request		Triggering event concerning a specific or general request for the improvement of physical equipment. An improvement is defined as the combination of all technical, administrative and managerial actions, intended to improve the dependability of physical equipment, without changing its required function.
Event observed by user		Type of triggering event concerning a dysfunction of physical equipment observed by the user who is a human resource.
Notification		Type of triggering event giving notice of future events such as planned maintenance or the prognostic RUL.

This model allows the instantiation of different maintenance types that is a subclass of the process pattern. For example, the process pattern *systematic maintenance* has *Scheduling* as the first step; the next transition corresponds to the *notification (Date)* that launches the next step corresponding to *management process*. The same occurs for the *predictive maintenance* process pattern, where *prognostic* presents the first step, the *RUL*[7] is the next transition and finally the management process as the next step. Hence, according to this model, each activity executed on the platform references a step in the process pattern, referencing the process including this activity.

In a more refined way, the trace model can be summarized in the concepts *Process, Activity, Actor, ActivityInPutOutPut* as well as the relationships that structure them (see Figure 6).

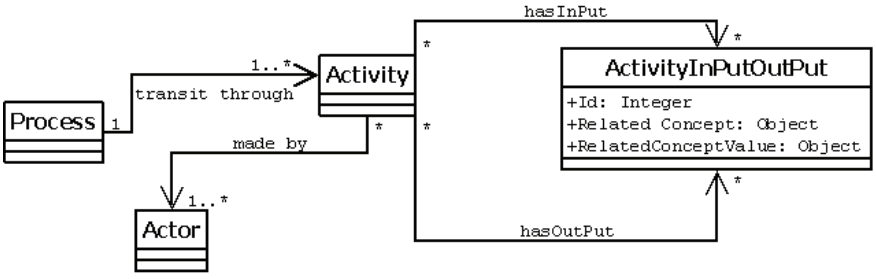


Figure 6 - Refined trace model

We note that IMAMO was developed in the language PowerLoom. Thus, we evoke PowerLoom's static and dynamic query optimizer that is similar to ones used in DBMS. PowerLoom has a strong relational data base interface that allows it to use the power of databases to handle large instance bases [Chalupsky et al., 2010]. These advantages allow the creation of different data views [Bertino, 1992], such as databases through joint queries between concepts and relationships.

[7] Remaining Useful Life

While the process view allows the instantiation of all maintenance processes, Figure 7 is a visual reconstruction of the general management process of condition-based maintenance [Jardine et al., 2006] instantiated via this part of the ontology.

Indeed, "Condition-based maintenance" is an instance of « Maintenance type » which is a sub-concept of "Process Pattern". Also, the steps and transitions of the « Condition-Based maintenance » process are already instantiated.

We note that the case of the management process in condition-based maintenance will be our central thread throughout this study in order to illustrate the functioning and the added value of PETRA on the s-maintenance platform.

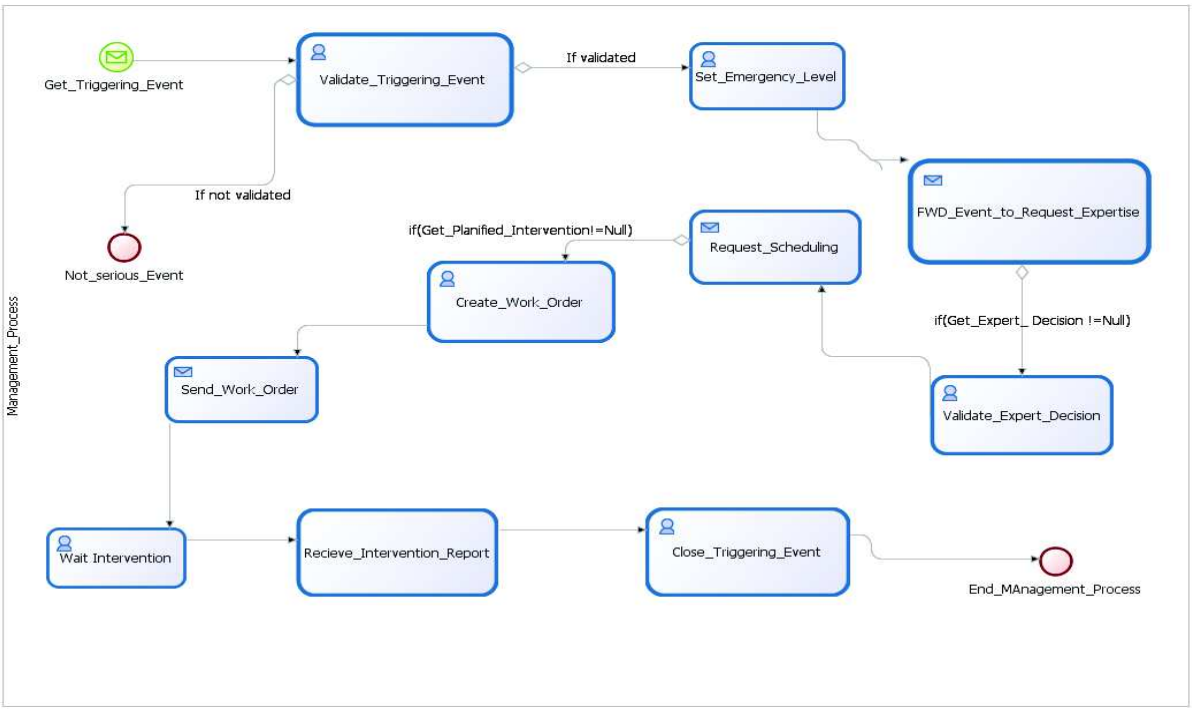


Figure 7 - Schematic illustration of the instantiated Condition-based Maintenance management process pattern

C-Formalized trace model

It is to be noted that here we are inspired from the structure of the state transition table that shows to which state a finite state machine will move, given the current state and inputs [Breen, 2005] used to formalize traces based on the trace model.

We define two levels of traces; *Concrete traces (CT)* corresponding to collected traces instantiated (i.e. ABOX of the knowledge base) with the trace model, and *Abstract traces (AT)* corresponding to concepts related to the Concrete traces (i.e. TBOX of the knowledge base). We use relational algebra [Imieliński, & Lipski, 1984] to formalize traces.

A Concrete trace of an executed activity is defined as the *5-uplet CT= {P, TE, Act, Att, E}* while:

- *P: a process instanced into the platform to manage the maintenance of physical equipment*
- *TE: the triggering event causing the launching of the process P*
- *Act: a set of instances of activities performed by an actor within a given process P*
- *Att: a set of attribute values of activities "Act" performed by the actor*
- *E: the concerned physical equipment on which the process P is performed*

Activities included in the Act set are essentially composed by the concerned activity A, the previous activity P and finally N, the next one (its successor). Act is formalized as:

$$Act = \underline{A} \cup \underline{P} \cup \underline{N} \quad while$$

$$\underline{P} = [\pi_{hasInPut.activity} (\sigma_{Activity = \underline{A}} (hasInPut \bowtie_{ActivityInPutOutPut} hasOutPut)]$$

$$\underline{N} = [\pi_{hasOutPut.activity} (\sigma_{Activity = \underline{A}} (hasInPut \bowtie_{ActivityInPutOutPut} hasOutPut)]$$

In contrast, the set of attribute values concerns the different ID of inputs and outputs of A, formalized as:

$$Att = \pi_{ActivityInPutOutPut} [(\sigma_{Activity = \underline{A}} (hasInPut)) \cup (\sigma_{Activity = \underline{A}} (hasOutPut))]$$

The triggering event of the process is formalized as:

$$TE = \pi_{Triggering\_event} [\sigma_{Process = P} (Launches)]$$

An abstract trace of a traced activity is an abstraction of the Concrete trace. It contains the model concepts related to concepts defined in the concrete trace. An abstract Trace is defined by the *6-uplet AT= {PP, TE, TR, Sp, Cos, E} emerged from the Concrete trace:*

- *PP: the process pattern related to the process P registered in the CT.*
  - $PP = \pi_{ProcessPattern} [\sigma_{Process=P} (Process)]$
- *TEC: the class of the triggering event causing the launching of the process P registered in the CT.*
- *Sp: Set of steps in the process. It corresponds to the steps referring to the activities registered in the CT.*
  - $Sp = \pi_{References\_Step} [\sigma_{Activity=\underline{A} \text{ or } \underline{P} \text{ or } \underline{N}} (Activity)]$
- *TR: transition set, corresponds to the transitions referring to the inputs and outputs of activities registered in the CT.*
  - $TR = \pi_{References\_Transition} [\sigma_{AcitivtyInPutOutPut \text{ in } \{Att\}} (ActivityInPutOutPut)]$
- *Cos: Constraint related to transitions in TR.*
  - $Cos = \pi_{Constraint} [\sigma_{Transition \text{ in } \{TR\}} (hasTransition)]$
- *EM: the equipment model corresponding to the concerned physical equipment registered in the CT.*
  - $EM = \pi_{Equipment\_Model} [\sigma_{Physical\_Equipment=E} (hasModel)]$

2- Learning system: knowledge discovery phase

A- Process of the Learning system

In traditional trace-based systems, the system visualizes the modeled traces for the user who interprets them to obtain new knowledge. The learning system in PETRA aims to discover new knowledge by interpreting the structured (modeled) traces without user intervention except in the validation phase.

Hence, once the traced activities are structured and stored as new instances in the knowledge base, the learning phase is initiated to interpret them and learn new knowledge. The learning process is the operation that automates the steps ensured by the learning system presented in the PETRA process. According to Nilsson, learning, like intelligence, covers a wide range of processes that are difficult to define with any precision. Nilson relates the definition of learning to “knowledge acquisition”, “skills understanding”, “experience by reuse" and "modification of a behavioral tendency by experience" [Nilsson, 1998]. Machine

learning refers to a system that can automatically acquire and integrate knowledge, a system that is able to learn from experience, training, analytical observation and other means.

A machine learning system usually begins with some knowledge organized in order to be able to interpret, analyze and test the acquired knowledge [worldofcomputing, 2011]. Thus, machine learning techniques are considered to be the heart of any learning process, to produce learned (acquired, discovered) knowledge [Lowe & Shirinzadeh, 2005].

Therefore, the learning process can understand different relationships and interactions in order to conclude dependencies and thus generate rules for new process models of maintenance.

However, even after pretreatment and transformation in the traceability process, the collected traces generally come in bulk and are sometimes not expressive. Analysis during the learning process then becomes difficult due to the complexity of the task. The learning system must process the traces using different methods of analysis and learning including statistical and data mining methods that remain among the most frequently used [Hilbert & Redmiles, 2000]. [Agrawal et al. 1998] confirm that the application of data mining techniques in the context of business processes can be beneficial beyond the analysis of decision activities that have been performed.

As noted by Xu et al., the learning process must satisfy the consistency condition, which means that the outcome must be consistent with the original knowledge already in the knowledge base [Xu et al., 1995]. The learning process is mainly based on the knowledge models of IMAMO and the knowledge base of the maintenance platform maintenance.

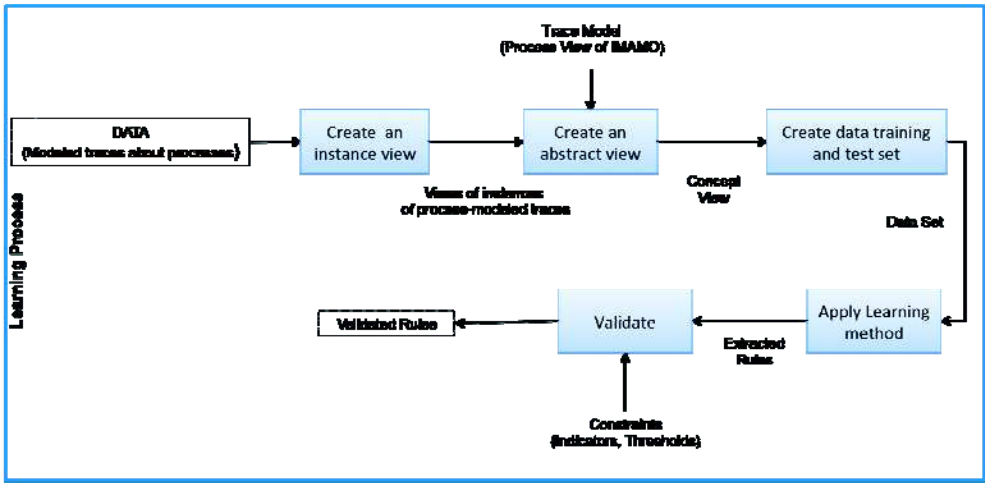


Figure 8 - Learning process

As shown in Figure 8, the first two steps in this process deal with the creation of data views that contain the needed knowledge to be analyzed from the knowledge base. These steps are performed to present the knowledge in a structured form (e.g. table with a specific schema).

B- View creation

The learning system processes instances of concept activity and/or process, but in order to discover new knowledge, the learning system must be applied on a conceptual level (concepts) and not on the instance level.

It is of interest to transform and to adapt the *Concrete Traces* and the *Abstract Traces* into view form so as to enable the learning system to analyze traces in a simplified way and on an abstract level.

Based on the formalized ***CT***, the system builds a first level view containing instances. In its second step, based on the formalized ***AT***, it then generates a second view containing instances of the abstract concepts, each instance of activity or process being replaced by the instance that refers to the concepts "Process pattern" or "Step". Tables 2 and 3 illustrate the creation of views. It must be noted that the view schemes are pre-designed into the system in accordance with the formalized traces CT and AT.

The pattern of the first view is presented by a 6-uplet containing the concerned activity with inputs/outputs, the following activities in the process managing the maintenance of the equipment:

< PreviousActiviy.Id, ActivityInPut.Id, Activity, ActivityOutPut.Id, NextActivity, PhysicalEquipment >.

Table 2 - First level view

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

With respect to Concrete Traces, Table 2, as it is, does not provide any exploitable information. We will therefore proceed on the use of Abstract Traces. Hence, the second level view is generated from the first, according to the formalized AT. The following 8-uplet composes the schema of the latter:

<PreviousStep, Transition, Step, ActivityOutPut.Concept, Transition, Constraint, NextStep, EquipmentModel>.

Step and *NextStep* in this view correspond to the *Second Activity* and *Next Activity* in the first view. Transitions correspond to the transitions of TR in the AT. Regarding *Constraint*, this is generated from *Transition* corresponding to *ActivityInPutOutPut*. *EquipmentModel* corresponds to *"Equipment Model"* referring to the maintained *"Physical Equipment"*. An example of the second level view is provided in Table 3.

Table 3 - Second level view

PreviousStep	Transition	Step	Transition	Constraint	NextStep	Equipment Model
GetMeasure	Alarm Measure Value = 46	Validate Alarm	Alarm.validated = True	Alarm.Measure.Value>42	Set Emergency Level	Pusher
Validate Alarm	Alarm validated = True	Set Emergency Level	Alarm.Emergency = Very High	Alarm.Measure.Value>42 And Alarm.Validate=True	Request expertise	Pusher
GetMeasure	Alarm Measure Value = 44	Validate Alarm	Alarm.validated = False	Alarm.Value>42	Close Work request	Puller
GetMeasure	Measure = 41	Control Data	Condition Not Violated	Measure.Value>42	Control Data	Indexer
GetMeasure	Measure = 42	Control Data	Condition Not Violated	Measure.Value>42	Control Data	Indexer
GetMeasure	Measure = 43	Control Data	Condition Violated	Measure.Value>42	Create Alarm	Indexer

From the second view, a learning set composed of the training and test set is built and then the data mining method is applied in the next step of the KDT process.

C- Illustration of the process steps by a maintenance case

Inspired by Rozinat et al. [2006], we convert every activity into a classification problem [Mitchel, 1997], where the classes are the different executed activities.
For the solution of such a classification problem various algorithms are available [Mitchel, 1997]. Also, in agreement with Rozinat et al. [2006], in this study we adopt the decision tree approach which is one of the most popular inductive inference algorithms. It provides a number of extensions that are important for practical applicability and can manipulate "symbolic" and digital data. Let us briefly recall that a decision tree assigns a class (or output) to an input model, filtering the model through tests in a decision tree that allows mutually exclusive and comprehensive rules [Nilsson, 1998].

With this as our framework, we have thus adopted the algorithm C4.5 [Quinlan, 1990 & 1993] from the Weka[8] platform to create a decision tree based on a set of entry data. The C4.5 system consists of four main programs: 1) the decision tree generator (C4.5) that builds the decision tree, 2) the production rule generator (C4.5 rules) which generates production rules from non-pruned trees, 3) the decision tree interpreter that classifies elements by using a decision tree and 4) the production rules interpreter that classifies elements using a set of rules. The C4.5 outputs are the right decision tree, the error tables for training and testing as well as the confusion matrix.

Thus, from these differing viewpoints, the various files needed for the algorithm C4.5 are automatically created. The forth step is to run the C4.5 algorithm corresponding to the learning method used in our case. According to the percentages of correct classification and the percentages of error, the system may or may not validate the learning results. Validation takes place according to strict thresholds previously defined concerning these indicators as shown in Table 1 as well as to expert evaluation.

a- Preparing the learning set of C4.5

From the second view, we construct the learning set of C4.5. This set consists of a collection of data sequences. Each sequence is a tuple of values for a fixed set of attributes (independent variables) A = {A1, A2, ..., An} and a class attribute (or dependent variable). The attribute A_a is described as continuous or discrete according to its values which are numerical or nominal (Kohavi & Quinlan, 2002).

In our study, we have taken ActivityOutPut.Val as a dependent variable and {ActivityInPut.Concept, ActivityInPut.Value, ActivityOutPut.Concept, constraint ConcernedEquipmentModel} as a set of independent variables.

Figure 9 shows an example of a C4.5 learning set while the validateAlarm.names file defines attributes and the validateAlarm.data file contains instances.

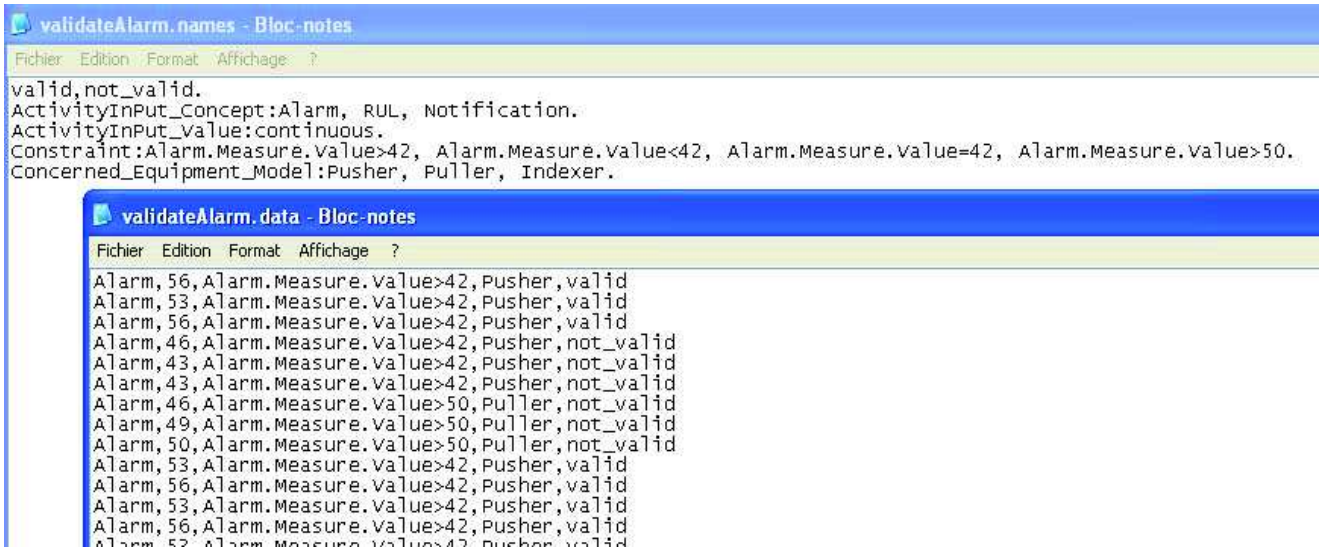


Figure 9 - Example of C4.5 files

b- C4.5: Launching and validation

The automatic launch of the C4.5 algorithm can happen either periodically (e.g. every month), or based on a threshold involving the number of identified instances in the knowledge base (e.g. the algorithm will launch at every 100 new instances of a concept). We chose the second alternative because the number of instances does not uniformly vary in time. This number is checked by the system whenever a new instance is added to the knowledge base. Figure 10 shows an example of a decision tree obtained after the C4.5 execution.

[8] Waikato Environment for Knowledge Analysis: http://wekaclassalgos.sourceforge.net/

Validation of learning outcomes will be according to four indicators: Correctly Classified Instances, Kappa statistic, relative absolute error and relative squared error. Kappa statistic is a measure of corrected chance agreement between estimated and true classes; it is calculated as [(Observed agreement – Chance agreement)/(1-chance agreement)]. The error rates are used for numerical prediction rather than classification. In numerical prediction, predictions are not just good or bad: the error has a magnitude and is reflected by their measures [Bouckaert, 2010].

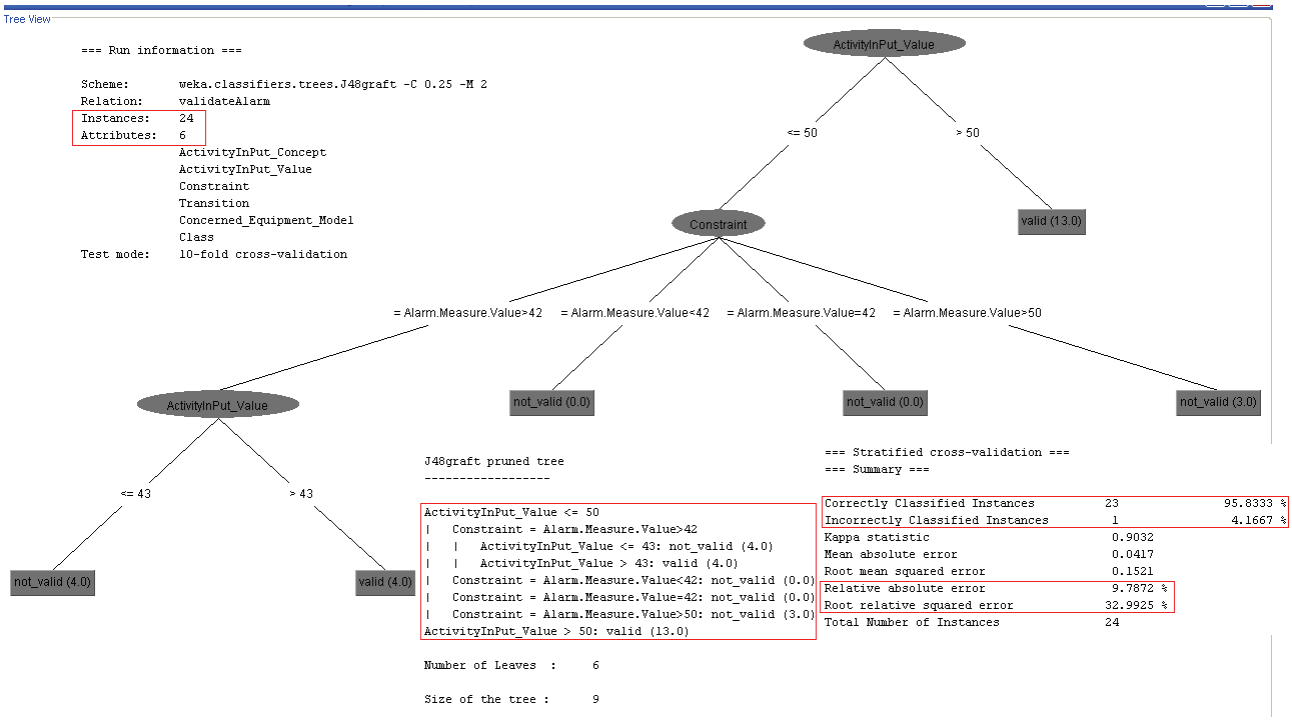


Figure 10 - Decision tree resulting from C4.5 execution

As we are oriented towards semi-automatic decisions with minimal human intervention, we have adopted a validation policy that sets high confidence thresholds as a default configuration of the system (see Table 4). These thresholds can be modified and updated by the platform administrator. In addition, we note that all thresholds must be met in order to validate the learning outcomes.

Table 4 - Validation thresholds

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

3- Knowledge capitalization system:

A- Process of Capitalization system

Capitalizing knowledge constitutes the last step in our KDT process. The objective of this activity is to confront the new rule with the existing ones in the knowledge base in order to avoid inconsistency, contradiction, and conflict, and also to adapt the validated rules generated during the learning step and save them as new instances in the knowledge base. The operation of this step follows the process shown in Figure 11 and can be divided into three activities:

1. The first activity is to create an interpretation table that includes both validated rules and rules already existing in the knowledge base, to compare them and to establish correspondences between them.

2. The second activity is the identification of actions that will be taken in relation to each generated rule (ignore, delete and add). The subsumption opportunities between rules to be added are then studied.
3. Finally, the knowledge base is updated by adding new rules and removing or modifying existing ones. We note that this update mainly concerns the instances of the concepts "Step", "Transition" and "Constraint".

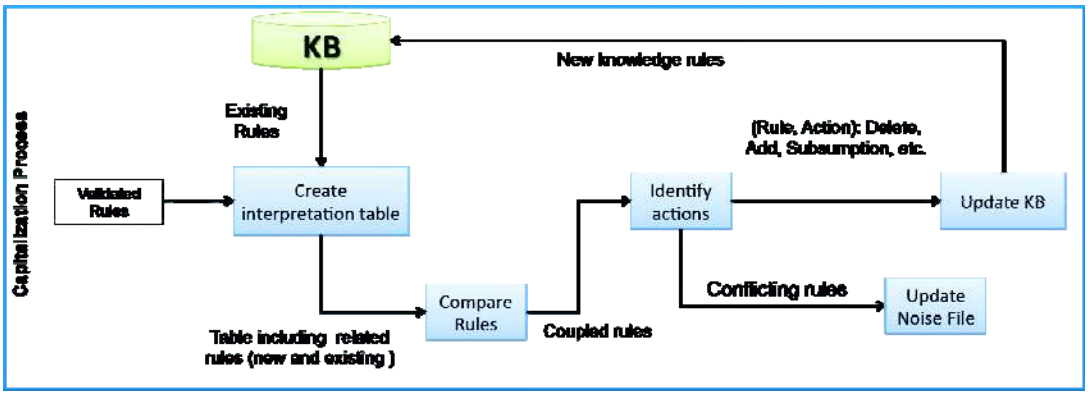


Figure 11 - Capitalization process

B- Interpretation table and action identification

A resulting decision tree is composed of a set of nodes corresponding to constraints involving independent attributes, and of a set of leaves corresponding to the values of class attribute that are the different transitions in our case. These nodes and leaves can be presented as paths. Each path is a sequence of nodes.

Hence, a decision tree can be expressed as rules. Each resulting rule can be defined as the couple (Path, Leaf) while the relation between these two elements respects the format:

If Path then Leaf

While Path corresponds to constraints and the leaf corresponds to transition, according to our trace model, then the rule resulting from the decision tree is presented as:

If {Constraints} then Leaf

Figure 12 provides an example of the learning of resulting rules. The system must translate these rules into a form respecting IMAMO. The result is composed of rules defined and expressed with the attributes of the C4.5 file. Therefore, the system takes advantage of the fact that the structure of all data files is the same for all instances of the concept “Step” and the attributes put forward concepts from IMAMO. Thus, the resulting rules are expressed using the ontology concepts and/or related terms. The following example provides rules about the possible transitions and constraints related to "Validate alarm" an instance of the concept “Step”.

The result is analyzed using the second level view related to the analysis step in the learning process. This view makes it possible to understand the relationship level between the attributes used in the rules and concepts. For example, in the case of "Submit alarm", “Step,” and “ActivityInPutValue” the value “Alarm.Measure.Value” is presented in the second level view.

```
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 12 - Example of learning resulting rules

To interpret the rules generated in the learning phase, the system establishes a map between rules, based on the second level view and the knowledge base. It builds an interpretation table having as data schema the 4-tuple *(Step, classified constraint, learned associated Transition, existing constraint)*. The table includes the *"Step"* on which learning took place, its associated classified constraint, the value of the resulting transition of this constraint and, finally, the constraint associated with the Step in the knowledge base. This table compares existing to classified constraints according to learning. It also enables creation of relations between these new rules and values of associated transitions in the knowledge base. Table 5 is the interpretation table of the learning result of the "Submit alarm" Step.

Table 5 - Interpretation Table

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

Through comparison between learned and existing rules, the system identifies actions to be undertaken regarding each learned rule, either an action of "*ignore"* or of *"modification"*.

According to our knowledge base and trace models, rules must be formalized as:

***If* LTR *Then* P.NextTransition (LTR) And LTR.NextStep (N)**

While *LTR* = the learned constraint, *P* = the previous step and *N* = the next step.

From the compared rules in Table 4, the following actions are identified:

- Ignore and save conflicting rules
  - The constraint Alarm.Measure.Value <= 50 extracted from the learning phase is contradictory to the existing constraint Alarm.Measure.Value> 50.
  - Henceforth, conflicting rules are saved in a file called noise file which is treated by the expert in order to observe the evolution of these rules and to analyze the origin of the conflict.
- Modify existing rules:
  - Alarm.Measure.Value> 42 Alarm.Measure.Value <42 Alarm.Measure.Value = 42 will be replaced by the rules Alarm.Measure.Value <= 43 and Alarm.Measure.Value> 43.

Based on these identified actions and values of the learned transitions, the system defines the rules and relationships to be set up in the knowledge base. For example, from actions identified in the case study, two tasks are defined:

- Remove relationships with existing constraints to be modified (between Transition and Constraint)
  - Alarm.Measure.Value >42
  - Alarm.Measure.Value <42
  - Alarm.Measure.Value =42

- Add new relationships for the learned rule
  - Rule1: If (Alarm.Measure.Value> 43) then [(NextTransition ← ValidAlarm) && (NextStep←SetEmergencyLevel)]
  - Rule 2: If (Alarm.Measure.Value <= 43) then [(NextTransition ← NotValidAlarm) && (NextStep←SetEmergencyLevel)]
  - Rule 3: If (Alarm.Measure.Value> 50) then [(NextTransition ← ValidAlarm) && (NextStep←SetEmergencyLevel)]

After mapping rules/actions, the system checks whether or not there are conflicting rules or whether there are more general rules among them than specific ones (check subsumption possibilities).

In our use case, rule 1 is more generic than rule 3; hence the most general rule is adopted. Therefore, at the end of this activity only rule 1 and rule 2 are validated.

C- Updating the knowledge base

An example of KB's update is shown in Figures 13 and 14 that provide upstream and downstream overviews of the knowledge base and the operations performed during the update.

```
(Assert (Step Validate-Alarm))
(Assert (Step getTriggeringEvent))
(Assert (Step setEmergencyLevel))
(Assert (Step CloseNotSeriousAlarm))
(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 (hasConstraint Alarm-Valid Alarm.Measure.Value>42))
(Assert (hasConstraint Alarm-Not-Valid Alarm.Measure.Value<=42))
(Assert (hasTransition (getTreiggeringEvent Alarm.Measure)))
(Assert (NextStep (Alarm.Measure Validate-Alarm)))
(Assert (hasTransition (Validate-Alarm Alarm-Valid)))
(Assert (NextStep (Alarm-Valid setEmergencyLevel)))
(Assert (hasTransition (Validate-Alarm Alarm-Not-Valid)))
(Assert (NextStep (Alarm-Not-Valid CloseNotSeriousAlarm))
```

Figure 13 - Initial knowledge base

Figure 3 contains a part of the knowledge base concerning the instantiation of a pattern of CBM process shown schematically in Figure 7. It presents a set of steps, transitions and their related constraints in this process.

According to the validated rules in the previous phase, the system begins by retracting instances to be deleted, and it then adds new instances. The knowledge base update in Figure 14 shows the retracted relations between the transitions Alarm-Valid and Alarm-Not-Valid with some constraints. Also, the figure shows the instantiation of new steps and constraints as well as their relations with the transitions Alarm-Valid and Alarm-not-Valid. It shows the new knowledge base resulting after the learning phase in the CBM process.

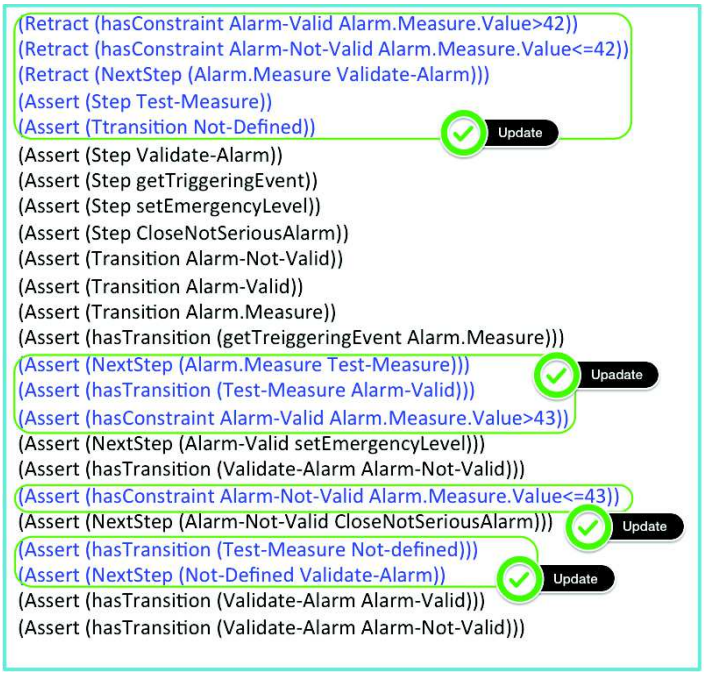


Figure 14 - Updated knowledge base

Figure 15 shows the new version of the CBM process after the update of the KB with the new rules. We observe the emergence of a new activity that addresses the test of measures and then the emergence of three possible transitions.

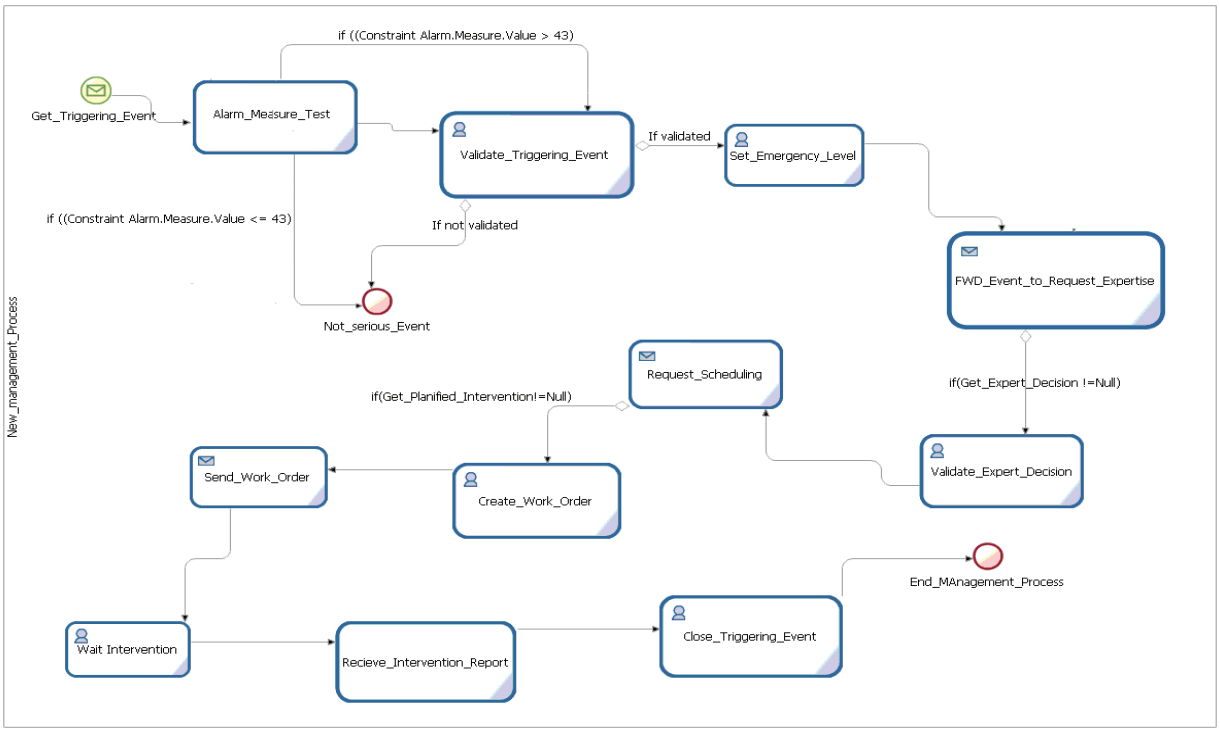


Figure 15 - The new version of CBM process

4- Simulation and feasibility

It is to be noted that the PETRA system, which will be an integrated part of the s-maintenance platform, is not fully developed. Hence, in order to evaluate the feasibility of this approach we will base our evaluation on simulation using some external tools.

We have applied the approach of our trace-based system to four maintenance process patterns, namely the diagnosis process, the scheduling process, the expertise process and the prognosis process, in the scope of the entire CBM process composed of the flowing activities which can be fully or partially followed: Data acquisition, Detection, Condition assessment, Diagnostic, Expertise, Decision and Scheduling. This process can be partially executed where not all activities are taken into account.

A- Simulation context

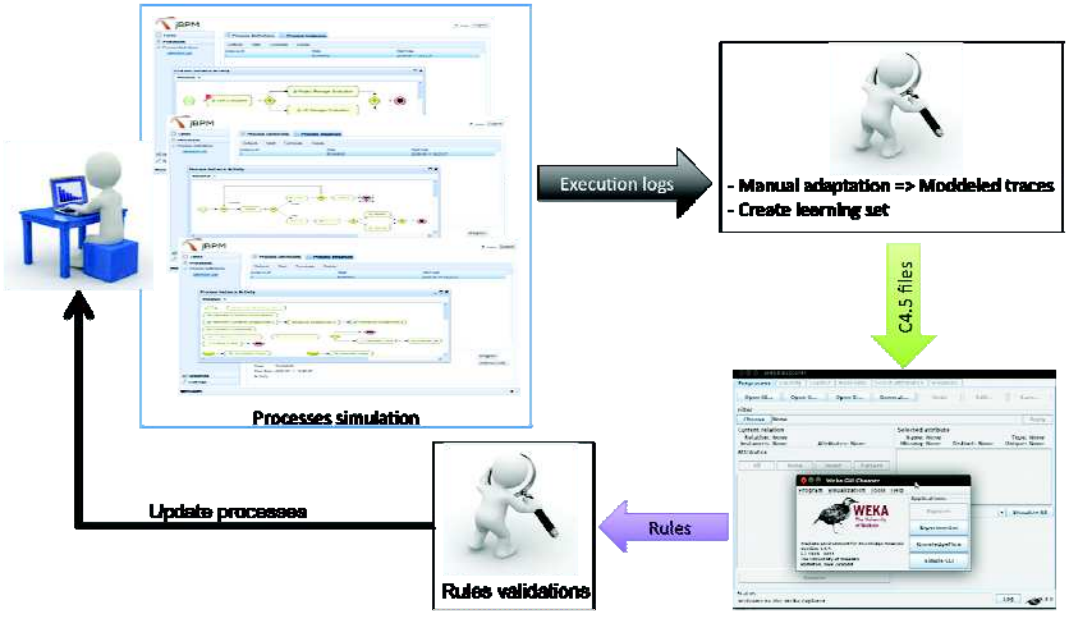
This simulation is based on a real industrial application. The application is a supervised industrial system for pallet transfer (SISTRE) consisting of a flexible production system, composed of five robotized working stations which are served by a transfer system of pallets organized into double rings (internal and external). Each station is equipped with pneumatic actuators (pushers, pullers and indexers) and electric actuators (stoppers) as well as a certain number of inductive sensors (proximity sensors). An inductive read/write module identifies and locates each pallet and provides information relative to the required operation at a concrete station. Pallet movement is ensured by friction on belts powered by electric motors. Each pallet has a magnetic label used as an embedded memory which can be read at each station by means of magnetic read/write modules (Balogh) and memorizes the product assembly sequence. More detailed description of the system is found in Chebel-Morello et al. [2013].

To simulate PETRA here, we used jBPM6[9] and the Weka platform. Let us remember that Weka (Waikato Environment for Knowledge Analysis) is a popular series of machine learning software written in Java. Weka provides the J48 algorithm which is the implementation of the C4.5 algorithm.

In contrast, jBPM6 is an extensible workflow engine written in pure Java which executes business processes using the latest BPMN 2.0 specification, supporting the entire life cycle of the business process (from authoring through execution to monitoring and management).

jBPM makes it possible to replay a simulation scenario, to create scenarios from the information in the history log and to replay the execution of a process instance. It can also dynamically change running process instances, adding tasks on the fly. In addition, this engine provides a history log storing all information about the execution of all the instanced processes, corresponding to the collect system in PETRA architecture.

Figure 16 presents the architecture of our simulation process. We have developed the four processes in jBPM6 and we instance and execute each process 100 times, one by one. Then, from log files provided by jBPM for process executions, we construct modeled traces and thus create a learning set (C4.5 files) as has been defined in PETRA. We then launch the learning step in Weka to generate rules about process activities. Finally, as maintenance experts, we transform and then validate rules classified by Weka.



[9] http://www.jboss.org/jbpm/

Figure 16 - Simulation process

B- Results obtained

This simulation allowed us to observe the results summarized in Table 6. The application enables us to observe the evolution of these processes through the rules learned through specific decision activities in the cited processes. After two launched learning phases on *management process,* three rules are validated by our maintenance expert, which adds a new step to this process along with three transitions and constraints, as well as an invalidated rule that we call a noise. The same thing is noted concerning the diagnostic process. The learning step on *scheduling process* does not provide any rule.

Table 6 - Summary of simulation results

Process Pattern name	*Number of pre-designed steps*	*Number of instances (execution simulation)*	*Number of launched learning phases*	*Number of validated rules*	*Number of added steps*	*Number of added transitions*	*Number of added constraints*	*Number of noises (invalidated rules)*
Management Process	13	100	2	3	1	3	3	1
Diagnostic process	9	100	3	3	2	4	5	1
Scheduling process	10	100	1	0	0	0	0	0
Expertise process	7	100	1	1	0	1	1	0

To be more precise, we furnish further details about the results obtained by focusing on the diagnostic activity in the CBM process concerning the SISTRE system:

In fact, the CBM process instantiated for SISTRE is composed of 7 activities:

- Activity 1: Request localization
- Activity 2: Locate Pallet
- Activity 3: Request Sensor Data
- Activity 4: Condition assessment
- Activity 5: Diagnosis failure
- Activity 6: Expert validation
- Activity 7: Decide actions

Hence, while applying PETRA services to the “diagnosis failure” activity, we obtained the following rules validated by our maintenance expert:

Validated rules	*Added steps*	*Added transitions*	*Added constraints*
IF hydraulic supply in [6-7 bars] ***THEN*** verify Electric valve	Send Problem = Verify Electric valve	Hydraulic supply value	6<Hydraulic supply< 7
IF pallet is stopped in post-zone ***AND*** stopper S6 is low ***THEN*** Balogh1's problem	Send Problem = Balogh1's problem	Zone and Stopper Position	Zone = post-zone stopper S6 position = **low**
IF electric supply in [180 V- 210V] AND pallet stopped in station2 ***THEN*** Check motor1 velocity	Check Motor velocity	Station and Motor	electric supply in [180 V-210V] pallet position = station2

By means of the proposed Trace-based system, the experience transiting via the platform may be captured by the different maintenance strategies (corrective for diagnosis and systematic for task planning), and then exploited so as to make the knowledge explicit and allow for an experience feedback which should be adapted to the current needs.

In fact, this experience feedback is specific to the behavior of the maintained system. In the above example, discovered rules are about repetitive problem solving, provided by diagnosis activity or expert validation. PETRA discovers rules that may go unnoticed by users, but once discovered they can largely be validated by them, to then be used before looking for another solution. The knowledge, (rules in our case), having undergone expert validation, permits dynamic evolution of maintenance processes and thus enables collective updating (from different users and/or platform experts).

The above simulation has shown its feasibility of PETRA. However, these results can be obtained only if the platform is in use for maintenance operations. We thus automate maximum operations, but validation must be carried out by an expert. One of the ways to improve this system is to devise methods to support validation of expert rules.

C- Discussion of PETRA impact:

The economic impact of PETRA, on the other hand, can be measured according to maintenance costs. In fact, there is an estimated cost assigned to the execution of each activity in a process. While services provided by PETRA make shortcuts in the process executions possible by skipping some activities, the global costs of these processes will decrease. In the medium and long terms, the decrease in process costs will have an impact on maintenance outlays, resulting in a remarkable reduction.

Cost variations will follow the curve pattern presented in Figure 17. Our presentation of this pattern is based on the following hypothesis:

H1: The cost of a process execution is the sum of the cost of each activity executed during this process.

H2: the cost of an activity in a process is a mean value.

H3: Rules are validated by senior maintenance experts.

H4: The platform is not in overlearning which can cause problems for taking learned rules into account.

To estimate cost limits, it is assumed that the platform load is at its maximum (all processes are executed in parallel on the platform). We define the boundaries of the maximum and minimum global cost. As shown in Figure 17, the cost curve of the instantiated processes on the platform is located at the lower end of the curve of the estimated minimum cost of process executions.

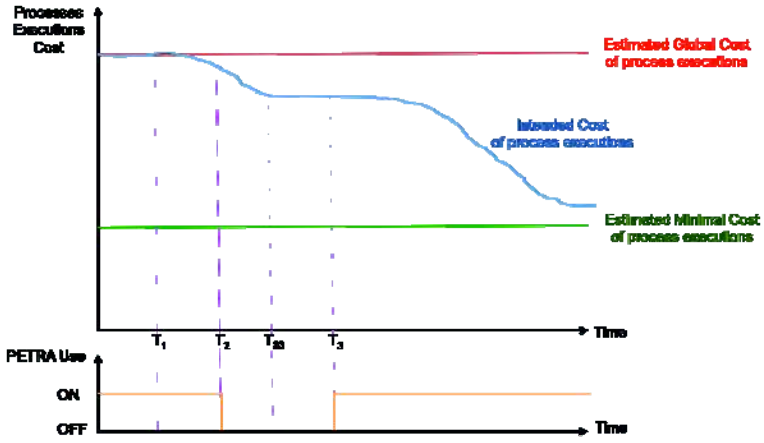


Figure 17 - Allure of cost curve

In this graph, we present a line that shows an estimated time constant for overall cost, which is to say the process executions according to the original pattern without the use of Petra.

A second line shows the minimum cost generated by a process execution if we reduce scripts to a minimum number of operations.

The cost curve while using PETRA can be analyzed as 4 phases:

[0, t1]: Maintenance processes are executed via the s-maintenance platform which includes the features of the PETRA system. T1 presents the moment of generation of new rules by PETRA.

In this phase, the cost curve is superimposed on the line of the estimated global cost of processes.

[T1-T2]: Maintenance processes run on the platform which applies the rules generated by PETRA and which continue to generate new rules. T2 is the moment at which the use of PETRA is on standby. In this phase a lower cost may be recorded due to the application of rules generated by PETRA for skipping activities.

[T2-T3]: In this phase, the platform executes maintenance processes without the use of PETRA services. At point T23 we note cost reduction; it stagnates and remains constant due a halt in rule generation. Also, if the platform is not used, PETRA services cannot be exploited.

[T3- T_i [: In this phase, the platform re-launches PETRA. For a moment the allure remains constant, which is the continuity of the previous phase and even the generation of new rules by PETRA which require time for tracing and learning. When new rules are generated, we can again obtain a decrease in process execution cost.

Nevertheless, in no case can cost reduction be lower than the estimated minimum cost. We can conclude that using PETRA in the maintenance platform, even if not permanently, the cost of process executions may have the allure of integral asymptotics for the estimated minimal and maximum costs.

For this reason and in order to validate this illustration, in the future we plan to integrate the entire system within the s-maintenance platform and to then monitor these processes via the real execution data and not through simulation of use cases. This integration will give greater precision for cost variation and possible gains.

IV. Conclusion

To meet new needs of maintenance actors, we have developed a knowledge-oriented maintenance platform called s-maintenance, providing a dynamic experience feedback approach in order to exploit maintenance process behaviors in real execution. Our intention was to develop a knowledge-management process taking into account the dynamic aspect of knowledge that constitutes a major challenge in knowledge engineering. After a brief review of experience feedback, this paper proposes a trace-based system called "PETRA" which is based on the work on modeled traces by Mille et al. In contrast to major studies of the mining process, our work does not reconstruct the business process. Instead we began with designed maintenance processes on the platform in the aim of including among them experience feedback for different activities.

The aim of this system is to extract knowledge rules from platform activities and user interactions (i.e. the experience of maintenance operators). The collective experience is made explicit via the rules discovered from the repetitive problem solving arising from maintenance activities. This knowledge is specific to the behavior of each maintained system, and it allows users to solve problems without looking for solutions. This knowledge is discovered through the repetitive execution of processes as they advance in the platform, which goes unnoticed by users and proves relevant and interesting for maintenance. In fact, the knowledge allows dynamic evolution of maintenance processes and thus enables general updating.

The system then shares the resulting rules with the knowledge base of the s-maintenance platform so that it can be reused by all users and also by services integrated into the platform.

The architecture of the proposed system is composed of three subsystems: *tracking, learning* and *knowledge capitalization*.

The global process of PETRA follows a KDT (Knowledge Discovery from Traces) process which is a specification of the KDD (Knowledge Data Discovery) process applied to traces.

The subsystem processes are mainly based on the use of the maintenance domain ontology (IMAMO) that we proposed in a previous study, and, more specifically, on the ontological model of maintenance processes. IMAMO is implemented in PowerLOOM, a description language that is integrated within the knowledge base *(KB)* of the s-maintenance platform.

After formalizing traces via relational algebra, we developed a learning method to extract explicit knowledge about activities and their relationships. The learning system exploits the machine-learning algorithm C4.5 which is applied to the modeled traces collected from platform execution traces revealed by the tracking system. Learning results are exploited by the capitalization system to validate the trusted rules and to feed the KB. Processes related to these systems are defined and illustrated by a management process case of conditional maintenance.

A simulation process using jBPM6 was applied to simulate repetitive executions of four maintenance processes, along with a Weka platform to apply the learning algorithm to the trace sets obtained from jBPM execution logs. The results thus obtained enabled maintenance processes to be updated and ensured that the experience return is adapted to user needs. The ontological model allowed us to handle the platform's maintenance processes. This simulation proves the feasibility of our approach.

Our medium-term plan is, (i) rather than to undertake case studies, to conduct a large-scale study of log data from platform executions for full-scale development of the approach, and (ii) to study the conflicting generated rules currently provided to the expert in order to identify whether these contradictions are noise or simply weak signals from noise. The study of these rules over time, and the likelihood of their occurrence will allow us in the future to predict changes in rules within the work context of the platform.

However, to be effective, PETRA requires a substantial period of platform execution in order to collect sufficient data and traces. The PETRA learning phase requires traces on a large scale if it is to obtain trusted rules.

In addition, a more closely defined method is necessary to identify the triggering level of learning. In fact, it is difficult to find the ideal batch of traces from which the learning step can be launched.

Moreover, the collection of traces being dependent on the number of equipment units managed by the platform, as well as on the number of users, PETRA results are useful only in a large-scale context (large equipment parks with many users) where the reuse of experience is massively required.

Also, the configuration of indicator thresholds in the validation phase is based on the experience of human experts, a fact that obliges us to define them empirically. One of the ways to improve the system is to establish methods to support rule validation by maintenance experts in order to minimize their intervention without eliminating it.

These key points constitute the road map for our future research to reinforce PETRA and to increase its efficiency.

Acknowledgement

The authors would like to acknowledge the European Regional Development Fund who funded this work as a part of ALTIDE Project (Aid to Lifecycle Traceability for Intelligently Developed Equipments).

References

Aalst, W. van der, Weijters, A., & Maruster, L. (2004). Workflow Mining: Discovering Process Models from Event Logs. *IEEE Transactions on Knowledge and Data Engineering, 16 (9), 1128–1142.*

Aalst, Wil M. P. van der (2009). Process-Aware Information Systems: Lessons to Be Learned from Process Mining. *Transactions on Petri Nets and Other Models of Concurrency. 2: 1-26.*

Aalst, Wil M. P. van der, Vladimir Rubin, H. M. W. Verbeek, Boudewijn F. van Dongen, Ekkart Kindler, Christian W. Günther. (2010) Process mining: a two-step approach to balance between underfitting and overfitting. *Software and System Modeling 9(1): 87-111*

Abecker, A., A Bernardi, H Maus, M Sintek, C Wenzel (2000). Information supply for business processes: coupling workflow with document analysis and information retrieval. Knowledge-Based Systems, Volume 13, Issue 5, October 2000, Pages 271-284

Agrawal, R., Gunopulos, D., & Leymann, F. (1998). Mining Process Models from Workflow Logs. *In Sixth international conference on extending database technology (pp. 469–483).*

Andrews. J. (1998) Testing using log file analysis: Tools, methods and issues. *ln Proceedings of the 13th Annual International Conference on Automated Software Engineering (ASE'98), Honolulu, Hawaii, October 1998.*

Marcelo de Almeida, M., Raquel Fialho, L. (2013). On the impact of trace-based feature location in the performance of software maintainers, *The Journal of Systems and Software 86 (2013) 1023– 1037*

Bachimont, B. (2004). Pourquoi n'y a-t-il pas d'expérience en ingénierie des connaissances ? *Actes de la conférence « Ingénierie des connaissances (IC2004) », p. 55-64.* Lyon.

Bertino, E. (1992). A view mechanism for object-oriented databases. *International Conference on Extending Database Technologies (EDBT'92). Vienna: Lecture Notes in Computer Science, Springer-Verlag.*

Bouckaert, R. (2010). *WekaManual 3-6-4.*

Bousbia, N., Rebaï, I., Labat, J.M, Balla, A. (2010). Learners' navigation behavior identification based on trace analysis. *User Model. User-Adapt. Interact. 20(5): 455-494 (2010)*

Breen, M. (2005). Experience of using a lightweight formal specification method for a commercial embedded system product line. *Requirements Engineering Journal 10 (2) .*

Champin, P., Prié, Y., & Mille, A. (2003). Musette: Modelling uses and tasks for tracing experience. *In ICCBR (vol. 3,*

pp. 279–286).

Champin, P., Prie, Y., & Mille, A. (2004). Musette : a framework for knowledge capture from experience. *Extraction et Gestion des Connaissances EGC'04.* Clermont Ferrand.

Champin, PA., Cordier, A., Lavoué, E., Lefevre, M. and Mille, A. (2012) Traces, Assistance and Communities, a review Kolflow project - Task 4 - State of the art (D4.1) *Research Report RR-LIRIS-2012-006, LIRIS, (April 2012).*

Champin, PA.,, P. Briggs, M. Coyle, B. Smyth. (2009) Semantics of Social Web Search. *Research report RR-LIRIS-2009-023, submitted to ISWC 2009 - Semantic Web In Use.*

Chalupsky, H., MacGregor, R., & Russ, T. (2010). PowerLOOM manual Powerful knowledge representation and reasoning with delivery in Common-Lisp, Java, and C++ *Version: 1.48 16. University of Southern California.*

Chebel-Morello, B., Karray, M. H. & Zerhouni, N. (2010). New perspectives of Maintenance systems: "towards s-maintenance". *workshop of Sustainable products and production, services and maintenance.* Zurich: IMS 2020.

Chebel-Morello, B., Haouchine, MK., Zerhouni, N. (2013). Reutilization of diagnostic cases by adaptation of knowledge models *Engineering Applications of Artificial Intelligence Volume 26 Issue 10, November, 2013 Pages 2559-2573*

Choquet, C., & Iksal, S. (2007). Modeling Tracks for the Model Driven Reengineering of a TEL System. *JILR, Journal of Interactive earning Research , Special Issue : "Usage Analysis in Learning Systems: Existing Approaches and Scientific Issues", Vol. 18, No. 2.* , 161-184.

Clauzel, D., Sehaba, K., & Prié, Y. (2009). Modelling and visualising traces for reflexivity in synchronous collaborative systems. *In IEEE International Conference on Intelligent Networking and Collaborative Systems, 2009. INCOS'09 (pp. 16–23).*

Clauzel, D., Karim Sehaba, and Yannick Prié. Enhancing synchronous collaboration by using interactive visualisation of modelled traces. *Simulation Modelling Practice and Theory, 19(1):84-97, July 2011. Modeling and Performance Analysis of Networking and Collaborative Systems.*

Jonathan E. Cook and Alexander L. Wolf. (1998) Discovering models of software processes from event-based data. *ACM Transactions on Software Engineering and Methodology, 7(3):215-249, July 1998.*

Cordier, A., Mascret, B., Mille, A. (2009). Extending Case-Based Reasoning with Traces. In Grand Challenges for reasoning from experiences, *Workshop at IJCAI'09, July 2009*

Cordier, A., Mascret, B., Mille, A. (2010) Dynamic Case-Based Reasoning for Contextual reuse of Experience - *Provenance-Awareness in Case-Based Reasoning Workshop. ICCBR 2010* Alexandria, Italy

Cram, D., Jouvin, D., & Mille, A. (2007). Visualisation interactive de traces et réflexivité : application à l'EIAH collaboratif synchrone eMédiathèque. *Journale des Sciences et Technologies de l'Information et de la Communication pour l'Éducation et la Formation. Numéro spécial "Analyses des traces d'utilisation dans les EIAH"* .

Craw, S. (2009). Agile case-based reasoning: A grand challenge towards opportunistic reasoning from experiences. *In Proceedings of the IJCAI-09 Workshop on Grand Challenges in Reasoning from Experiences*, pp. 33-39, Pasadena, CA, 2009.

J. Endrenyi, S. Aboresheid, R.N.Allan et al. (2001). The present status of maintenance strategies and the impact of maintenance on reliability. *IEEE Trans. on Pwr Sys, Vol. 16, No. 4, Nov., 2001, pp 638 –646.*

Ermine JL. (2000) Challenges and approaches for knowledge management. *Proceedings of the Conference on Principles and Practice of Knowledge Discovery in Databases.*

Fayyad, U.; Piatetsky-Shapiro, G.; Smith, P. & Uthurusamy, R. (1996) *Advances in Knowledge Discovery and Data Mining*. AAAI/MIT Press, MA

Frawley, W.J., Piatetsky-Shapiro, G., Matheus, C.J.(1992) Knowledge discovery in databases: An overview. *AI magazine 13(3), 57{70 (1992)*

Gaber, MM., Zaslavsky, A. and Krishnaswamy, S. (2005). Mining data streams: a review. ACM Sigmod Record, 34(2):18–26, 2005.

Georgeon, O., Mille, A., Bellet, BT., Mathern, B., Ritter, F. (2012) Supporting activity modelling from activity traces. *Expert Systems 29(3):261-275. 2012.*

Hilbert, D. M., & Redmiles, D. F. (2000). Extracting usability information from user interface events . *ACM Computing Surveys, Vol. 32, No. 4.* , 384-421.

Imieliński, T.; Lipski, W. (1984). "The relational model of data and cylindric algebras". *Journal of Computer and System Sciences 28: 80–102*.

Jabrouni, Hicham and Kamsu Foguem, Bernard and Geneste, Laurent (2011) Continuous Improvement Through Knowledge-Guided Analysis in Experience Feedback. *Engineering Applications of Artificial Intelligence, 24 (8). pp. 1419-1431. ISSN 0952-1976*

Jardine, A. K., Daming, L., & Banjevic, D. (2006). review on machinery diagnostics and prognostics implementing condition-based maintenance. *Mechanical Systems and Signal Processing , 1483-1510*.

P. JERMANN, A. SOLLER, M. MUEHLENBROCK, From Mirroring to Guiding: A Review State of the Art Technology for Supporting Collaborative Learning. In P. Dillenbourg, A. Eurelings & K. Hakkarainen (Eds.) *EuroCSCL 2001 Proceedings: European Perspectives on Computer-supported Collaborative Learning*, (2001), Maastricht: Maastricht McLuhan Institute, Netherlands, pp. 324–331.

Kamsu Foguem, B., Coudert, T., Geneste, L., Béler, C. *(2008*) Knowledge formalization in experience feedback processes : an ontology-based approach; Knowledge formalization in experience feedback processes : an ontology-based approach. *(2008) Computers in Industry, vol. 5 (n° 7). pp. 694-710. ISSN 0166-3615*

Karray, M. H., Chebel-Morello, B., & Zerhouni, N. (2009). Toward a maintenance semantic architecture. *Proceedings of the Fourth World Congress on Engineering Asset Management (WCEAM)* (pp. 98-111). Athens: Springer-Verlag London .

Karray M.H., Chebel-Morello B., Zerhouni N., (2010). A contextual semantic mediator for a distributed cooperative maintenance platform. *In the 8th IEEE International Conference on Industrial Informatics, INDIN'10., Japon (2010)*

Karray M.H., Chebel-Morello B., Lang C., Zerhouni N. (2011). A component based system for S-maintenance. *In the 9th IEEE International Conference on Industrial Informatics, INDIN'11., Portugal (2011)*

Karray M.H., Chebel-Morello B., Zerhouni (2012). N, A Formal Ontology for Semantics in Maintenance Platforms. *Applied Ontology Journal 7 (3) 2012, pp 269-310.*

Kohavi, R., & Quinlan, J. R. (2002). Decision-tree discovery. In Will Klosgen and Jan M. Zytkow, *Handbook of Data Mining and Knowledge Discovery* (pp. 267-276). Oxford University Press.

Lafifi, Y., Gouasmi, N., Halimi, K., Herkas, W., Salhi, N., & Ghodbani, A. (2010). Trace- based collaborative learning system. *Journal of Computing and Information Technology, 18.*

Laflaquière, J., Settouti L.S.,Prié, Y., Mille, A. (2006). Trace-Based Framework for Experience Management and Engineering. *In Knowledge-Based IntelligentInformation and Engineering Systems KES (1) 2006: 1171-*

Laflaquière, J., Alain Mille , Magali Ollagnier-Beldame , Yannick Prié,. (2010) Modeled traces for systems allowing reflection on personal experience, *Submitted to International Journal of Human-Computer Studies, January 2010, 12 p.*

Lee, J., & Wang, H. (2008). New Technologies for Maintenance. Dans K. A. Kobbacy, & D. N. Murthy, *Complex System Maintenance Handbook* (pp. 49-78). Springer Series in Reliability Engineering.

Lejarraga T. and Gonzalez C. (2011) Effects of feedback and complexity on repeated decisions from description; *Organizational Behavior and Human Decision Processes, 2011, vol. 116, issue 2, pages 286-295*

Qiang Li , Marie-Hélène Abel Jean-Paul A. Barthès (2013). Modeling and exploiting collaborative traces in web-based collaborative working environment. Computers in Human Behavior xxx (2013)

Lowe, G., & Shirinzadeh, B. (2005). A Knowledge-base Self-Learning System. *Proceeding (483) ACIT - Automation, Control, and Applications.*

Mathern, B., Mille, A., Cordier, A., Cram, D., Zarka, R., (2012) Towards a Knowledge-Intensive and Interactive Knowledge Discovery Cycle. In *20th ICCBR Workshop Proceedings*, Luc Lamontagne, Juan A. Recio-Garcia ed. Lyon, France. pp. 151-162. 2012.

Matta N. Ermine J.L., Aubertin G., Trivin J.Y., (2001°) Knowledge Capitalization with a knowledge engineering approach : the MASK method. *Proceedings of IJCAI'2001 Workshop on Knowledge Management and Organizational Memory,* August 2001.

Mille, A. (2006). From case-based reasoning to trace-based reasoning. *Annual Reviews in Control, 30(2):223–232, October 2006. Journal of IFAC*

Mitchell T. M., (1997) Machine Learning. *McGraw-Hill.*

Nilsson, J. (1998). *Introduction to Machine Learning: An Early Draft of a Proposed Textbook.* http://robotics.stanford.edu/people/nilsson/mlbook.html.

Nonaka, I., Takeuchi, H., (1995). The knowledge-Creating Company: How Japanese Companies Create the Dynamics of Innovation. *Oxford University Press*.

J. P. PERNIN, CSE, un modele de traitement de traces. *(2005) Research report of CLIPS-IMAG.*

Piatetsky-Shapiro, G. & Frawley, W. (1991) *Knowledge Discovery in Databases*. AAAI/ MIT Press, MA.

Piatetsky-Shapiro, G. (1994). *An overview of knowledge discovery in databases: Recent progress and challenges. Rough Sets, Fuzzy Sets and Knowledge Discovery*, pages 1–11.

Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning.* San Francisco: Morgan Kaufmann.

Quinlan, J. R. (1990). Induction of Decision Trees. In J. Shavlik, & T. Dietterich, *Readings in Machine Learning* (pp. 57-69). San Francisco: Morgan Kaufmann.

Rech, J., Ras, E., Decker, B. (2007). Intelligent assistance in german software development: A survey. *IEEE Softw., 24:72–79, July 2007.*

Rozinat, A., van der Aalst, W.M.P.: Decision mining in ProM. *In: Dustdar, S., Faideiro, J.L., Sheth, A. (eds.) International Confer-ence on Business Process Management (BPM 2006). Lecture Notes in Computer Science, vol. 4102, pp. 420–425. Springer, Berlin (2006)*

Sehaba, K. (2011). Adaptation of shared traces in e-learning environment. (2011) *11th IEEE international conference on advanced learning technologies (ICALT) (pp. 103–104).*

Settouti, L.-S., Prié, Y., Marty, J.-C., & Mille, A. (2007). *Vers des Systèmes à Base de Traces modélisées pour les EIAH.* Lyon: rapport de recherche, LIRIS-RR-2007-016.

Settouti, L., Prie, Y., Marty, J., & Mille, A. (2009). A trace-based system for technology-enhanced learning systems personalisation. *In Ninth IEEE international conference on advanced learning technologies, ICALT 2009 (pp. 93–97).*
worldofcomputing. (2011). *Machine-learning overview*. From intelligence.worldofcomputing.net: //intelligence.worldofcomputing.net/machine-learning/machine-learning-overview.html

Xu, C., Xu, Z., Xiao, P., Zhou, Z., Liu, S., & Jiang, Z. (1995). A self-learning system and its application in fault diagnosis. and Measurement Technology Conference*, IMTC/95. Proceedings of "IEEE Integrating Intelligent Instrumentation and Control".*

Walker K. (2006) A method for creating collaborative mobile learning trails. *Convergence Workshop, Intersecting and integrating collaborative-mobile-inquiry learning*. 4–6 December 2006, Amsterdam.

Weber, R.; Aha, DW.; Branting, L. K.; Lucas, J. R; Fernandez, I. (1999). Active case-based reasoning for lessons delivery systems;; *AAAI Press; The 13th International Florida Artificial Intelligence Research Society Conference, FLAIRS 1999, Menlo Park, FL: pp. 170-174.*

Yee Fan Tang, S., Wai Kwan Chow A. (2007). Communicating feedback in teaching practice supervision in a learning-oriented field experience assessment framework *Teaching and Teacher Education; Volume 23, Issue 7, October 2007, Pages 1066–1085*

Zarka, R., Cordier, A., Egyed-Zsigmond, E., & Mille, A. (2011). Trace replay with change propagation impact in client/server applications. *In Conference AFIA*.