Unsupervised Discovery of Intentional Process Models from Event Logs

Ghazaleh Khodabandelou, Charlotte Hug, Rebecca Deneckère, Camille Salinesi
Sorbonne University, Centre de Recherche en Informatique
Paris, 75013, France
ghazaleh.khodabandelou@malix.univ-paris1.fr,
charlotte.hug, rebecca.deneckere, camille.salinesi@univ-paris1.fr

ABSTRACT

Research on guidance and method engineering has highlighted that many method engineering issues, such as lack of flexibility or adaptation, are solved more effectively when intentions are explicitly specified. However, software engineering process models are most often described in terms of sequences of activities. This paper presents a novel approach, so-called Map Miner Method (MMM), designed to automate the construction of intentional process models from process logs. To do so, MMM uses Hidden Markov Models to model users' activities logs in terms of users' strategies. MMM also infers users' intentions and constructs fine-grained and coarse-grained intentional process models with respect to the Map metamodel syntax (*i.e.*, metamodel that specifies intentions and strategies of process actors). These models are obtained by optimizing a new precision-fitness metric. The result is a software engineering method process specification aligned with state of the art of method engineering approaches. As a case study, the MMM is used to mine the intentional process associated to the Eclipse platform usage. Observations show that the obtained intentional process model offers a new understanding of software processes, and could readily be used for recommender systems.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Workflow Management; H.1.2 [**Information Systems**]: Models and Principle

General Terms

Theory, Algorithms, Management

Keywords

Software Development Process, Intentional Process Modeling, Event logs Mining

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.

MSR'14, May 31 – June 1, 2014, Hyderabad, India
Copyright 2014 ACM 978-1-4503-2863-0/14/05...$15.00
http://dx.doi.org/10.1145/2597073.2597101

1. INTRODUCTION

Fueled by the growing presence of events logs in software and software engineering platforms, process mining has emerged a few years ago as a key approach to design software processes [29, 30]. Mining software engineering processes from logs can be useful for understanding how people really work, analyzing how actual software engineering processes differ from the prescribed ones, and thereby improve software engineering methods and products. Whereas most process mining approaches specify behaviors in terms of sequences of tasks and branching, we believe that the fundamental nature of processes is mostly intentional. Intention-oriented software process specification has been promoted since long ago. In the 80s, intentional models were proposed in the IS community as a *potential theoretical foundation* to determine users' behavior [9] or in a former work to model humans behavioral intention, particularly for computer usage behavior [1]. Later on, in the early 90s, intention analysis and modeling have been promoted as a driving paradigm to study strategic alignment, to define actors and roles [32], to guide requirements elicitation [23], to handle traceability [15], to study users behavior to identify and name use cases, etc. Representing processes by intentions and strategies has proved a powerful tool to better understand the deep nature of processes, to see how processes interweave and combine, to abstract processes and visualize them under man-manageable form, even when they are extremely complex [25].

This paper presents a method, so-called Map Miner Method (MMM) that aims at discovering users' intentions and strategies to construct intentional process models from event logs. MMM generates intentional models specified with the Map notation [24, 25]. This notation is chosen rather than other intentional process models such as KAOS or I* [32, 8] because (a) it has already proved effective for specifying software engineering processes [23], (b) it supports process variability and multi-process specification [24], and (c) its way of combining intentions and strategies at multiple levels of abstraction scales well to large and complex processes [25]. In MMM framework, an intention is defined as a goal, an objective or a motivation to achieve a goal with a clear-cut criteria of satisfaction, which can be fulfilled by the enactment of a process [27]. The intentions are explicitly represented and are of high-level of abstraction (*e.g.*, organizational goals).

In software engineering context, MMM can be useful at different stages of the process model life-cycle, for instance: (i) at the requirements level, to understand users intentions from software usage activity logs; (ii) at the project manage-

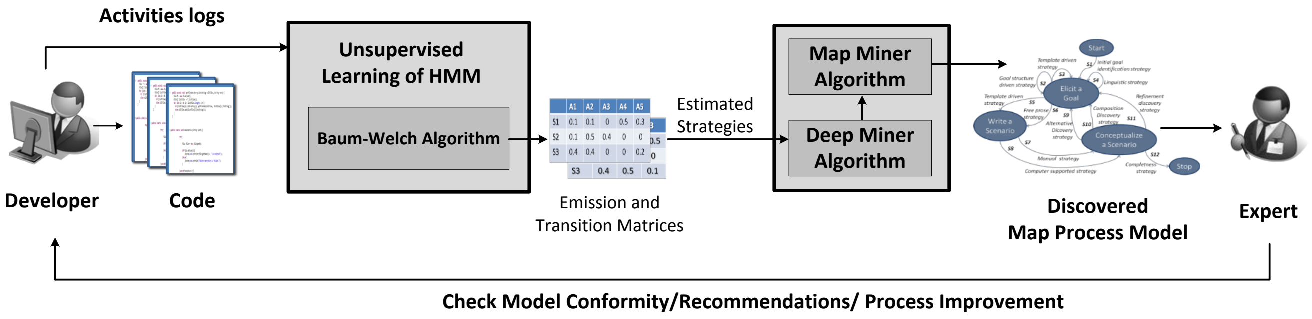
Activities logs
Developer
Code
Unsupervised Learning of HMM
Baum-Welch Algorithm
A1 A2 A3 A4 A5
S1 0.1 0.1 0 0.5 0.3
S2 0.1 0.5 0.4 0 0
S3 0.4 0.4 0 0 0.2
S3 0.4 0.5 0.1
Emission and Transition Matrices
Estimated Strategies
Map Miner Algorithm
Deep Miner Algorithm
Start
Elicit a Goal
Write a Scenario
Conceptualize a Scenario
Stop
S1 Initial goal identification strategy
S2 Goal structure driven strategy
S3 Template driven strategy
S4 Linguistic strategy
S5 Template driven strategy
S6 Free prose strategy
S7 Manual strategy
S8 Computer supported strategy
S9 Alternative Dicovery strategy
S10 Composition Discovery strategy
S11 Refinement discovery strategy
S12 Completness strategy
Discovered Map Process Model
Expert
Check Model Conformity/Recommendations/ Process Improvement

Figure 1: An overview of the Map Miner Method

ment level, to check alignment between prescribed process models and what stakeholders actually do; or (iii) at the application level, to monitor users and provide run-time recommendations.

The contribution made in this paper is triple: (i) First, modeling the users' activities logs in terms of strategies (using Hidden Markov Models (HMMs) [22]); unsupervised learning is used to estimate the parameters of HMM, (ii) Second, using estimated strategies and Map formalism, we develop Deep Miner and Map Miner algorithms to generate respectively, fine-grained and coarse-grained Map process model, (iii) Finally, the entire proposed approach is applied on two practical datasets; first in a laboratory context, second in a large-scale case study by mining the **Eclipse UDC** (Usage Data Collector) developers' logs [10]. The resulting Maps provided a precious understanding of the processes followed by the students and the developers, and some feedback on the effectiveness and scalability of MMM. Note that, although the application of HMMs on software logs is not new, its application to model the multi-level topology of humans' cognitive process (intentions ans strategies) is suggested for the first time in our previous works [16, 17, 18] and improved and developed in this paper.

The remainder of this paper is organized as follows. Section 2 introduces MMM, which consists in modeling users' activities in terms of strategies (using HMMs). This Section also describes Deep Miner and Map Miner algorithms. Section 3 describes the application of MMM on a dataset resulting from the event logs of Entity/Relationship diagrams creation. Section 4 represents the large-scale case study with event logs from Eclipse UDC to find out how developers use the *Eclipse technology*. Related works are discussed in Section 5 and threats to validity in Section 6. Finally, Section 7 concludes this work and presents its perspectives.

2. MAP MINER METHOD

The key idea of MMM is modeling the users' behaviors in terms of their underlying intentions and strategies, from event logs. The strategies are the alternative ways to fulfill the intentions. MMM uses the Map metamodel to model actual process followed by users. According to the Map meta model, **intentions** express what users intend to perform during the enactment of a process [24]. In other words, the enactment of a process is the execution of a sequence of **activities** that are caused by users' intention. According to the fuzzy mechanism of a cognitive process, several intentions cause the performance of several activities at the same time. However, the intentions are not directly related to the activities; the intentions can be fulfilled by combining different activities, which are different ways of achieving an intention, *i.e.*, strategies. Indeed, in the Map metamodel, strategies are used to move from one intention to another and are made of one or several activities. This relationship between the intentions, the strategies and the activities, represents the top-down structure of reasoning and acting in cognitive processes of human brain. Nevertheless, only the low-level part of this structure, *i.e.*, users' activities, is observable. The middle and high-level part (strategies and intentions) are abstract notions and therefore unobservable directly. MMM proposes to trace back this structure to achieve the sources, which are users' intentions. It is important to define some notions used in this paper. The notion of *intention* is defined in the introduction. However, there are also sub-intentions, which are the low-level intentions. They are the finest intentional objects of the Map metamodel. Each sub-intention is associated to a parent intention, and one intention is fulfilled if at least one of its children sub-intention is fulfilled. The sub-intentions allow obtaining the fine-grained Map and the intentions allow obtaining the coarse-grained Map.

MMM uses the observable users' activities traces (a set of event logs), generated while interacting with Information Systems. MMM consists of three phases: (i) First phase: estimating the users' strategies from observed activities using HMMs. The topology of HMM permits modeling observed process (users' activities) in terms of hidden states (underlying users' strategies), (ii) Second phase: generating fine-grained Map process model using estimated strategies along with Map formalism (Deep Miner algorithm), (iii) Final phase: generating coarse-grained Map process model from fine-grained one (Map Miner algorithm). Constructing the Map process model thereby allows rebuilding the actual process model, *i.e.*, the model followed by users. Figure 1 depicts an overview of MMM.

Before addressing the problem of discovering a Map, the associated framework is briefly explained in the following Section.

2.1 Map Metamodel

Map is an intentional process metamodel. Map process model (an instance of Map metamodel) allows representing process models in terms of users' intentions and strategies. Figure 2 illustrates a Map process model where the nodes represent the intentions and the edges represent the strategies. A set of <*Source Intention, Strategy, Target Intention*> represents a *section* in the Map. Map allows representing flexible process models, enacted in a dynamic way since the sections of a Map can be executed non-sequentially and as long as intentions are not completely fulfilled. For example, on Figure 2, one way to fulfill the intention *Specify an entity* is to select the strategy S_4: *By generalization* which is composed of two activities: create an entity and create generalization link (see Table 1). Thus, confronted to a specific situation and a particular intention, the Map reveals the alternative strategies to fulfill the intention. The corresponding section of this example is <*Specify an entity, By generalization, Specify an entity*>.

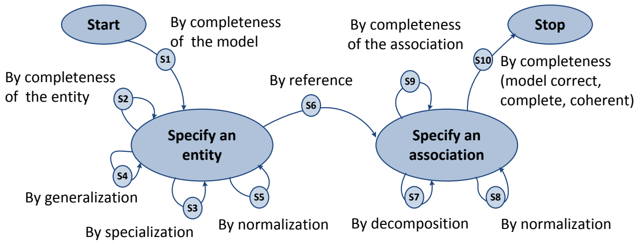

Figure 2: Map process model for the construction of E/R diagrams.

2.2 Estimating Strategies from Activities

HMMs allow modeling the structure of complex temporal dependencies between two complementary Markov processes: hidden and observed processes, given that the observed process is generated depending on the state of the hidden process. Generally, the states of hidden process are not visible but the probability of a given state can be inferred by computing the Maximum Likelihood of the observed process. The two complementary processes are defined next.

2.2.1 *Hidden Process: Users' Strategies*

Let $\mathbf{s} = (s_1, \ldots, s_L) \in \mathcal{S}^L$ be a temporal sequence of users' strategies of length L. The topology M_1 is chosen for hidden process, which means that the strategy s_l at step l only depends on the strategy at step $\ell - 1$. This choice is justified by the fact that strategies are performed in a logical order by users. A homogeneous Markov chain, which parameters are denoted by $\mathbf{T}$ and π, models the hidden process (strategies) with:

$$\begin{aligned} \mathbf{T}(u, v) &= \Pr(s_\ell = v | s_{\ell-1} = u) \ \forall u, v \in \mathcal{S},\ \ell \in [2, L], \\ \pi(u) &= \Pr(s_1 = u) \ \forall u \in \mathcal{S}. \end{aligned} \tag{1}$$

The vector π contains the probabilities of strategy at the initial state and the matrix $\mathbf{T}$ contains the transition probabilities for the following strategies, *i.e.*, the transition probabilities from any strategy at step $\ell - 1$ to any other strategy at step ℓ (including itself).

2.2.2 *Observed Process: Users' Activities*

Let $\mathbf{a} = (a_1, \ldots, a_L) \in \mathcal{A}^L$ be a temporal sequence of users' activities of length L. We choose the model M_0 for observed process, meaning that the emission of a_ℓ, at a given step ℓ, does not depend on any previous observation. It only depends on the hidden strategy at the same time step. The emission probability of an observation $a \in \mathcal{A}$ for a given strategy $u \in \mathcal{S}$ is given by:

$$\mathbf{E}(a, u) = \Pr(a|u). \tag{2}$$

The matrix $\mathbf{E}$ contains the emission probabilities of any activity for any strategy.

2.2.3 *Learning HMMs Parameters*

Assuming that $\mathcal{S}$, $\mathcal{A}$ and π are known, the HMM model parameters is fully described by $\mathcal{H} = \{\mathbf{E}, \mathbf{T}\}$, which represents the core information about the HMM behavior. The MMM highly relies on the emission matrix $\mathbf{E}$ and the transition matrix $\mathbf{T}$ to respectively characterize the users' strategies and the structure of the Map. For these reasons, it is extremely important that these two matrices precisely match the process under study. To do so, the coefficients of the two matrices are learned based on logs produced by the enactment of the process. This paper focuses on unsupervised learning to estimate $\mathbf{E}$ and $\mathbf{T}$ since it has been proven to be efficient and offers a higher performance than supervised learning of HMM [22]. Unsupervised learning estimates these parameters based only on traces of activities (without prior knowledge about the strategies set $\mathcal{S}$). The Baum-Welch Algorithm (BWA) [3] is most commonly used to learn the parameters of a HMM. The BWA estimates the HMM parameters that locally maximize the probability of having these sequences generated by the HMM. More precisely, the BWA maximizes the likelihood of $\mathcal{H}$:

$$\mathcal{H}^* = \arg\max_{\mathcal{H}} \prod_{n=1}^{N} \Pr(\mathbf{a}_n|\mathcal{H}), \tag{3}$$

where N is number of observed sequences of activities.

It is important to highlight the BWA requires the cardinals of $N = |\mathcal{S}|$ and $M = |\mathcal{A}|$ to run. Regarding the set of activities $\mathcal{A}$ can simply be obtained by identifying the different activities in the dataset. However, the information about number of strategies does not exist in the dataset. Therefore, a heuristic method is used in this paper, which consists in generating several HMMs models with different numbers of strategies and observing the associated emission matrices $\mathbf{E}$. It occurs that as the number of possible strategies increases, the number of different strategies obtained in the emission matrices reaches a threshold. It means that when the number of possible strategies is too high, the BWA produces an emission matrix with several identical strategies. Consequently, this observed threshold sets the right number of strategies.

2.3 Deep Miner Algorithm

Once model parameters $\mathbf{T}$ and $\mathbf{E}$ are estimated by the BWA, the problem is how to extract a Map process model which fits actual process model. As a first step into this direction, we propose a metric which has the interesting property of taking into account both fitness and precision to optimize the Map process model.

2.3.1 *Extracting a Fine-grained Map*

We recall that the matrices generated by the BWA are an emission matrix $\mathbf{E}$, giving the probabilities of generating any activity while performing a strategy and a transition matrix $\mathbf{T}$, giving the probabilities of transition between any couple of strategies $(s, s') \in \mathcal{S}^2$.

Clearly, there is a strong link between the transition matrix and the topology of the Map process model we want to extract. To extract a Map from a transition matrix, the two following constraints must be verified: (i) any transition between possible strategies in the transition matrix should be possible on the Map; (ii) any transition between possible strategies in the Map should be possible in the transition matrix.

The first constraint can be seen as a criterion for fitness since it ensures that all the transitions learned from the dataset are present in the Map. The second constraint corresponds to a criterion of precision since it aims at avoiding introducing extra-transitions in the Map that are not learned from the dataset. Our goal is to find the Map that best satisfies both of them. In the next part, we define a metric which is a trade-off between fitness and precision and also captures the relative importance of transitions.

2.3.2 Proposed Metric of Fitness and Precision

The topology of a Map $\mathbf{m}$ can be defined by the set of its sections, each comprising a source sub-intention, a strategy and a target sub-intention. We formally write

$$\mathbf{m} = (m_k)_{k \in \{1,\ldots,K\}}, \tag{4}$$

where k denotes the index of a section and K is the total number of sections of the Map. For each $k \in \{1, \ldots, K\}$, $m_k = (i, s, j) \in \mathcal{I} \times \mathcal{S} \times \mathcal{I}$. The component $m_k(1)$ is the source sub-intention of section k, $m_k(2)$ is the strategy of section k, and $m_k(3)$ is the target sub-intention. On the Map $\mathbf{m}$, a transition from strategy s to strategy s' is possible if and only if there exist $(k, k') \in \{1, \ldots, K\}^2$ such that $m_k(2) = s$, $m_{k'}(2) = s'$, and $m_k(3) = m_{k'}(1)$. In the following, we use the symbol α to denote if a transition is possible or not in the Map:

$$\alpha_{s,s'} = \begin{cases} 1 \text{ if } \exists (k, k') \in \{1, \ldots, K\}^2 \text{ such that } m_k(2) = s \\ m_{k'}(2) = s', \text{ and } m_k(3) = m_{k'}(1), \\ 0 \text{ otherwise.} \end{cases} \tag{5}$$

In the transition matrix $\mathbf{T}$, we only consider as valid transitions with a probability above a given threshold ε. The value of ε has to be chosen heuristically, to counter the effects of noise and artifacts in the dataset. We define:

$$\omega_{s,s'} = \begin{cases} 1 \text{ if } \mathbf{T}(s, s') \geq \varepsilon, \\ 0 \text{ if } \mathbf{T}(s, s') < \varepsilon. \end{cases} \tag{6}$$

Classically, the criteria of fitness and precision between $\mathbf{T}$ and $\mathbf{m}$ can be expressed by the expressions known as recall and precision. In our context, we define these two expressions as

$$\mathrm{Rec}(\mathbf{T}, \mathbf{m}) = \frac{\sum_{s,s'} \omega_{s,s'} \alpha_{s,s'}}{\sum_{s,s'} \omega_{s,s'}}, \tag{7}$$

$$\mathrm{Pre}(\mathbf{T}, \mathbf{m}) = \frac{\sum_{s,s'} \omega_{s,s'} \alpha_{s,s'}}{\sum_{s,s'} \alpha_{s,s'}}. \tag{8}$$

The numerator of both expressions is the number of significant transitions in $\mathbf{T}$ that are present on the map $\mathbf{m}$, while the denominators are the number of significant transitions in $\mathbf{T}$ and the number of transitions on $\mathbf{m}$, respectively.

Since our goal is to find a map that fits best the transition matrix with respect to both recall and precision, we can use the classical F-measure which expression is:

$$F_1(\mathbf{T}, \mathbf{m}) = 2 \frac{\mathrm{Pre}(\mathbf{T}, \mathbf{m}) \mathrm{Rec}(\mathbf{T}, \mathbf{m})}{\mathrm{Pre}(\mathbf{T}, \mathbf{m}) + \mathrm{Rec}(\mathbf{T}, \mathbf{m})}. \tag{9}$$

2.3.3 Optimization Problem

Now that the proper metric has been defined, we need to find the Map that maximizes it. The solution of this problem belongs to the set

$$\mathcal{M} = \arg\max_{\mathbf{m}} F_1(\mathbf{T}, \mathbf{m}). \tag{10}$$

Since we want to obtain a Map with the simplest structure, we choose the solution with the lowest number of sections. In other words, the solution is

$$\mathbf{m}^* = \arg_{\mathbf{m} \in \mathcal{M}} \min |\mathbf{m}|, \tag{11}$$

where $|\mathbf{m}|$ stands for the number of sections in $\mathbf{m}$. However, finding $\mathbf{m}^*$ is a difficult task since $\mathbf{m}$ generally belongs to a high-dimension space. Indeed, it can be shown that there are $2^{|\mathcal{S}|^2}$ possible Maps for $|\mathcal{S}|$ different strategies. Consequently, computing all the possible Maps with a brute force method then comparing their F-measures is not an option. Instead, we developed an algorithm that solves (11) with a complexity bounded by $|\mathcal{S}| * (|\mathcal{S}| - 1)$. This algorithm is detailed below.

```
Data: strategy set $\mathcal{S}$, transition matrix $\mathbf{T}$, threshold $\varepsilon$
Result: map $\mathbf{m}^*$
for each strategy $s \in \mathcal{S}$ do
    associate to $s$ a target sub-intention $i_s$;
end
for each strategy $s \in \mathcal{S}$ do
    for each strategy $s' \in \mathcal{S}$, $s' \neq s$ do
        if $\mathbf{T}(s, s') \geq \varepsilon$ then
            create a section from $i_s$ to $i_{s'}$ with strategy $s'$
            ;
        end
    end
end
```

Algorithm 1: How to obtain a Map from $\mathcal{S}$, $\mathbf{T}$, and ε.

The first part of algorithm 1 associates a target sub-intention to each strategy of $\mathcal{S}$. In the second part, if a transition probability from strategy s to strategy s' is above the threshold ε, a section is added to the Map from the target sub-intention of s to the target sub-intention of s'. This section ensures that the transition given by $\mathbf{T}$ is also present in the Map. With this algorithm, recall and precision, defined in (7) and (8), have the advantage of being equal to 1. Indeed, ε defines the accuracy level of the Map. When ε is close to 0, almost all the transitions from the unsupervised model are present in the obtained Map. Consequently, the likelihood of the obtained Map is high but the Map is hardly understandable by humans since it has too many sections. However when ε increases, the number of sections, as well as the likelihood of the obtained Map, decrease. The Map gets more easily understandable by humans but it is not as accurate in terms of transition.

2.4 Map Miner Algorithm

Deep Miner algorithm generates the low-level intentions or *sub-intentions* shown by $\{a, b, c, \cdots\}$ (Figure 3). To extract high-level intentions from low-level intentions, Map Miner algorithm has been developed to automatically perform this task. To do so, Map Miner algorithm uses a clustering algorithm, *K-means* [13] to group the sub-intentions into the intentions. Indeed, this multi-level topology is due to the deep architecture of the brain. The extensive studies on the visual cortex show cognitive processes have a deep structure and humans organize their ideas and concepts hierarchically. First, they learn simpler concepts and then compose them to represent more abstract ones [4]. Note that the number of intentions is a parameter that has to be chosen. The choice of these parameters allows researchers to obtain Maps with different level of precision. Due to lack of space, this algorithm will be detailed in a future work.

3. A PROOF OF CONCEPT

To evaluate MMM, we conducted a tailored experiment with the Master students in computer science of Sorbonne University. According to the experiment, 66 students created Entity/Relationship (E/R) diagrams using the prescribed Map process model (Figure 2). Indeed, for this particular experiment, the students were asked to follow the prescribed Map while creating their E/R diagrams. Our goal here is to compare the actual Map process model discovered by MMM to the prescribed Map. Regarding this prescribed Map, students could select ten strategies to fulfill four intentions, *i.e.*, **Start**, **Specify an entity**, **Specify an association** and **Stop**. From **Start**, it is possible to progress in the process by selecting strategies leading to intentions but once the intention of **Stop** is achieved, the enactment of the process finishes. Table 1 represents the activities and related strategies that the students performed to fulfill the four intentions. The event logs comprising students' activities were recorded by a web-based tool. Once the event logs were available, MMM has been applied on them to trace back this procedure and rebuild the Map process model that was actually followed. Finally, the conformity between the prescribed Map and the discovered Map is checked (see Section 3.1). This knowledge can be useful to (i) improve the software used by the students to make the prescribed model easier to follow, (ii) modify the prescribed model by accounting to the process the students actually followed.

3.1 Applying MMM on Dataset

Once the students' traces are recorded, MMM is applied to obtain the fine-grained and coarse-grained Map process models. As a first step, the number of strategies must be fixed by the heuristic method described in Section 2.2. As a result, we obtain 10 as number of strategies. Then BWA estimates the parameters of the transition matrix and the algorithm described in 2.3.3 is then applied to obtain the structure of a fine-grained Map, *i.e.*, what is the source sub-intention and the target sub-intention of each strategy (the sections). As described earlier, the value of ε has to be set to obtain a trade-off between the accuracy of the Map and its understanding. Figure 3 depicts the fine-grained Map (small nodes) for $\varepsilon = 0.07$ and coarse-grained Map (large nodes) obtained by Deep miner and Map Miner algorithms, respectively.

Table 1: Strategies and related activities

Labels	**Name of strategies**	**Related activities (activities labels)**
S_1	By completeness (model)	Create entity (a_1)
S_2	By completeness (entity)	Link attribute to entity (a_2)
S_3	By normalization	Delete Link attribute to entity (a_6), Delete entity (a_5), Define primary key (a_7)
S_4	By generalization	Create entity (a_1), Create generalization link (a_3)
S_5	By specialization	Create entity (a_1), Create specialization link (a_4)
S_6	By reference	{Delete link attribute to entity, Create entity, Create association, Link association to entity} (a_8), {Create association, Link association to entity} (a_9)
S_7	By decomposition	{Create association, Link association to entity} (a_9)
S_8	By normalization	{Delete association, Delete Link attribute to association}(a_{10})
S_9	By completeness (assoc.)	Link attribute to association (a_{11})
S_{10}	By completeness (final)	Check the model (a_{12})

What we learn from the obtained Map is:

• Deep Miner algorithm found 9 sub-intentions as well as intentions **Start** and **Stop**. Map Miner algorithm groups the sub-intentions into the high-level intentions to obtain coarse-grained Map. The high-level intention are shown by I'_2 and I'_3 (large circle on Figure 3). The discovered strategies for each high-level intention allow identifying the intentions I'_2 to **Specify an entity** and I'_3 to **Specify an association**.

• There is a number of matches between the new groups of strategies in the obtained Map and the groups of strategies in the prescribed Map. For instance, S_1 and S'_1, S_2 and S'_1, S_{10} and S'_{10} are made of similar activities. The few mismatches between prescribed and obtained strategies indicate that some prescribed strategies were not followed as intended by the students. This shows how the students preferred to behave. For instance, in the prescribed Map, activity a_2 is only present in strategy S_2 for intention I_2 whereas in the obtained Map, it is present in many strategies: S'_1, S'_6, S'_7, and S'_9 for intentions I'_2, and I'_3. The same phenomenon is true for activity a_1. It means that these particular activities are used by students during the entire process and not only for a single intention.

It is important to highlight that the likelihood (3) of the HMM model is $-2.36e^3$ (logarithmic scale), which is higher than the likelihood of the obtained Map ($-2.54e^3$). This is normal since the Map formalism restricts some transitions (*e.g.*, from the Stop to the Start) that are generated by the HMM model.

4. CASE STUDY: ECLIPSE UDC

The goal of this case study is to reconstruct the Map process model of developers who committed their code to Eclipse UDC server hosted by *Eclipse Foundation* [10]. The

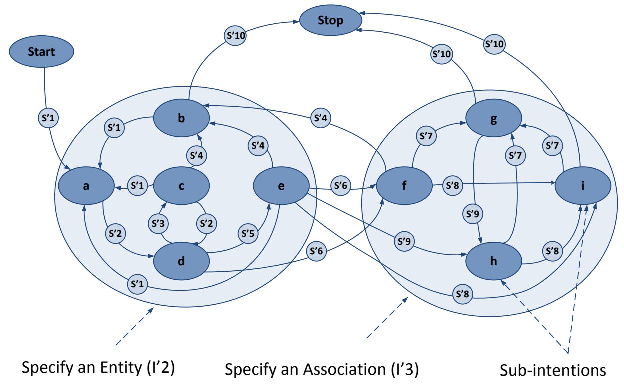

Figure 3: Map obtained by MMM for E/R diagrams experiment.

Eclipse Foundation provides UDC dataset to help committers and organizations to better understand how the community makes use of Eclipse platform [11]. In this perspective, our contribution is to model the UDC developers' behaviors in terms of intentions and strategies while using Eclipse platform. The obtained Map can help to better understand developers' behaviors.

The dataset contains $1,048,576$ event logs from developers who agreed to send their data back to the Eclipse Foundation. These data aggregate activities from over $10,000$ Java developers between September 2009 and January 2010. The activities are recorded by timestamps for each developer, which allows knowing when and by whom activities were committed.

4.1 Applying MMM on Dataset

In order to apply MMM, it is important to prepare the dataset. The number of unique developers' activities per month exceeds 500 activities. This number contains both the recurring activities and the non-recurring activities, *i.e.*, activities which are not frequently performed by developers. These activities are not representative of the developers' behavior characteristics because they have not been repeated enough to be a behavioral-pattern. For this reason, and also for readability, we limit this study to the 150 most frequent activities performed by developers. Table 2 contains the list of these activities. Some of these activities are the commands performed directly by developers; some of them are the frameworks, plug-ins or built-in features of Eclipse used by developers during their development process. For readability reasons, the prefix *org.eclipse* of the activities is removed. The plug-ins and frameworks are shown in bold letters and the related activities are inside brackets.

Once the dataset is ready, BWA estimates the transition matrix (developers' strategies). Note that, once again the number of strategies obtained by the heuristic method for this case study is 10. The strategies are represented in Table 2 with their corresponding groups of activities. Finally, the Map obtained by Deep Miner and Map Miner algorithms is shown on Figure 5.

Figure 4 depicts the effect of the choice of ε on the likelihood and the number of sections of the obtained Map. As mentioned earlier, ε expresses the level of abstraction for a Map. An expert can choose the value of ε regarding the expected level of abstraction. In this case study, the value of ε is set to 0.06 to have a good trade-off between having a likelihood with a relative high value and a reasonable number of sections.

Regarding the obtained Map, 22 sub-intentions are grouped by Map Miner algorithm into 7 groups of high-level intentions. Note that MMM can discover accurately the beginning and the end of a process; thus the intentions **Start** and **Stop** are clearly determined on the obtained Map. The transition probabilities from one intention to a strategy are annotated on the arrows. These values correspond to the probabilities that the developers selected a strategy from a given intention. The values on the loops indicate the probabilities that the developers continued to perform the activities related to the looped strategies.

4.1.1 *Naming Strategies and Intentions*

As found in E/R diagrams experiment and this case study, MMM discovers the strategies (the arrows) and where they lead, *i.e.*, intentions (the nodes). In other words, MMM reconstructs the topology of the Map process model from event logs; the names of strategies and intentions are not fully automatically generated. Nevertheless, it is possible to infer the names of strategies and intentions from the emission matrix **E**. Indeed, **E** specifies the activities associated to each strategy discovered by the MMM (see Table 2). Therefore, based on the names of activities grouped into a strategy and through a semantic analysis of their properties and interrelationships, it is then possible to manually infer the names of the strategies. In the same way, since the strategies lead to intentions, the names of intentions can be inferred by analyzing the strategies leading to each intention. For instance, the main activities grouped into the strategy S_5 are 'refactoring.commands', 'jdt.junit', 'debug.ui.commands', etc. From these activities one can infer the developers wanted to debug, to refactor and to test a code; thus, the name inferred for this strategy is *by refactoring, testing and debugging*. Further, the main activities for strategy S_6 are 'delete', 'paste', 'copy', 'undo', etc. This means the developers wanted to modify a code or a file; the name inferred for this strategy is *by file modification*. Since both strategies lead to an intention, we infer that the developers who performed S_5 and S_6 intended to **Fix a bug**. By applying this procedure, the names of all strategies (denoted on the arrows) and intentions are semi-automatically inferred. The inferred names of intentions are **Start, Initiate the development, Manage tasks, Fix a bug, Improve the code, Commit the code**. This naming protocol remains to be fully automated by building ontologies.

4.2 Developers' Behavior Analysis

Discovering the Map for Eclipse UDC developers allows understanding the developers' behaviors during the development process. As shown on Figure 5, they have selected different sequences of strategies with different probabilities, to fulfill their intentions. An expert can analyze these behaviors in order to understand how/why developers make use of different components or plug-ins of Eclipse: where they follow the *best practice* of software development projects and where they deviate from these rules, which components or plug-ins are more involved than the others, which strategies are more/less taken or where are system

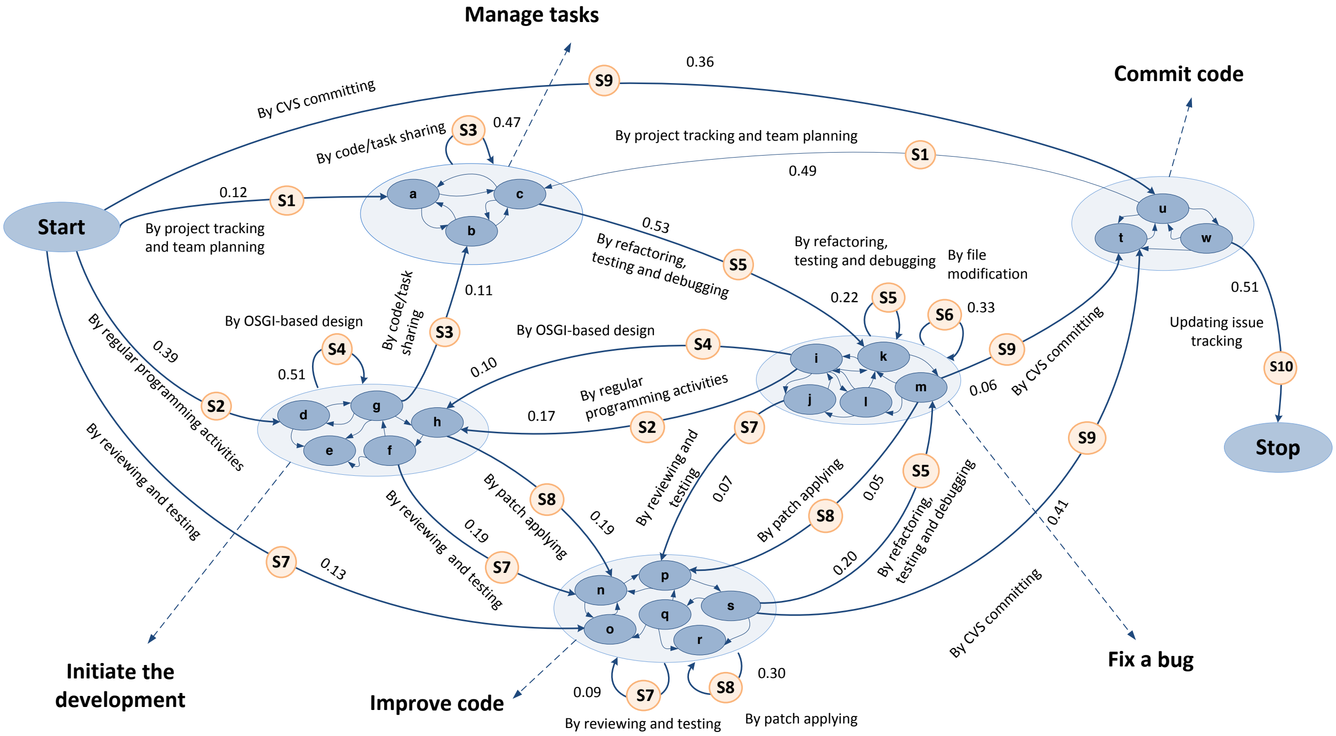

Figure 5: The obtained Map for Eclipse UDC.

Table 2: Strategies labels and related activities for Eclipse UDC

Strategies labels	Activities names
S_1	**mylyn.tasks.ui.commands.**[OpenTask, AddTaskRepository, ActivateTask, SearchForTask], **mylyn.context.ui.commands.**[Open.context.dialog, AttachContext, interest.Increment, interest.Decrement], **mylyn.monitor.ui**, **mylyn.team.ui**
S_2	**core.**[jobs, net, filesystem, resource, runtime, variables, contenttype, databinding.observable], **equinox.p2.ui.sdk.install**
S_3	**mylyn.context.ui.commands.**[Open.context.dialog, AttachContext, interest.Increment, interest.Decrement],**team.cvs.ui.**[branch, replace, GenerateDiff, ShowHistory, Add, Tag,merg, compareWithTag], **jsch.core**, **mylyn.**[monitor.ui, team.ui, commons.ui, bugzilla.ui]
S_4	**pde.ui.**EquinoxLaunchShortcut.run, **equinox.p2.ui.**sdk.update, **equinox.**[ds, simpleconfigurator.manipulator, frameworkadmin, app, common, directorywatcher, engine, core, metadata.repository, garbagecollector, ui.sdk.scheduler, repository, preferences, exemplarysetup, registry, updatechecker]
S_5	**core.**[databinding.observable, core.net, core.filesystem, core.resource, core.runtime, core.variables, core.contenttype], **debug.ui.commands.**[RunLast, Debuglast, eof, StepOver, TerminateAndRelaunch, execute, AddBreakPoint, TogglebreakPoint], **jdt.debug.ui.** [commands.Execute, commands.Inspect], **jdt.junit.**[junitShortcut.rerunLast, gotoTest, junitShortcut.debug], **ltk.ui.refactoring.commands.**[deleteResources, renameresources, moveResources], compare.selectPreviousChange
S_6	**ui.edit.**[delete, paste, copy, undo, text.goto.lineEnd, text.contentAssist.proposals, text.goto.wordNext], **ui.file.save**
S_7	cdt.ui.editor, **jdt.junit.**[junitShortcut.rerunLast, gotoTest, junitShortcut.debug], **team.cvs.ui.**[CompareWithRevision, CompareWithLatestRevisionCommand, CompareWithWorkingCopyCommand],**ui.edit.**[delete, paste, copy undo, text.goto.lineEnd, text.contentAssist.proposals, text.goto.wordNext]
S_8	**team.ui.**[synchronizeLast, teamSynchronizingPerspective, synchronizeAll, applyPatch], **ltk.core.refactoring.refactor.**[create.refactoring.script, show.refactoring.history]
S_9	**mylyn.monitor.ui**, **mylyn.context.ui**, **mylyn.commons.ui**, **team.cvs.ui.**[commitAll, Commit, CompareWithRemote, Sync]
S_{10}	**mylyn.monitor.ui**, **mylyn.bugzilla.core**, **mylyn.bugzilla.ui**, **team.cvs.ui.**[commitAll, Commit, CompareWithRemote, Sync]

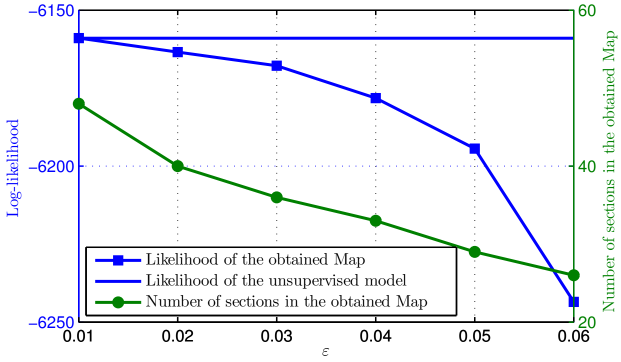

Figure 4: The effect of the choice of ε on the likelihood and the number of sections of the obtained Map for the Eclipse traces.

bottlenecks, etc. The obtained Map can also be used to provide recommendations to developers in order to choose the best path (strategies) to fulfill his/her intentions. In this case study, there is no prior model to compare with the obtained Map. However, MMM has proven to be effective with the controlled experiment (see Section 3) as the likelihood of the obtained Map is higher than the prescribed one. Hereafter, several observations, from the obtained Map, are detailed to attest the aforementioned claims in practice.

Observation 1. The developers' activities involved the usage of frameworks and plug-ins such as **Mylyn, Equinox, team/CVS, Junit**, built-in features of Eclipse such as **Eclipse Core, Debug** and API such as **ltk** (Language Toolkit). Figure 6 depicts the usage probabilities of these frameworks in each strategy. We can also observe the usage frequency of each framework/plugin/tools among the strategies. For instance, the usage frequency of *Mylyn* framework is 4 times among 10 strategies and its usage probabilities for strategy S_1 is 1 whereas for strategy S_5 it is 0. These values help detecting if any of these frameworks or tools is underused. For instance, we observe that the refactoring tool **ltk** is underused since it has a low usage probability (see also observation 4).

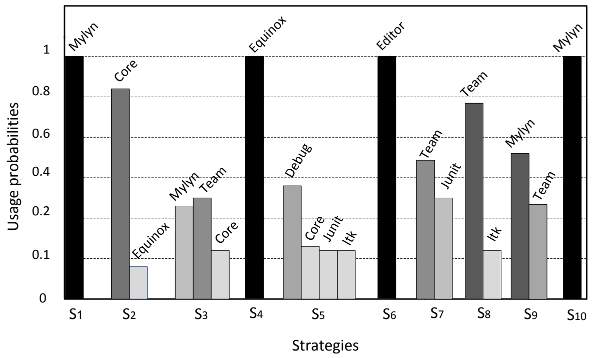

Figure 6: Usage probabilities of different Eclipse elements for each discovered strategy.

Observation 2. The developers who had started a development process have chosen one of the 4 first strategies, *i.e.*, S_1, S_2, S_7, S_9. If they already had an ongoing program at hand, they have tended to adopt either the strategy S_7: **by reviewing and testing** to **Improve the code** or the strategy S_9: **by CVS committing** to **Commit the code**. On the other side, the developers who had started a new development have chosen either the S_2 strategy: **by regular programming activities** to **Initiate the development** or the strategy S_1: **by project tracking and team planning** to **Manage tasks** with probability 0.39 and 0.12 respectively. This observation suggests that the developers of this case study have tended to start programming by building first the baselines for software architecture through dividing the programs into different modules/packages using **OSGI-based design**. After decomposing the software into sub-modules, the developers have utilized the strategy S_3 : **by code/task sharing**, which involves mainly Mylyn framework (described in observation 3) that can be integrated with the Bugzilla bugtracker system and issue system. Therefore, tasks and the content of these tasks can be shared among developers. The high transition probability of this strategy means the developers tend to **share code/task** in order to fulfill **Manage tasks**.

Observation 3. The developers who had the intention to **Manage tasks** chose the strategies S_1 and S_3. Regarding Table 2, the activities related to these strategies involve Mylyn framework, which is the task and application life-cycle management (ALM) framework for Eclipse. It helps the developers to work efficiently with many different tasks such as bugs, problem reports or new features. It monitors users' activities and preserve the context of the task-at-hand to focus the Eclipse UI on the related information. For instance, while working on a current task, if the developers have to work on another task, *e.g.*, an occurred bug, Mylyn preserves the context of the current task. Thus, the developers can work on another task without losing the context of previous task. This procedure is discovered in the obtained Map of Eclipse. The activities of the developers while they **Manage tasks** were interrupted (*e.g.*, for an urgent bug) and they have chosen the strategy S_5 to **Fix a bug**. To switch to the previous task, they have first committed and reported the bug, then they have continued managing tasks. We can deduce the Mylyn framework has reliable and relevant functionalities for developers and it is not underused since they work with it 47% (transition probability of 0.47) of their time and they use it frequently (Figure 6).

Observation 4. When the activities of the developers during **Manage tasks** were interrupted, they have chosen strategy S_5 to **Fix a bug**. The procedure of fixing a bug may involve refactoring existing code, writing unit tests, editing and modifying involved code and finally fixing the affected code. The strategy S_5 : **by refactoring, testing and debugging** represents this procedure. This strategy is defined as a *best practice* to fix a bug, which means the developers who have adopted this strategy respect the guidelines of software development. However, they have performed this strategy with probability 0.22 whereas they have preferred to **Fix a bug** by **file modification** with a higher transition probability (0.33). Moreover, Figure 6 shows a low usage frequency and usage probability for **ltk** (Language Toolkit), which is an API for automated refactoring in Eclipse-based IDEs. This means the developers have preferred refactoring manually instead of using the refactoring and debugging tools of Eclipse provided for this purpose. This confirms some results of empirical research [20, 28, 12, 31] that refactoring tools are seldom used and that they are not developers-friendly enough.

Observation 5. Bug fixing includes debugging and refactoring of different software modules. Once developers have fixed a set of bugs, they have chosen either strategy S_8: **by patch applying** with probability 0.05 or strategy S_7: **by reviewing and testing** with probability 0.07 to fulfill the intention of **Improve the code**. This observation means when debugging and refactoring have an impact on different parts of the program, the developers have improved their code **by patch applying**. If debugging and refactoring change the program locally, they have manually changed the code and have sent it for **reviewing and testing**. The probabilities describe that the developers have tended to modify the codes manually.

Observation 6. The Map can be used as a behavioral pattern to build recommender system for assisting developers in their daily development tasks. For example, this recommender system can suggest developers to avoid some problematic paths/strategies which might deviate from organizational best practices and to take some more effective paths/strategies. For instance, since the procedure of refactoring and debugging might affect the code in different parts of the program, it is recommended to check the coherence integrity of the program. To do so, after **fixing a bug**, the developers can be recommended to take the strategy S_4: **by OSGI-based design** to **Initiate the development**, to check if the refactoring and debugging have been impacted the **OSGI-based design** (*e.g.*, if the dependencies between the modules are changed, etc).

4.3 Process Improvement

The obtained Maps can also contribute to improve the usability of the software products. The Map can help the novice or unfamiliar users learning system features by making the task of learning easier, by using the Map obtained from previous development process as a guideline. For instance, when their intentions are known, they can be recommended which strategies and activities might be useful to fulfill their intentions. The intentional topology of a Map can help software designer designing systems, which enable users to be more efficient in their operations. This can be done by adapting the system to the users' needs or by assisting users step by step while enacting a process. Using a Map increases also the users' confidence and satisfaction in the enactment of a process.

5. RELATED WORK

The workflow models approaches propose to discover process models from logs generated during process enactment. However, they model users' behaviors in terms of activities [29, 30, 7, 19] and overlook the underlying users' intentions and strategies. Some works propose to mining users' intentions from logs [2]. These works differ from our approach because they look for a single user's intention underlying a single log while our approach proposes to infer common users' intentions during enactment of a process to achieve the ultimate process goal. HMMs are used for model discovery by Cook et al. [7, 14]. However, these works do not address the humans' cognitive process such as users' intentions and strategies, which cause the activities performance. Finite State Machine (FSM) is used in some works to model users' behaviors [6, 19]. While our approach models hidden states as users' strategies, FSM is a simple automaton and is not appropriate to model cognitive processes. Several experimental works address the developers' behavior in the software engineering field [20, 28, 19]. In [20], a technique id proposed to improve the user interface of automated refactoring by instructing several participants to apply the Extract Method refactoring on several open source projects. In [28], the focus is on Eclipse UDC refactoring histories to understand why developers underuse automated refactoring. However, these approaches are based on statistical reports and do not consider underlying users' intentions during the development. Approach of [26] suggests assisting developers of Hadoop Clouds by a simple technique, which consists of log abstraction, log linking, and simplifying sequence, etc. The approach of [5] proposes a tool for debugging. It uses three types of mined temporal invariants to guide the model space exploration. However, these kinds of techniques do not allow to uncover users' cognitive process such as, intentions and strategies.

6. THREATS TO VALIDITY

Four main issues threat the validity of the proposed approach. First, the mined Map may suffer from underfitting problems if the number of activities sequences used to estimate the parameters of the HMM is not high enough. Indeed, these sequences have to capture all the possible behaviors while enacting the process under study to produce an accurate Map. Second, the BWA requires an important number of iterations to converge to a result. For instance, it converges at $9,986$ learning iterations for the controlled experiment and $20,237$ learning iterations for the Eclipse case study. Moreover, it cannot always be guaranteed to converge to the global maximum likelihood. Third, although the MMM automatically discovers the topology of the Map process model, the names of strategies and intentions are still inferred semi-automatically. The manual part of this naming procedure introduces a human bias. Fourth, the M_1M_0 topology chosen for the HMM is the most sophisticated topology allowing the use of an algorithm such as BWA or equivalent algorithms. More complex topologies may actually be more appropriate to model some processes. However, there is no known algorithm to estimate the parameters of these topologies, as BWA does for M_1M_0. Therefore, the scope of MMM is limited to the M_1M_0 topology.

7. CONCLUSIONS

This paper has presented the Map Miner Method (MMM), a method for automating the construction of an intentional process model from users' activities logs. Constructing an intentional process model from logs is a novel approach to model processes. It helps understand the users' intentions and strategies while using software products. Further, it can help software designers adapting the products functionalities to better meet users' needs, therefore making products more user-friendly. On the other hand, the intentional model can assist any user (novice or experimented) by providing recommendations to choose the next step of a process. The main contribution of this work is to build from theory a method which fully constructs the topology of an intentional model, only based on users' activity logs. This makes the method easily applicable to any dataset, as long as it is made of users' traces of activities. In this paper, MMM was validated by a controlled experiment (creation of E/R diagrams by students) and then by a large-scale case study (Eclipse UDC). Although the MMM automatically discovers the topology of the intentional model, the names of strategies and intentions are still inferred semi-automatically. This could be fully automated by building a more sophisticated ontology. In addition, we are developing an ProM [21] plug-in, which will allow modeling processes in intentional manner.

8. REFERENCES

[1] I. Ajzen and M. Fishbein. Attitude-behavior relations: A theoretical analysis and review of empirical research. *Psychological bulletin*, 84(5):888, 1977.

[2] R. Baeza-Yates, L. Calderón-Benavides, and C. González-Caro. The intention behind web queries. In *String processing and IR*. Springer, 2006.

[3] L. E. Baum, T. Petrie, G. Soules, and N. Weiss. A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains. *The annals of mathematical statistics*, 41(1), 1970.

[4] Y. Bengio. Learning deep architectures for ai. *Foundations and trends in ML*, 2(1):1–127, 2009.

[5] I. Beschastnikh, Y. Brun, S. Schneider, M. Sloan, and M. D. Ernst. Leveraging existing instrumentation to automatically infer invariant-constrained models. In *The 19th ACM SIGSOFT symposium and the 13th ECFSE*, pages 267–277. ACM, 2011.

[6] A. W. Biermann and J. A. Feldman. On the synthesis of finite-state machines from samples of their behavior. *Computers, IEEE Transactions on*, 100(6):592–597, 1972.

[7] J. E. Cook and A. L. Wolf. Discovering models of software processes from event-based data. *ACM (TOSEM)*, 7(3):215–249, 1998.

[8] A. Dardenne, A. Van Lamsweerde, and S. Fickas. Goal-directed requirements acquisition. *Science of computer programming*, 20(1):3–50, 1993.

[9] F. D. Davis, R. P. Bagozzi, and P. R. Warshaw. User acceptance of computer technology: a comparison of two theoretical models. *Management science*, 35(8):982–1003, 1989.

[10] Eclipse. Filtered udc data. `http://archive.eclipse.org/projects/usagedata/`, 2013.

[11] Eclipse. Usage data collector. `http://eclipse.org/org/usagedata/`, 2013.

[12] M. Fowler. *Refactoring: improving the design of existing code.* Addison-Wesley Professional, 1999.

[13] J. A. Hartigan and M. A. Wong. Algorithm as 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society*, 28(1):100–108, 1979.

[14] J. Herbst and D. Karagiannis. Integrating machine learning and workflow management to support acquisition and adaptation of workflow models. In *Database and Expert Systems Applications, 1998. Proc. 9th Intl WS. on*, pages 745–752. IEEE, 1998.

[15] M. Jarke and K. Pohl. Establishing visions in context: towards a model of requirements processes. In *ICIS*, pages 23–34, 1993.

[16] G. Khodabandelou. Contextual recommendations using intention mining on process traces. In *Proceedings of 7th Intl. Conf. on RCIS*, 2013.

[17] G. Khodabandelou, C. Hug, R. Deneckère, and C. Salinesi. Process mining versus intention mining. In *Enterprise, Business-Process and Information Systems Modeling*, pages 466–480. Springer, 2013.

[18] G. Khodabandelou, C. Hug, R. Deneckere, C. Salinesi, et al. Supervised intentional process models discovery using hidden markov models. In *Proceedings of 7th Intl. Conf. on RCIS*, 2013.

[19] D. Lorenzoli, L. Mariani, and M. Pezzè. Automatic generation of software behavioral models. In *ICSE'08. ACM/IEEE 30th*, pages 501–510. IEEE, 2008.

[20] E. Murphy-Hill and A. Black. Breaking the barriers to successful refactoring. In *Software Engineering, 2008. ICSE'08. ACM/IEEE 30th International Conference on*, pages 421–430. IEEE, 2008.

[21] ProM. `http://processmining.org`, 2013.

[22] L. Rabiner and B. Juang. An introduction to hidden markov models. *ASSP Magazine, IEEE*, 3(1), 1986.

[23] C. Rolland. Modeling the requirements engineering process. In *Information Modelling and Knowledge Bases V*, pages 85–96, 1993.

[24] C. Rolland, N. Prakash, and A. Benjamen. A multi-model view of process modelling. *Requirements Engineering*, 4(4):169–187, 1999.

[25] C. Rolland and C. Salinesi. Modeling goals and reasoning with them. In *Engineering and Managing SR*, pages 189–217. Springer, 2005.

[26] W. Shang, Z. M. Jiang, H. Hemmati, and e. a. Adams. Assisting developers of big data analytics applications when deploying on hadoop clouds. In *Proceeds. of the 2013 ICSE*, pages 402–411. IEEE Press, 2013.

[27] P. Soffer and C. Rolland. Combining intention-oriented and state-based process modeling. In *Conceptual Modeling–ER 2005*, pages 47–62. Springer, 2005.

[28] M. Vakilian, N. Chen, S. Negara, B. A. Rajkumar, B. P. Bailey, and R. E. Johnson. Use, disuse, and misuse of automated refactorings. In *ICSE, 2012 34th*, pages 233–243. IEEE, 2012.

[29] W. Van der Aalst, T. Weijters, and L. Maruster. Workflow mining: Discovering process models from event logs. *Knowledge and Data Engineering, IEEE Transactions on*, 16(9):1128–1142, 2004.

[30] W. M. Van der Aalst and W. van der Aalst. *Process mining: discovery, conformance and enhancement of business processes.* Springer, 2011.

[31] Z. Xing and E. Stroulia. Refactoring practice: How it is and how it should be supported-an eclipse case study. In *Software Maintenance, 2006. ICSM'06*, pages 458–468. IEEE, 2006.

[32] E. Yu. Modelling strategic relationships for process reengineering. *Social Modeling for Requirements Engineering*, 11:2011, 2011.