

Collaborative Activity Indicators Engineering: Using modeled traces in the context of Technology Enhanced Learning Systems

T. Djouad, A. Mille, C. Reffay and M. Benmohammed

Abstract— Designing and exploiting collaborative activity indicators is a strong challenge for Technology Enhanced Learning Systems (TELS). In this work we compute indicators using modeled traces transformation and we use to allow possible this computation a Trace Based System. We develop a management TBS for Moodle. We propose an original model-driven approach, introducing the concepts of modeled traces (MT) and of trace-based system. In this approach, designing an indicator consists in explicitly modeling the trace to compute it, highlighting the elements of observation necessary to collect in order to elaborate a specific indicator. Therefore, effective implementation of an indicator consists in expliciting the transformations carried out of available traces on a particular learning platform. These initial observed elements (obsels) form what we call the primary modeled traces. A «path» of transformation is explicitly designed and applied for both documenting and performing the indicator computation. Using this approach, it is possible to form a library of enriched indicators models in association with the corresponding primary trace model and the explicit transformation path. Reusing an indicator in various contexts is easier and well documented. This approach is illustrated on the Moodle platform in order to explain the practical use of our implemented system.

Index Terms— Modeled trace, Trace Based System, indicators computation, collaborative learning.



1 INTRODUCTION

EVALUATING collaborative learning situations by considering a learner alone or a group of learners in Technology Enhanced Learning Systems is a delicate task, which requires a permanent adaptation during the learning activity. To understand the dynamic of learning and effectively evaluate collaborative learning situations, researchers and TELS designers use interaction traces analysis in a significant number of systems. A useful survey of such systems is presented in [1]. When activity traces are used to create indicators, the models of interaction traces have to be adapted to enable teachers to design and automate the collaborative indicators computation, assisting them to understand, evaluate, monitor and support online learning.

In this paper, we are specifically interested in studying methods used to facilitate the analysis of traces in collaborative learning situations. These methods are based on a Model Driven Engineering (MDE) approach to design and compute effectively collaboration indicators, which are learning-platform independent. Our research question addresses three issues:

- How to get and restructure the raw data from their sources (ex: log files) to create first modeled traces called *primary traces* [2]. During this data collecting phase, *the primary trace* is built from needed data delivered by the TELS as basic information to elaborate the target indicator;
- What are the *transformations* required, according to the MDE approach, and what are the operators to define

and to compute, from that primary trace, intermediate traces that make sense to describe the collaborative activity;

- What are the ultimate transformed trace models associated to each collaborative indicator, and what is the structure of these models. These indicators models can be used by the researcher, the activity's designer, the trainer, the tutor or even the learner himself in a learning situation.

This work is in the field of Computer Supported Collaborative Work (CSCW), and is based on Model Driven Engineering, to transform trace models and build collaborative indicators.

1.1 Indicators in Computer Supported Collaborative Learning (CSCL)

According to Dimitracopoulou [3] "Interaction analysis indicators constitute variables that indicate 'something' related to the mode or the 'quality' of individual activity (e.g. variables that he/she change, order of significant actions, etc.), the mode or the quality of the collaboration (e.g. division of labour, participation rates, categories of specific contributions), the process or the quality of the collaborative product". An indicator is a mathematical variable which has a list of characteristics. It is a **variable** that takes digital, alphanumerical or graphical **values**. A value has a **status**: a value is *calibrated* or *interpreted*. An indicator may depend on other variables such as time, or even other indicators. Dimitracopoulou in [4] proposes to use interaction analysis tools to compute indicators values. Fig. 1 explains how these tools work. Users (trainers, learners, etc.) use different activity tools offered by learning environments like whiteboard, chat, forums, etc. Data selection helps to get (in automatic or semi-automatic way) different user's interaction traces issued from the tools based activities. We can retrieve information needed to form these traces from files such as log files of learning platform or we can collect observed elements directly during the activity itself by instrumenting the application. Basic observed elements are chosen in the available observable elements according to their role for indicators computation.

Analysis processes produce several kinds of indicators. An indicator may indicate the mode or the quality of individual contributions (ex: Send a message in a chat), collaboration (ex: the division of labor, the density or the cohesion of a group) or also the quality of what was finally produced (ex: the depth of a discussion thread in a forum).

The indicator value is used to build a feedback to different users. According to the categories proposed in [1], this feedback can be a direct visualization of the indicator value (*mirroring*), or the value can be compared to a desired state (*monitoring*) or it can feed a more elaborated process providing guiding information to learners (*guiding*).

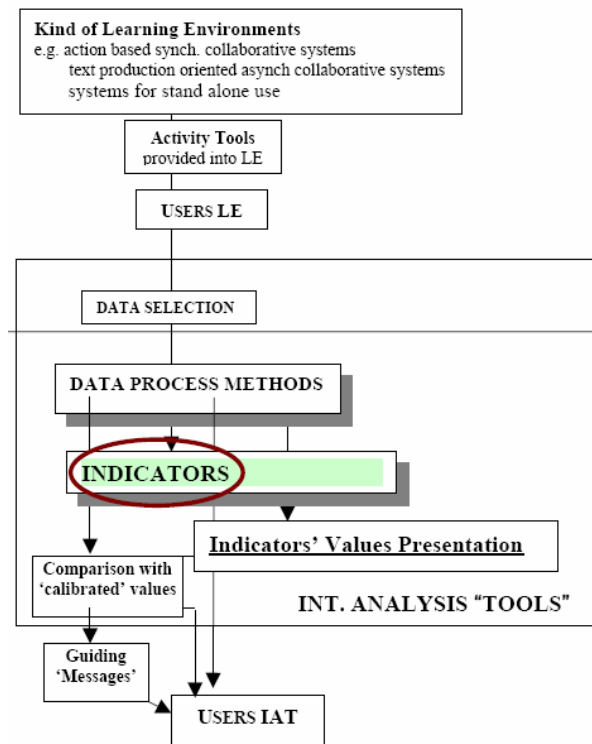


Figure 1 • Indicators computation [3]

A lot of work has been published about indicators, generally respecting the definition proposed in [3]. For example in [5] we compute the cohesion and the centrality in social networks from discussion forums. The platform ACOLAD [6] provides tutors with a tool which gives information about the activity 3-tuple: Assiduity, Availability, and Involvement. Santos et al. [7] offer a tool that computes the degree of involvement of each learner during the session from interaction traces. This tool identifies: participative learner, useful learner, non-collaborative learner, learner who takes initiatives, and communicative learner. Other indicators are qualitatively interpreted as in [8] where the density of the social network is interpreted using histograms. Tedesco [9] measures agreement and disagreement between learners.

May et al. [10] provides a method and tool to compute and visualize the indicator *“Read a message in a forum”* using traces from the server and from the client. As illustrated on Fig. 2, the sphere size shows the time spent by a user to read a message. The various spheres are located on the time axis, and their color depends on the type of action done by the user (ex: blue for posted messages, green for completely read (exposed) messages). This visualization gives more accurate information on quality of readings based on the part (scrolling) exposed and the time the user spent on it.

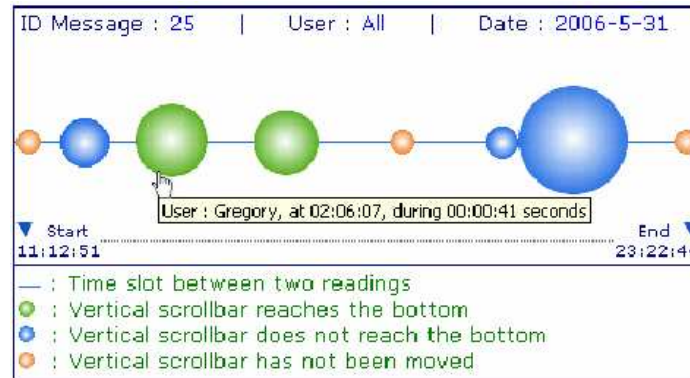


Figure 2 • Indicator visualization « Read a message » in Travis [10]

The "ForumExplor" platform [11] works on forums' archives. It simplifies large forum analysis by offering different tools: *thematic* views and *overviews* of forums' archives. The overview presented on (Fig. 3) is based on a simple statistics and represents the participation distribution along the time axis. The thematic view is based on lexical recognition and pre-registered themes. Each theme is associated to a word list and a color. (Fig. 4) shows for each post of a thread, the different themes it is related to.

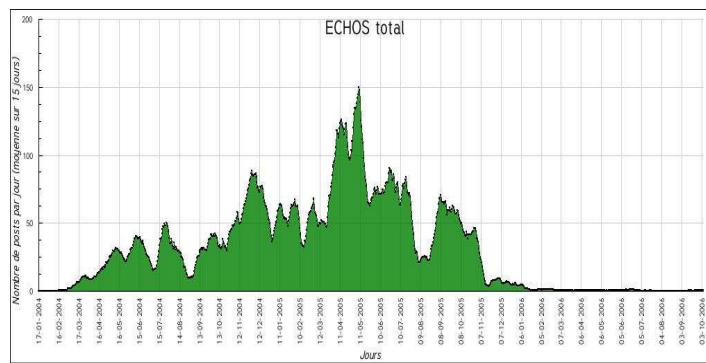


Figure 3 • Global view of the indicator: Participation in a forum

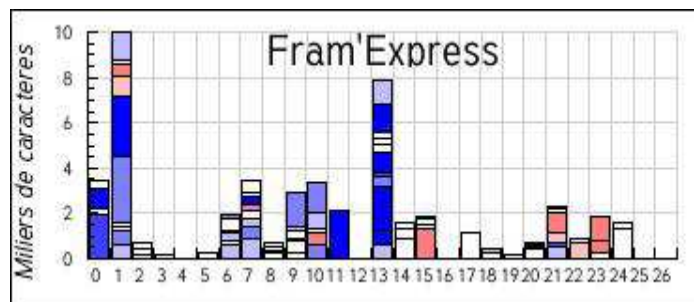


Figure 4 • Thematic view: color of a discussion thread

As seen in the above examples, indicators may play an important role for evaluating and guiding the learning process but there is often no explicit method to construct them and to integrate them in TEL systems. To address this issue, several researches tend to propose generic frameworks to design and implement indicators.

EM-AGIIR, an open multi-agents architecture defined in [12] provides a framework to reuse indicators. A *Query Agent* explores the tracing sources, and identifies the collected data needed to compute the indicator value; a *Database Agent* stores traces that feed the indicator; an *HMI Agent* displays the indicator value. The *Indicator Agent* analyzes the traces sent by the Database Agent and compute the indicator value (where the indicator is computed by a function f), and thereafter, sends results to the *HMI Agent* to display the results. Fig. 5 explains the principle of this architecture.

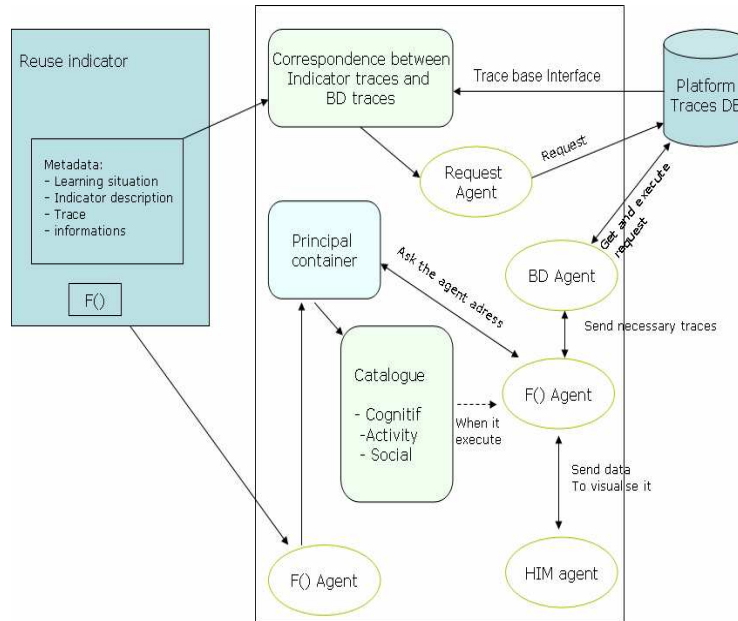


Figure 5 • The architecture of EM-AGIIR [12]

This architecture offers a nice way to integrate different codes for computing an indicator but there is no model of what is a trace, no explicit formalism to describe the indicator computation and no way to explicit how trace elements are used during the indicator computation.

Reference [13] proposes to reuse and improve educational scenarios using a formal grammar to compute indicators. The representation model of the indicator is based on three parts: the *Defining* part qualifies the need for observation, the *Getting* part describes how data acquisition is performed, and the *Using* part shapes the indicators use.

This model is very interesting to guide and to document the way a descriptor is composed but there is no way to move from this description to an operational process of the indicator computation when described. Nevertheless, this explicit description could become the input of an automatic indicator computation.

1.2. The indicators in CSCL: some examples

Many indicators are listed in [3] within their corresponding platforms. The computation of some of these indicators is detailed in this section in order to be used as examples in part 2. They have been chosen as representative of a large

panel(?) of indicators. In this section, we also refer to the learning platforms they have been implemented in.

1.2.1. The division of labor

This indicator is defined and implemented in the platform “*Collaborative TRAffic Simulator*” [14]. It identifies the division of labor adopted by two users working on a set of shared resources. Specifically designed for researchers, this indicator identifies the role taken by each participant in the collaborative learning process. The authors define three types of division of labor: *Task based division* where each person acts on separate resources; *Role based division* where one person edits all resources and *Concurrent division* where both persons act more or less equally on all resources. The characterization of these three types of division of labor can be computed from the *Sum of Differences (SD)* and the *Sum of Absolute Differences (SAD)*:

$$SD = \frac{\sum_i (S1Ai - S2Ai)}{S1A + S2A} \quad SAD = \frac{\sum_i |S1Ai - S2Ai|}{S1A + S2A}$$

With $S1Ai$ (respectively $S2Ai$): the number of actions made by the subject $S1$ (respectively $S2$) on the resource Ai , and $S1A$ (respectively $S2A$): the number of actions made by the subject $S1$ (respectively $S2$) on all the resources. The SAD indicates the symmetry of actions. The value 0 means that both users have the same number of actions, while the value 1 means that all actions were made by the same user. Fig. 6 shows the different types of division of labor according to extreme and meaningful values (0, 1, -1) for variables SD and SAD .

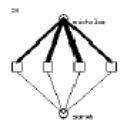
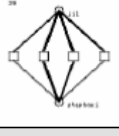
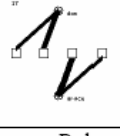
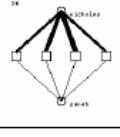
	Sum of Absolute Differences (SAD)	
Sum of Differences (SD)	0	1
1	N / A	Role 
0	Concurrent 	Task 
-1	N / A	Role 

Figure 6 • Classifying the division of labor using SD and SAD indicators [14]

1.2.2. Interaction indicator

The *Interaction* indicator is implemented in the platform MODELLINGSPACE [15], and used by teachers to measure the activity rate in problem solving. It computes the number of actions made in an activity module (like a chat) within a time interval. If we consider the time interval $[t_0-t_m]$ associated to a collaboration session, time is quantified as follows: $t_i = t_0 + i * d$ with $d = (t_m-t_0) / n$. $Interactions(k, t_i)$ is the number of actions made in an activity module k during the time interval $[t_{i-1}, t_i]$, with k in $[k, k_{max}]$ corresponding to the interaction tools ($k = 1 \Rightarrow$ Chat, $k = 2 \Rightarrow$ Forum,...).

If $Interactions(k, t_i) = 0$ then no action is made on the activity module k , in the time interval $[t_{i-1}, t_i]$. If the value of $Interactions(k, t_i)$ is high, collaboration may have occurred between users during this time interval.

1.2.3. Active Agent Indicator

This indicator implemented in the platform MODELLINGSPACE [15] is used by teachers to measure the activity (eg : send a message in chat) during a problem solving activity. An actor (ex: learner) is only active if he interacts in an activity module during a time interval. The indicator *Active Agent* shows the number $Agents(k, t_i)$ of actors who have interacted at least *once* in the activity module k during a time interval $[t_{i-1}, t_i]$.

1.2.4. Collaborative Action function indicator (CA)

This indicator, also implemented in the platform MODELLINGSPACE [15] is used to represent both the number of actions and the number of active agents during the problem solving. This indicator is computed from two previous indicators: $Agents(k, t_i)$ and $Interaction(k, t_i)$, during a time interval t_i . The indicator Collaborative Action $CA(t_i)$ is computed by the following formula:

$$CA(t_i) = \sum_{k=1}^{k=k_{max}} Agents(k, t_i) * Interactions(k, t_i)$$

1.2.5. Non Verbal Actions indicator (NVA)

The indicator NVA, implemented also in the platform MODELLINGSPACE [15], is used by teachers and researchers during and after the collaboration session. It represents the percentage of all non-verbal actions made on different interaction tools. The idea is to compute first: all verbal actions (ex: chat message, forum message, etc.), and then, consider the rest of actions as non-verbal. The following example considers that chat actions are the only verbal actions:

$$NAV(t_i) = \frac{\sum_{k=1}^{k=k_{max}} Interactions(k, t_i) - Interactions('Chat', t_i)}{\sum_{k=1}^{k=k_{max}} Interactions(k, t_i)} \text{ if } \sum_{k=1}^{k=k_{max}} Interactions(k, t_i) > 0$$

If $NVA \approx 0$ then the actors use only verbal actions. $NVA \approx 1$ means that no verbal action is made in the activity tools during the selected time interval t_i .

1.2.6. Selected Agent Contribution indicator (SAC)

The SAC indicator implemented also in the platform MODELLINGSPACE [15] is used by learners, researchers and teachers to evaluate the participation of a given actor during a synchronous problem solving. It computes the interaction rate of this actor (Agent) compared to all interactions occurring in an interaction module k , during a time interval $[t_{i-1}, t_i]$. The formula is:

$$SAC(agent, t_i) = \frac{\sum_{k=1}^{k=\max} Interactions(k, agent, t_i)}{\sum_{k=1}^{k=\max} Interactions(k, t_i)} \text{ if } \sum_{k=1}^{k=\max} Interactions(k, t_i) > 0$$

$Interactions(k, agent, t_i)$ is the number of actions made by a referred actor (agent) in module k during the time interval $[t_{i-1}, t_i]$. If SAC=0 then the actor performed no action in the time interval. SAC=1 means that the referred agent performed all actions in the module k during this time interval.

1.2.7. Participation percentage indicator (PART)

PART is implemented in the platform MODELLINGSPACE [15], and represents the participation level of actors in any activity module. This indicator gives the percentage of agents who act in a specified time interval. The formula is:

$$PART(t_i) = \frac{Agent(t_i)}{TotalAgents}$$

For example, if PART = 0.5 then half of agents have interacted in some of the activity modules during $[t_{i-1}, t_i]$. If PART = 0 then no actor has acted in the group within this time interval.

2. COMPUTING INDICATORS FROM INTERACTION TRACES

To provide a general approach capable of both describing and making operational indicators in a TELS, we propose a MDE [16] method to compute indicators with a Traces-Based System (TBS) [17]. We explain the general approach of our method in the first section. We show how we use a Models Driven Engineering approach in the second section. The last section illustrates this approach through three different use cases based on TBS that build and reuse trace transformation to compute several indicators.

2.1. Approach

Indicators are built from observed elements in a trace. The right level of abstraction of observed elements needed for the indicator formula is obtained by trace model transformations from elementary observed elements traces in the TELS. This method includes three steps: Collecting data, transforming traces, and finally the computation step. Fig. 7 shows the order of these steps:

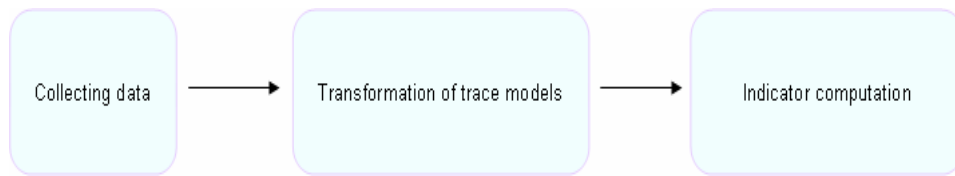


Figure 7 • Our proposed approach in three steps

2.1.1. Trace based system

A Trace Based System [18] is proposed by the research group SILEX² to manage modeled traces. A Modeled Trace (M-Trace) in a TBS is defined as a trace model and a set of traces instances according to this model. Each observed element (obsel) of a trace is located in the time axis. So, we call *trace* a collection of temporally situated observed elements. We mean by "observed element" any structured information resulting from observing interactions. The architecture of the TBS [17] represented in Fig. 8, includes: - a collecting system that builds the modeled traces from a tracing source; - a kernel offering tools to transform traces. This kernel includes: -a trace database (instances and model) and a transformation system to transform M-traces; - a visualization system; - a query System to interrogate the trace database.

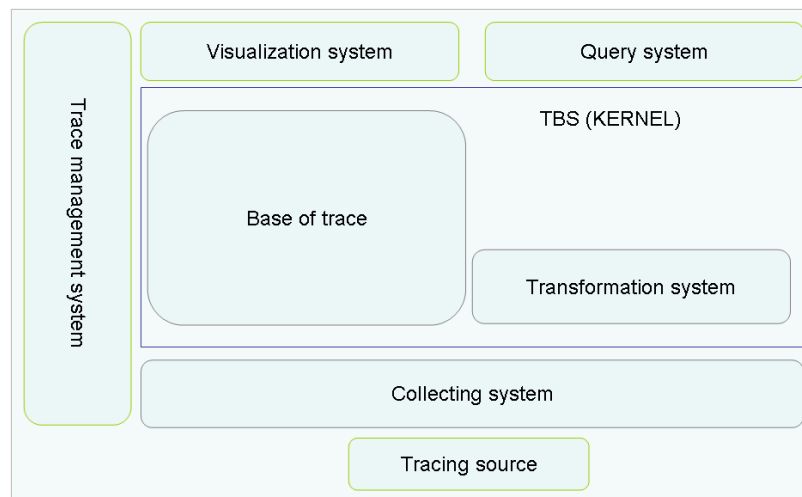


Figure 8 • TBS Architecture

The trace management system adds and removes M-traces, defines access rights, and manages the M-traces transformations. One can send requests to the M-Traces base, and retrieve information adapted to the users' needs. A *transformation of a modeled trace* is a process that transforms an M-Trace (managed by a Traces-Based System) to another M-Trace managed by the same system. The *primary M-Traces* in TBS are the only untransformed M-Traces. Fig. 9 shows an example of M-trace transformation. We can merge two M-Traces, the result is an M-Trace containing the instances of the two input M-Traces. We can create the model of the resulting M-Trace by a reformulation process or simply filter

² Liris.cnrs.fr/silex/

an M-Trace according to some criteria, etc. Main operators involved in indicators construction are presented in more details in section 3.2.

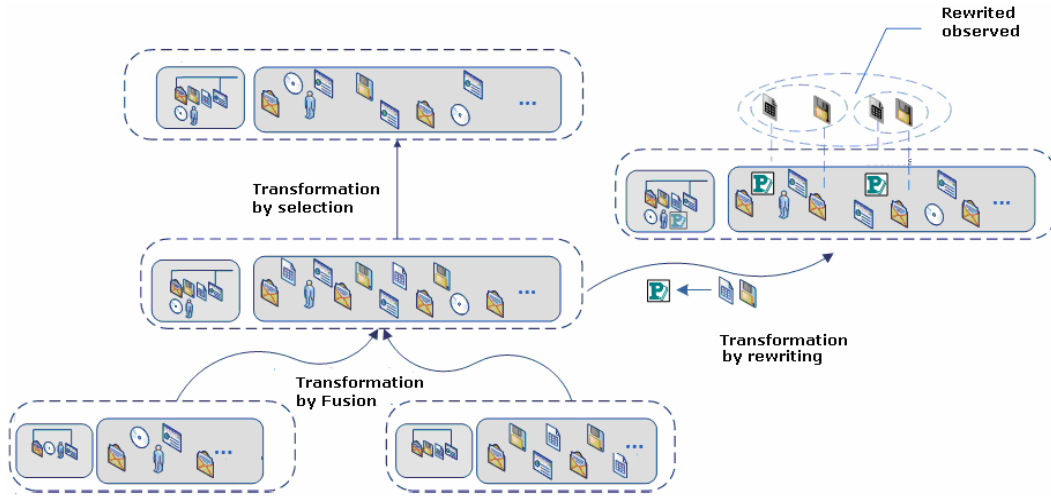


Figure 9 • Example of M-Traces transformation

2.1.2. Collecting data

Data collection consists of selecting pertinent/useful data within the tracing sources. On the one hand, this collection depends on learning platforms, and on the other hand, it depends on the primary trace model. The primary trace model defines what information is *needed*, and the tracing sources define what is *available*. In this Collecting phase, we use collaborative activity models, and, according to them, we collect all *obsels* that can feed these models. Section 3 will present an example of primary trace model for collaborative activities on the Moodle platform.

2.1.3. Transformations models

Starting from the primary trace, we propose transformation sequences using model-transformation operators (explained in Section 3.2). These operators modify either the model or its instances. We provide a library of models for transformed traces associated to indicators. Each of them is associated with its transformation path: we call these traces TM_{RI} , standing for Indicator Rule oriented Modeled-Trace. We can then reuse and easily adapt the transformation path to generate other models of M-Traces and therefore, other indicators.

2.1.4. Computation of indicators

Proposition

Our method associates a trace model with each indicator “I”. This model must contain all observation elements needed to compute the indicator directly (i.e: without any further transformation). We define an indicator “I” by:

$$I = \{R_I, TM_{RI}\}$$

With: R_I : the computation Rule, TM_{RI} the trace model used for computation.

Fig. 10 illustrates this proposition, where a trace $Trace(I)$, corresponding to its model TM_{RI} , is associated to a collaborative indicator $Indicator(I)$. The indicator value is derived from $Trace(I)$ by applying a computation rule R_I . For example, we can compute the indicator "Active Agent" by using a computation rule we call $R_{ActiveAgent}$.

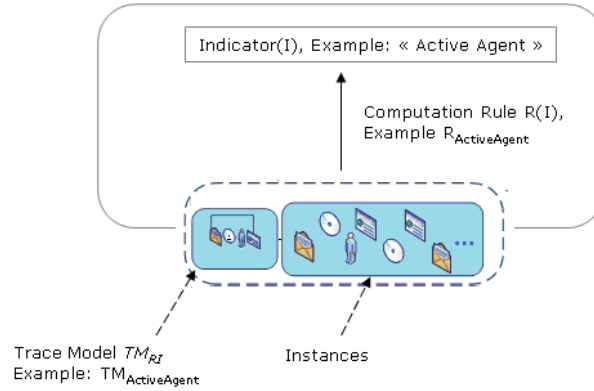


Figure 10 • Associating a trace model to each indicator to compute it

2.2. Model Driven Engineering

A model in [19] is defined as "A set of statements about some system under study". Another definition in [20] defines a model as "a simplification of a system built with an intended goal in mind. The model should be able to answer questions in place of the actual system". These general definitions do not allow easy computations and we adopt the definition given in [16]: A model is a description or a prescription for all or part of a system using a defined language. In the case of description, the model (describing the system) is correct if its characteristics and behaviour are evolving over time in the same way as the real system. Whereas in the case of prescription, the system is considered valid if none of the model's characteristics is contradictory with the obtained system. We use meta-models to describe models. A meta-model in [21] is a model, it defines a language that expresses the model. Our system handles models describing traces and provides transformation operators. The final traces (in the transformation path) may represent useful observed elements as information for indicators calculus. So, our system provides the operational semantics to execute the transformation path in order to get indicators values.

Model Driven Engineering in CSCL is inspired from software engineering and focuses on model changes (according to their contexts of use), rather than system coding. This reduces widely the effort of designers, teachers, researchers, etc. We consider our work is directly related to model driven engineering as it allows to manage dynamically models during the design of indicators. The indicators computation method we propose requires a model (M-Trace) and a transformation sequence to translate the primary trace model into the indicator trace model. This is the essential contribution of this paper to the indicators computation, compared with current methods.

Moreover, a Trace Model is a formal model, and an M-trace (a Trace with its model) has a precise semantics allowing requests for transformations. Thanks to the trace transformation process, the TBS provides facilities to design new models and to produce corresponding traces from existing ones.

2.3. Indicators engineering driven by trace models

With traditional methods (Fig. 11) and as we explained in section 1.1, to get an indicator, we prepare data before starting a special computation. This special computation may require new and important coding efforts if the platform changes or evolves. So, for each new indicator on the one hand, and for each learning platform on the other hand, we have to change the code for the “Specific Computation” part or/and for the “data pre-processing” part.

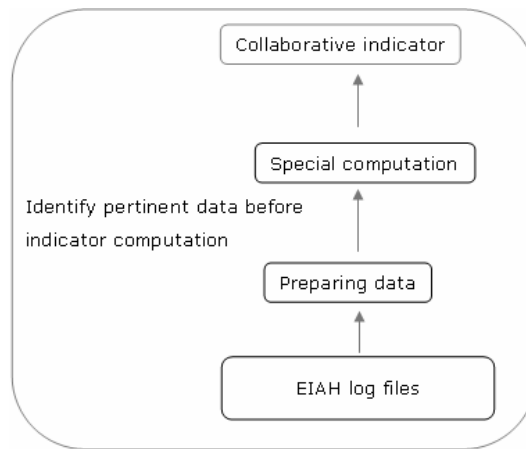


Figure 11 • Ad-hoc indicator computation

We describe and compute an indicator with/using a transformation sequence of trace models that is expressed in the generic transformation language of the TBS. This transformation sequence is first directly applied to models but not to data (not yet available). The four following steps are used to describe and to compute an indicator using a TBS:

Step 1: To build a new indicator "I" in the TBS, we propose to define a trace "I" and its computation rule $R_i(X_i)$ where X_i are all the variables we need to compute the indicator. At the beginning, there is no instance associated with the indicator model, just an empty set of instances before starting the next step (Fig. 12).

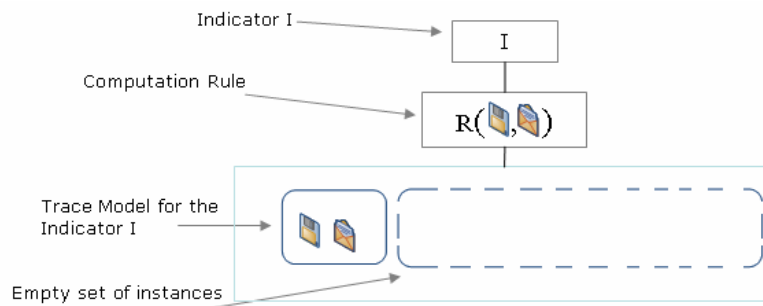


Figure 12 • Proposing a model for a new trace indicator "I"

Step 2: In this second step, the user provides a transformation sequence of a given trace model. The identification of a transformation sequence for the indicator "I" provides a transition path from the primary trace model to the indicator model. We have always a set of empty instances, and TBS operators can build this sequence. Fig. 13 illustrates how we build a transformation sequence, and shows the relationships between the primary trace model and the indicator model. Section 3 will give concrete examples using operators we defined.

Step 3: The third step called "data pre-processing" generates the various instances of the primary trace models. The result of this step is the primary M-Trace (models and instances): It is also called the *trace collecting* step. Fig. 14 shows an example of trace collecting for the primary trace.

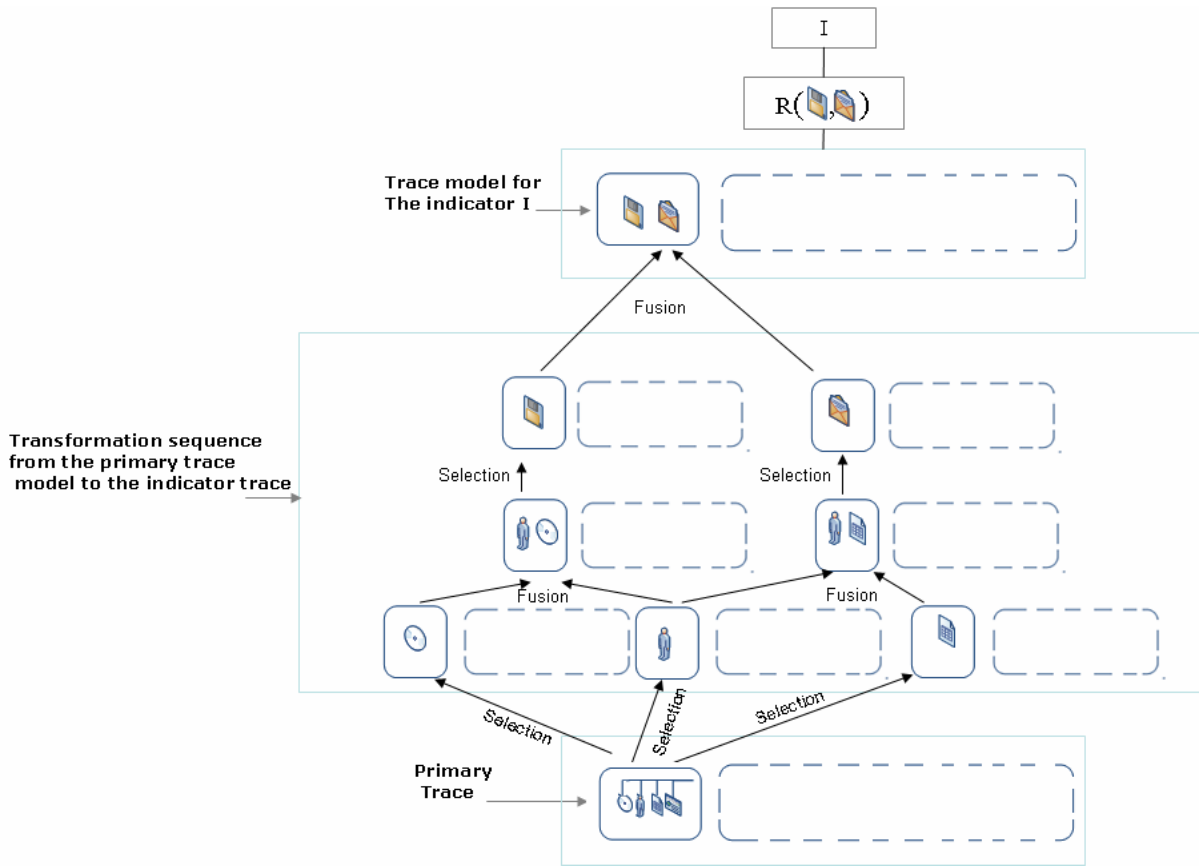


Figure 13 • A transformation sequence from the primary trace to the indicator model

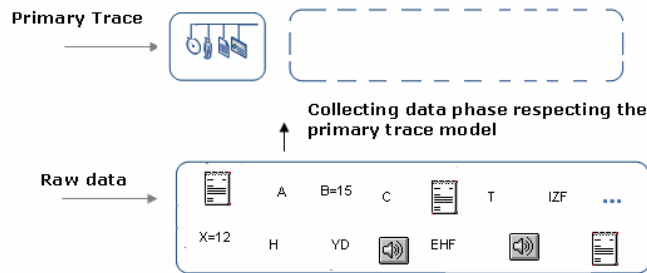


Figure 14 • Preparing data: instantiate the primary trace model

Step 4: In the last step, we execute the transformation sequence (defined in step 2) from the primary trace (instantiated in step 3) to instantiate all the intermediate models through propagation of transformed data across the transformation paths. Transformations spread up in intermediate transformed traces until the trace which is directly associated with the indicator. The result of this last step is the "I" indicator M-trace (model and instances). Fig. 15 shows the execution of the transformation sequence from the primary trace to the indicator model. The instances of the "I" indicator trace are used as input data for the computation rule "R_I".

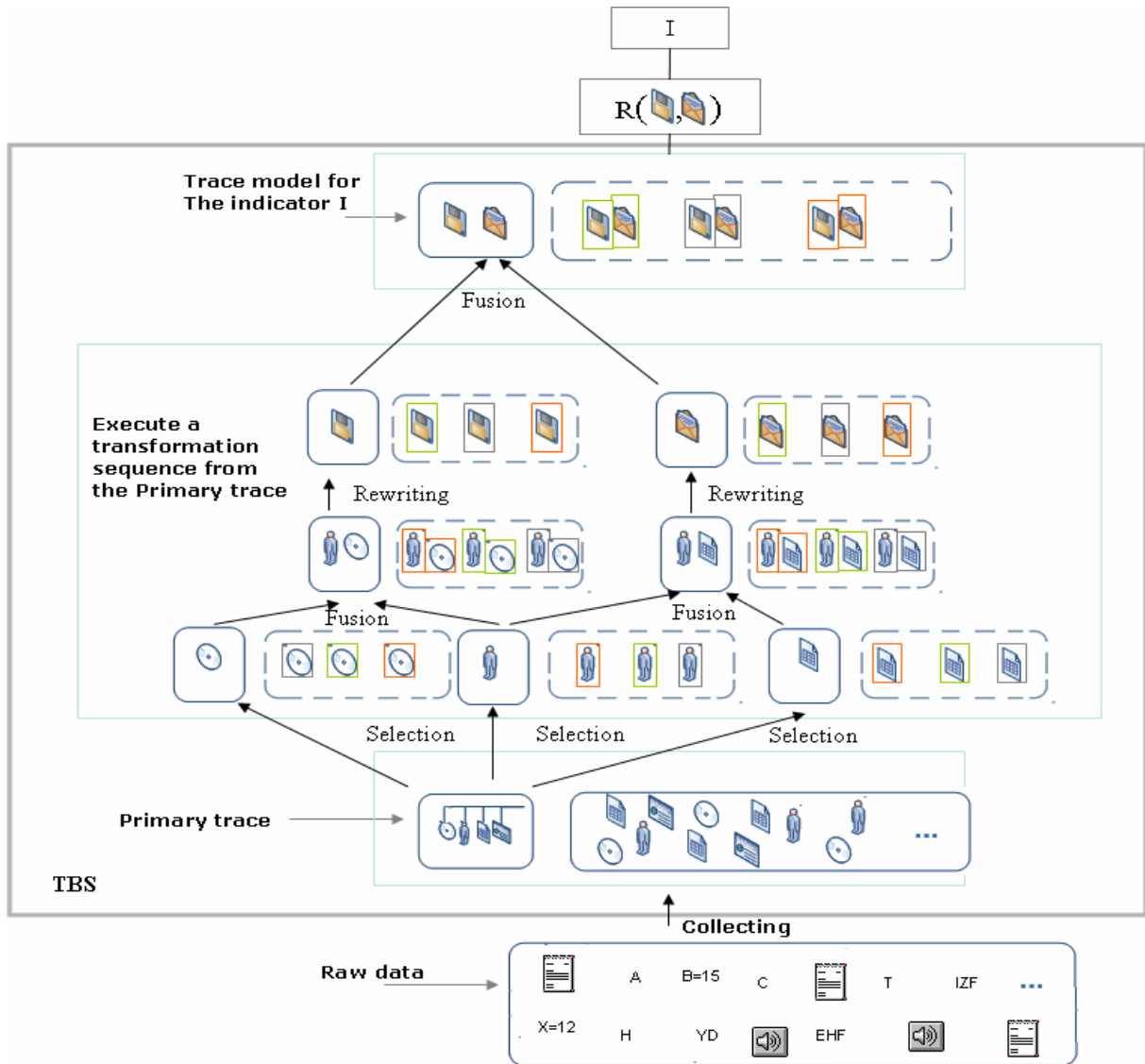


Figure 15 • Running the transformation from the primary trace

Our method allows model transformation and reduces the effort to compute indicators. It minimizes the gap between indicators and tracing sources, because we focus on the model transformation, and not on a special computation associated to each indicator in CSCL. Fig. 16 compares the indicator computation process between ad-hoc methods (that we presented earlier) and our method (TBS).

	Collecting data				Preparing data				Indicator computation			
	Ad Hoc		TBS		Ad Hoc		TBS		Ad Hoc		TBS	
	What	Who	What	Who	What	Who	What	Who	What	Who	What	Who
First indicator	Coding	Com.D	Coding	Com.D	Inter. Calc.	Com.D	Trace Trans	Educ Com.D	Calc Final	Com.D	Formule.	Educ
New indicator with Same platform	Coding	Com.D	-	-	Inter. Calc	Com.D	Reuse Trans.	Educ (Com.D)	Calc Final	Com.D	Formule	Educ
same indicator With new platform	Coding	Com.D	Coding	Com.D	Inter. Calc	Com.D	Reuse Trans.	Educ	Calc Final	Com.D	Formule	Educ

Figure 16 • Comparison between the ad-hoc calculation and our method oriented transformation's traces models (Educ: Educator; Com.D :Computer Designer; Inter.Calc : intermediate computation, Trans: Transformation).

To illustrate the benefits of our method (based on models driven engineering), we propose to detail the indicator creation steps in three different cases: (1) creating a new indicator from scratch, (2) creating a new indicator based on an existing indicator, and (3) adapting an existing indicator to another platform.

2.3.1. First case: Building a new indicator in a TBS from scratch.

In this first case, let us build (with the help of the TBS) a new indicator "Active Agent" [15] with Moodle as tracing source. Being a creation from scratch, we will follow the four steps presented before: (1) proposing a trace model for the indicator, (2) defining the associated transformation sequence, (3) preparing the data collecting phase, and then (4) feeding the different models across the transformation graph. Fig. 17 shows an example of how to build the indicator "Active Agent" from the Moodle platform and illustrates the flexibility of our method, and the transformation graph from the raw data to the indicator value without using a specific computation.

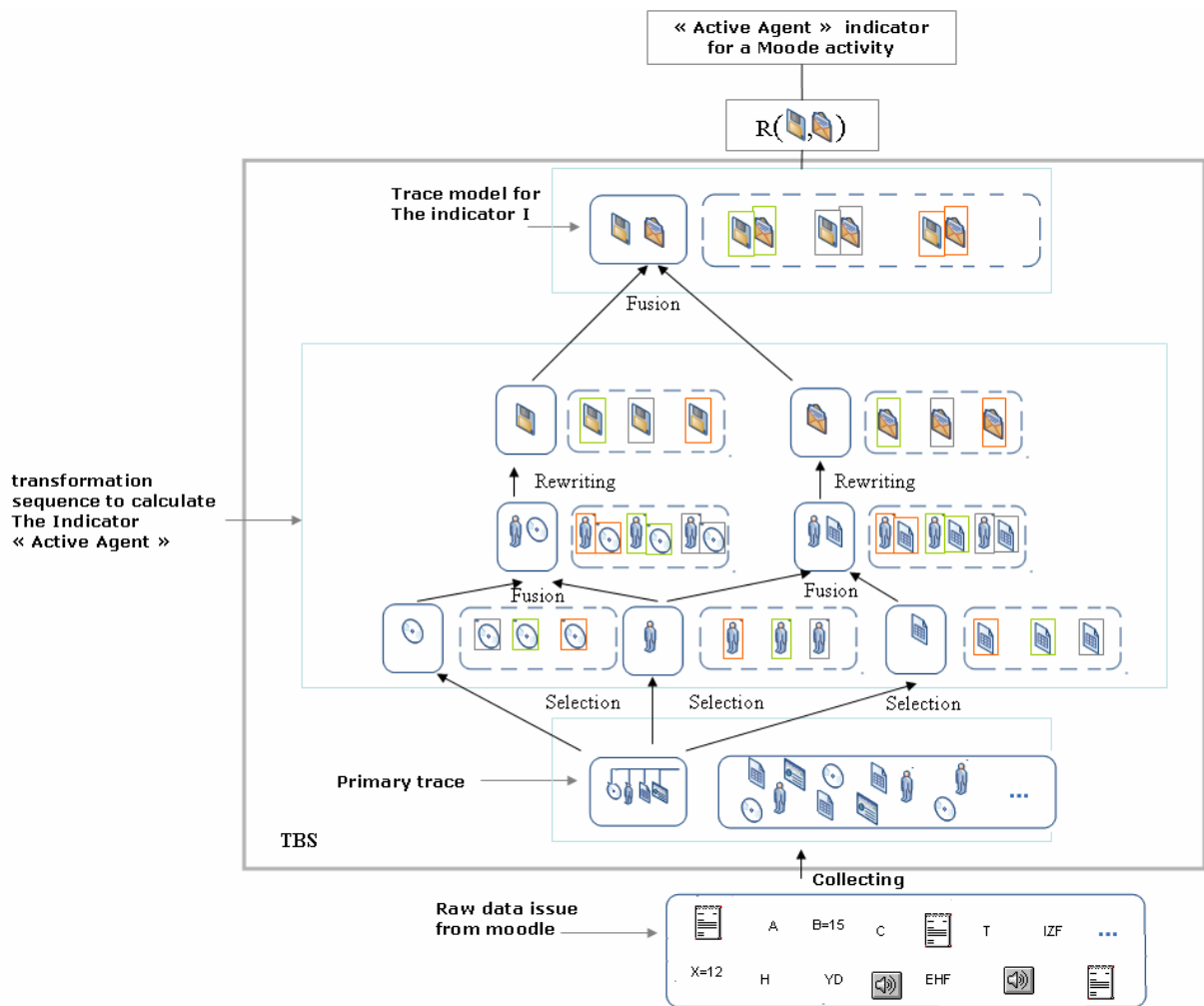


Figure 17 • Build from scratch the new indicator "Active Agent" for an activity taking place on the Moodle platform

2.3.2. Second case: calculating a different indicator by using a Trace Model for the same platform, existing in the transformation library

Principle

In this case, we compute an additional indicator in the TBS using the same platform. It means that we have already built indicators in the TBS, and we have saved the associated transformation sequences in a library. This second case differs from the first case in that the process reuses a transformation sequence available in the library.

Example: building the indicator I_2 "Division of labor" [14] on Moodle, considering we already have built the indicator "Active Agent" on Moodle.

Steps:

- Propose a trace model for this new indicator.
- Search in the models library of indicators a trace model indicator similar to this new model. In our example, we suppose that the indicator model "Active Agent" created in the illustration of the first case is similar in its structure to the indicator "Division of Labor". The measure of similarity between models is out of the scope of

this paper.

- Modify the transformations sequence associated to the selected model. This modification allows the reuse of an existing transformation sequence in the TBS, and then, derives the transformation associated to the new indicator.
- Execute the new transformation sequence to obtain instances for the new model.

Fig. 18 explains these steps. We can see in this case that we just modify a part of the existing transformation sequence (the shaded part of the sequence in Fig. 18) to get the new model.

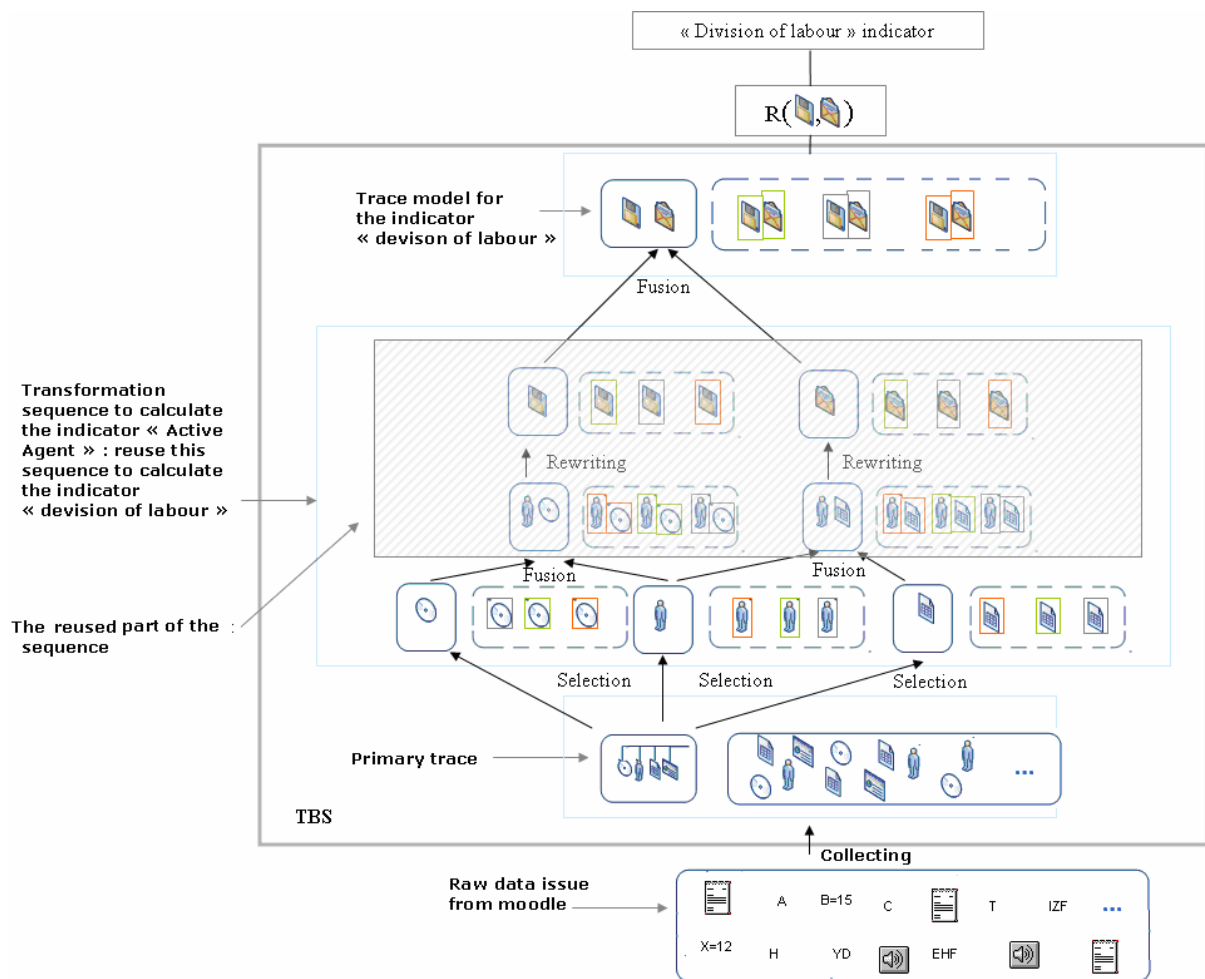


Figure 18 • Calculating a new indicator "Division of Labor" in the TBS from the existing sequence "Active Agent"

Fig. 18 emphasizes the part our TBS deals with: from the primary trace to the Indicator model, leaving raw data and output values outside the system. TBS becomes alike a function that has the raw data for input and has the indicators value for output. Inside the TBS box, there are only trace model transformations leading to the desired indicators' model.

2.3.3. *Third case: Adapting an existing indicator to an other platform*

In the third case, we want to reuse an existing indicator in the TBS, in order to adapt it to another learning platform. Because TBS is independent from the learning platform, we try this time to generate the indicator value (built previously) considering another learning platform as the tracing source. For example, we build the transformation sequence of the indicator "Active Agent" on the platform Moodle, and we rebuild this indicator for the WebCT platform. In this case we have the indicator model we want to compute as well as its transformation sequence. This transformation sequence is not reused directly but requires some modification steps:

- Find the indicator model we want to compute in the models library, for example, the indicator model "Active Agent",
- Load the transformation sequence associated with this indicator,
- If the learning platform changes then the primary trace model has to be changed, because it is linked to the various activities offered by the new learning platform. Thus, we have to change a part of the transformation sequence. As shown in Fig. 19, we modify only the shaded part. The rule used to compute the indicator remains unchanged. This strategy allows us to adapt indicators computation to different learning platforms, which leverages the method we propose.

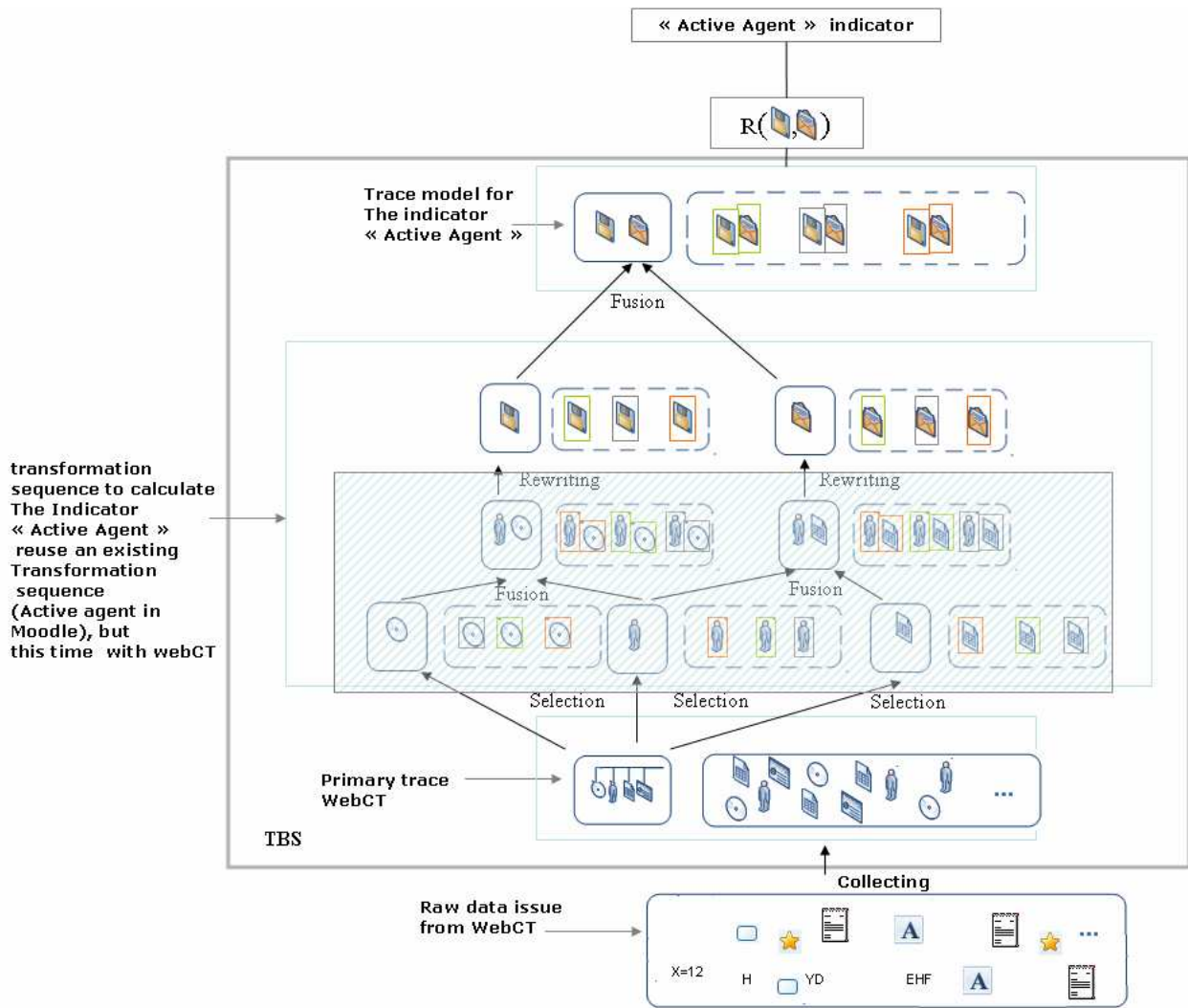


Figure 19 • Calculating the same indicator "Active Agent" for a different platform (Moodle / WebCT)

In this section, we have presented a method to compute indicators from the transformation of M-traces based on engineering driven by trace models. We have identified three possibilities: a new indicator from scratch, a new indicator for the same platform, and an existing indicator for a different learning platform. The next section shows this method implementation with a case study on the specific platform Moodle.

3. IMPLEMENTATION

This third part of the paper illustrates an implementation of what we have presented in the previous sections: TBS, transformation operators and transformation sequences of trace models. The implemented Trace-based system uses the platform Moodle [22] as tracing source, but its architecture is open to other learning platforms. Section 3.1 presents the data collecting phase from Moodle to generate a primary M-trace. Section 3.2 presents the transformation sequence of the traces with generic operators. Section 3.3 illustrates some indicators models.

3.1. Collecting data phase for collaborative activities

We propose here a specialized collecting phase for various collaborative learning activities. We are interested in synchronous and asynchronous activities, where actors work together on the same resources. The resources used in Moodle are: Wiki, chat, text resources, and private messages.

3.1.1. Example on Moodle

In a previous work [23], we proposed a primary trace model for Moodle. We extended this model to support additional actions that manage contacts. This additional information can feed indicators, which help researchers to study learners' collaborative behavior.

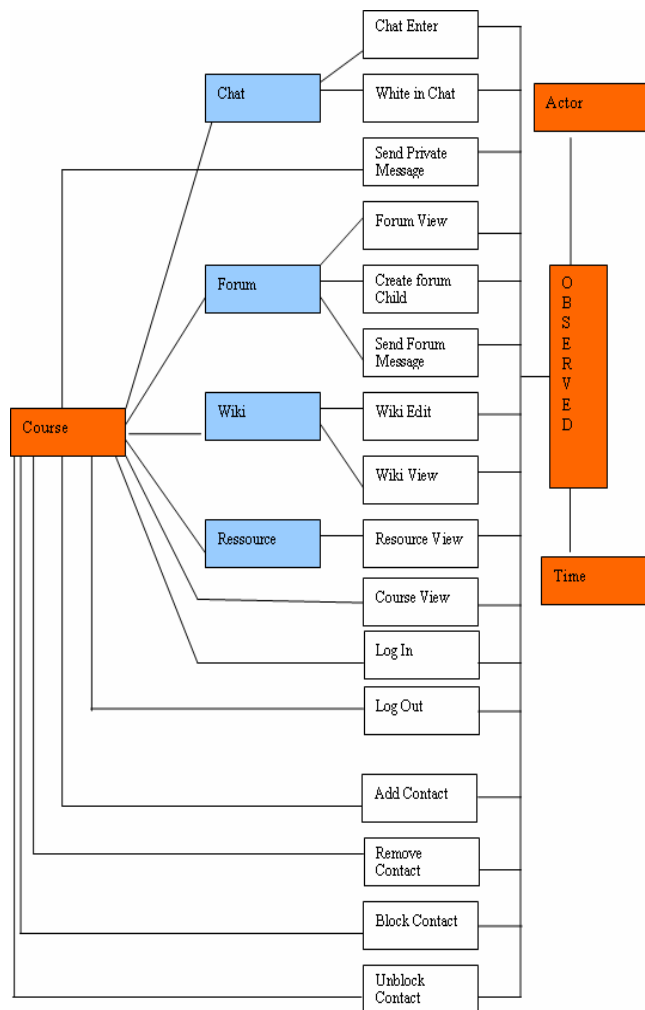


Figure 20 • Primary trace model from Moodle platform

Our tool provides a selector to choose the list of elements the user wants to collect. This choice depends on the trace models the user wants to build as a basis to compute the target indicator. Connected to the Moodle database, the tool imports necessary data, and instantiates the trace model in OWL format. This format follows the syntax of the Jena parser [24]. We provide a graphical user interface (GUI) to select data. As shown in Fig. 21, the GUI gives a list of ob-

servable actions the user can pick up. Here, the user chooses only 5 from 18 possible actions. The collecting module collects data from tracing sources and instantiate the primary trace model. The result of this collecting phase is a primary trace in OWL format.

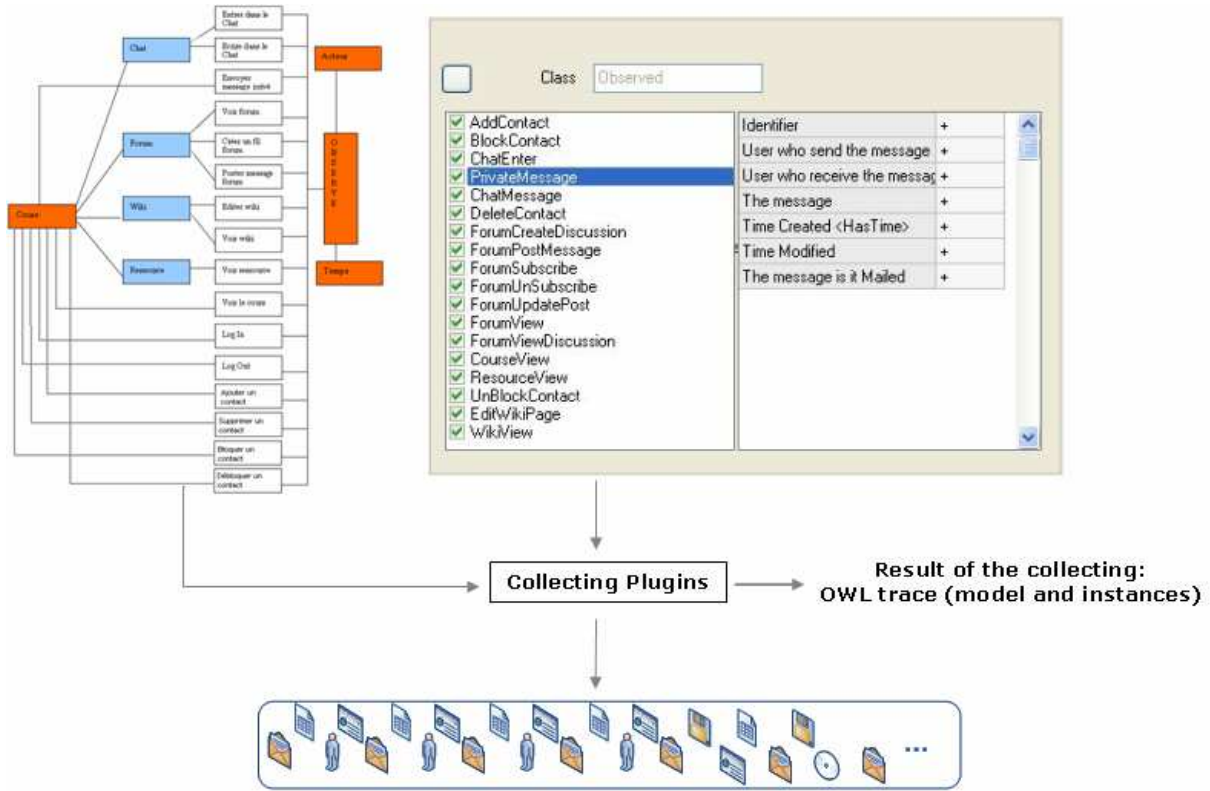


Figure 21 • User interface used to collect observed actions

3.2. The transformation operators of the implemented TBS

We propose a list of operators to build generic transformations. These transformations are independent from the collaborative learning platform. We classify these operators into three categories presented in the following subsections (3.2.1-3).

3.2.1. Operators that do not modify the trace model

These operators do not modify the model, but only the trace instances. This category includes: Matching, Selection and Fusion of two traces instances (these instances have the same trace model).

3.2.1.1. The Matching operator

This operator identifies a sequence of observed actions using patterns. To find a pattern in such a sequence, we use an algorithm proposed in [25]. This operator can be used in the following way:

$TraceX := \text{Matching}(\text{pattern}) [TraceY];$

Where $TraceX$ is the resulting trace, $pattern$ is the pattern used to define our research criteria, while $TraceY$ is the

source trace where the matching function looks for episodes that fit the given pattern. For example, we can say that an *effective* entry in a Chat is sequence of actions such as "Chat Enter" followed by writing a message in the Chat "ChatWriteMessage". Fig. 22 illustrates the Matching operator on this example. Let $A=ChatEnter$ and $B=ChatWriteMessage$, X and Y are any other observed action. If we consider "XYXYXYAYXXYBXXXBXYAYXYXXBXYXX" as an input sequence of actions, then, the resulting (output) sequence with a matching (A, B) would be a list of "AB".

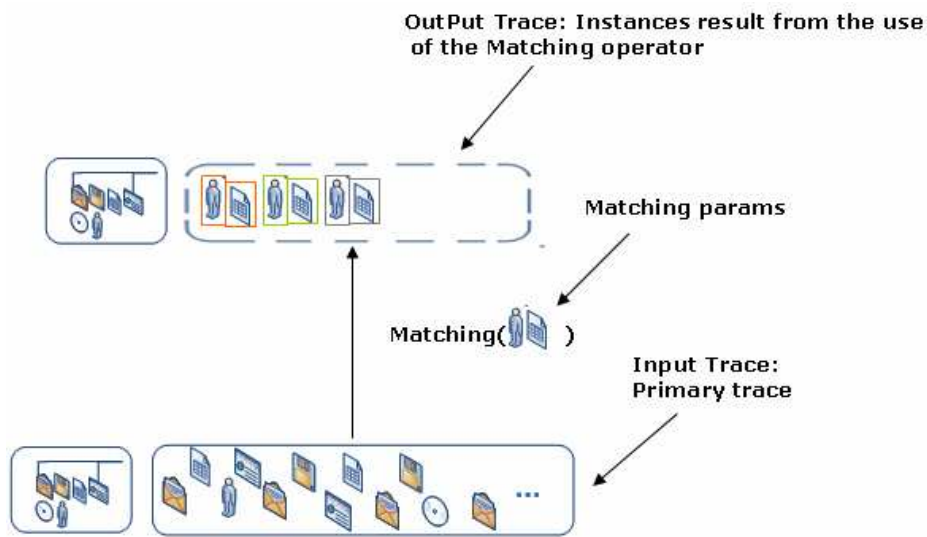


Figure 22 • Example of Matching

3.2.1.2. The selection

This operator works as a filter on the input instances. The selection criteria may use the following attributes: *time*, *action type*, *associated tool*, and *actor*. The syntax of this operator is:

$TraceX := Selection (Criterion) [TraceY];$

TraceX is the resulting trace; *Criterion* is a logical expression based on a general instance attributes; and *TraceY* is the source where instances are selected. Fig. 23 shows how to select all observed actions associated with the ActorX from the input trace.

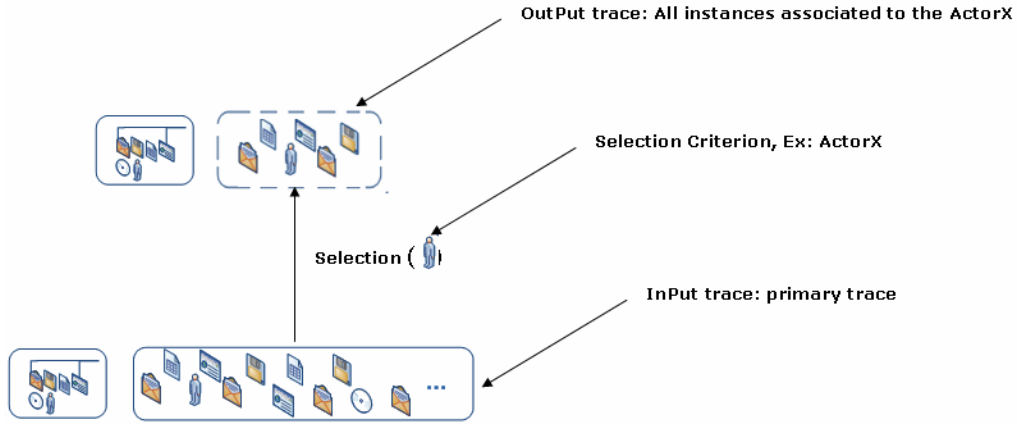


Figure 23 • Example of selection

3.2.1.3. The fusion of two traces:

This operator concatenates two instances of traces (TraceX and TraceY) into a new trace (TraceZ). The two traces must have the same model. The operator is used as follows:

$\text{TraceZ} := \text{Fusion}(\text{TraceX}, \text{TraceY})$ / where TraceX and TraceY have the same model.

To illustrate how we use this operator (Fig. 24), we consider two traces: *E-mails sent* and *E-mails saved* by ActorX. We combine these two traces by fusion. We call the result of this fusion: *Emails-Manipulations* associated to ActorX.

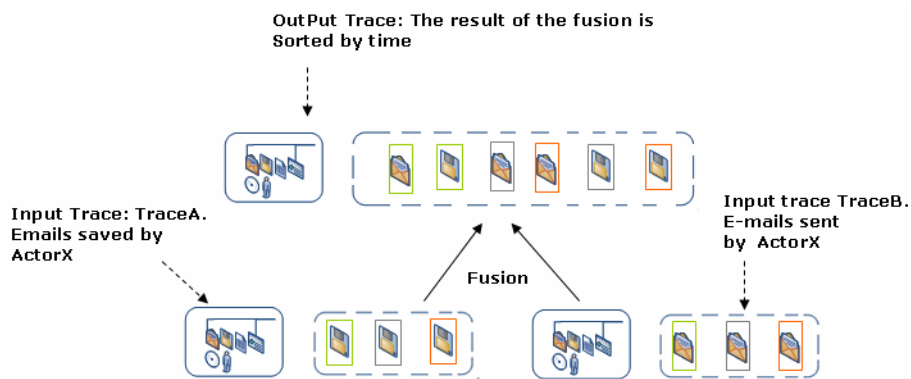


Figure 24 • Example of a fusion of two traces

3.2.2. Operators that modify a Trace model

These operators change the model structure. The trace model is structured into classes. We propose two operators: *rewriting* the model and *pruning* the model.

3.2.2.1. Rewriting models

This operator rewrites the name or the attributes of a model (class). This modification is propagated into instances structure by changing its labels. The result of this rewriting is a new model different from the initial model. The operator is used as:

$$\text{ModelY} = \text{Rewrite}(\text{ModelX}, \text{ClassNameA}, \text{ValueNameB})$$

Where: ModelX is the initial model, ModelY is the result Model, ClassNameA is the class to rewrite in ModelX, and ValueNameB is the newname of ClassNameA in ModelY (ClassNameA will not exist in ModelY, and is replaced by ValueNameB). This operation is shown in Fig. 25. We can change the name of the class "Record" of the input trace model to "Save" in the output trace model.

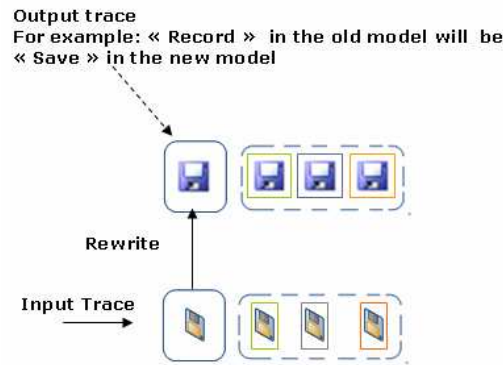


Figure 25 • Example of a Rewrite operation

3.2.2.2. Pruning model

This operator removes classes or class attributes in a model. We give as parameters for this operator the list of what we have to keep in a model. The operator is used as:

$$\text{TraceX:} = \text{Pruning}(\text{TraceY} . (\text{List of classes or attributes to keep})).$$

Fig. 26 gives an example of pruning where only the attributes of the source model are kept in the result model.

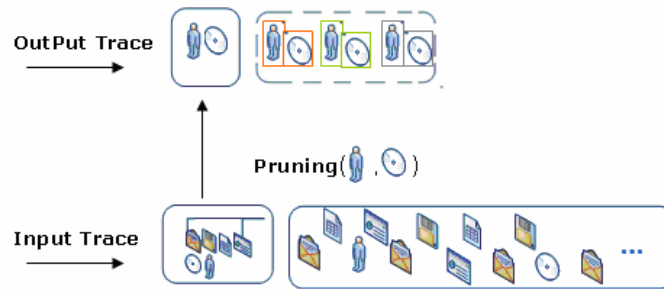


Figure 26 • Example of pruning

3.2.3. Operators used for the indicators computation

We also propose two operators *Count* and *Sort* used to compute indicators. These operators are not integrated in the TBS. However, we can use them to build the computation rule (*Ri*) from the indicator M-trace.

- **Count:** it counts the number of instances in a trace. The result is a positive integer or zero.
- **Sort:** it sorts observed actions using as criteria: time, observed type or observed tools.

4. TRANSFORMATIONS ILLUSTRATED IN OUR TBS CONNECTED TO MOODLE

4.1. The implement TBS in Moodle

All operators mentioned in Section 3.2 have been included in our implemented TBS. Fig. 27 illustrates the *selection* operator. On this example, we selected “User4” actions from the primary trace Moodle. The resulting trace instances “TrackUser4” contain all obsels related to this user.

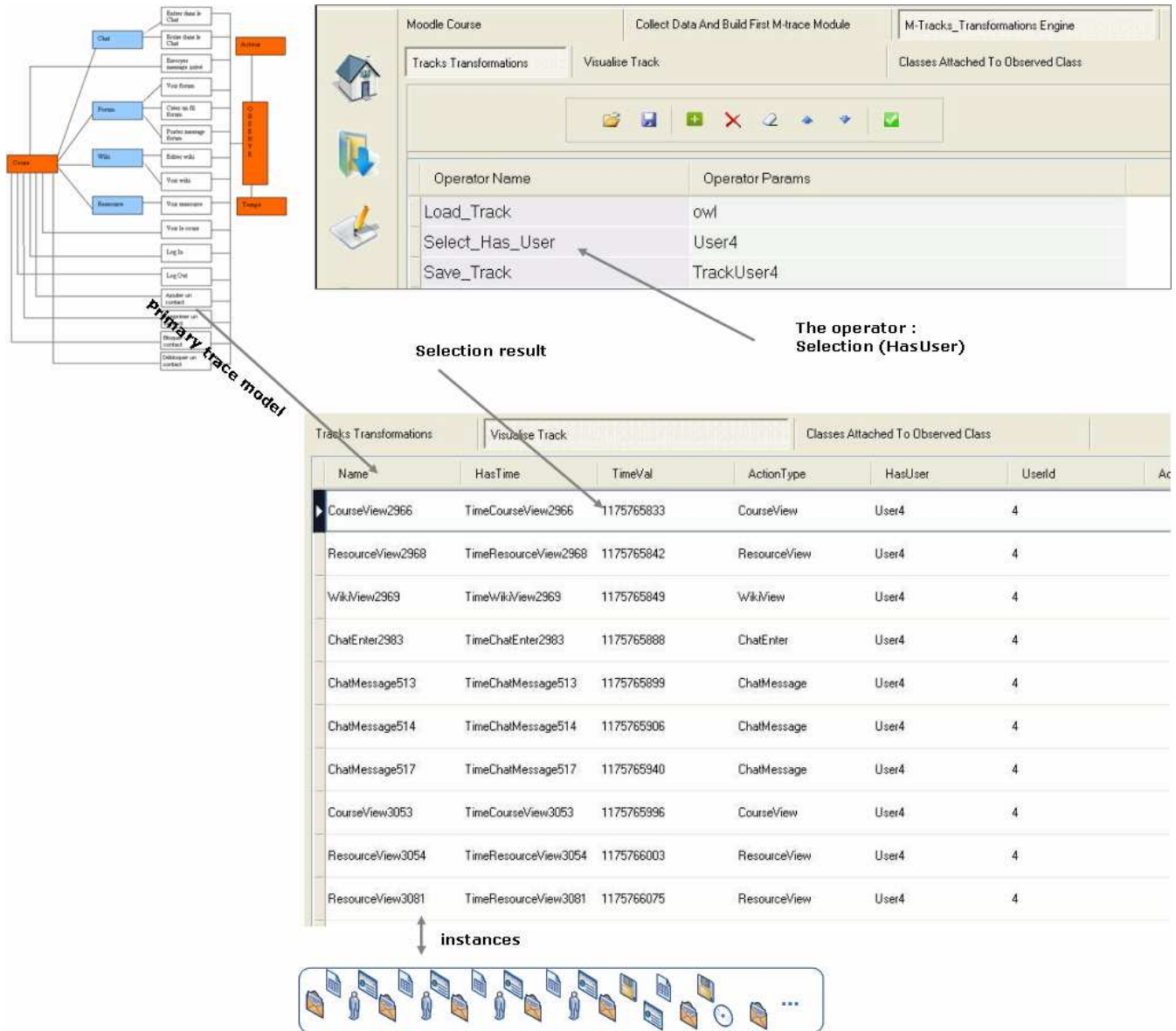


Figure 27 • Selection operator applied to a primary trace (from Moodle)

Fig. 28 shows the *matching* operator. In this example, the pattern contains observed actions like “write a private message” and “write in a chat” (PrivateMessage, ChatMessage).

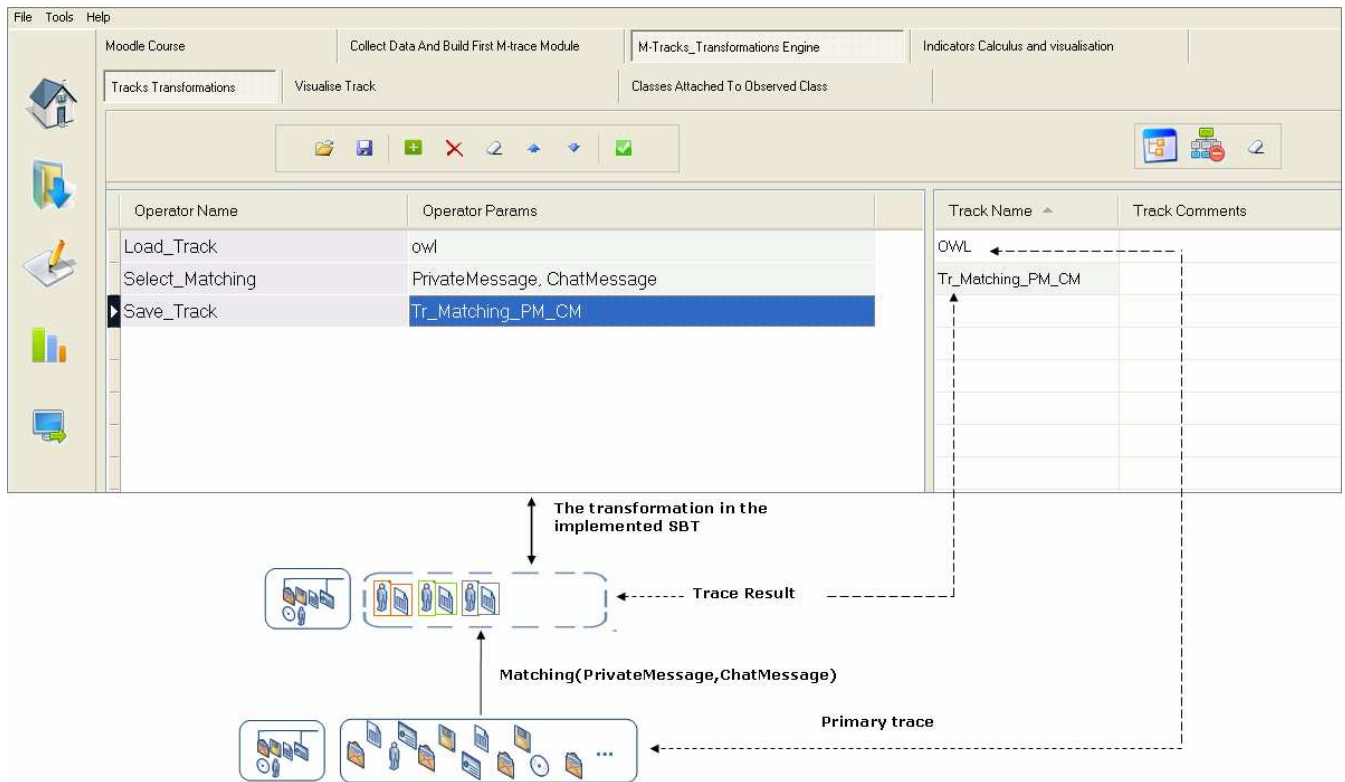


Figure 28 • Matching operator applied to a primary trace (from Moodle)

4.2. The indicators computation

We show in this section how to apply operators on the M-trace to build transformation sequences, and then generate M-trace for collaboration indicators. In order to show how easily our implemented TBS can build indicators, we picked some indicators presented in Section 1 to illustrate here their transformation and concrete computation.

4.2.1. Example1: The proportion between 2 different types of actions for the same actor over a time interval.

Let us remind that the computation rule of the proportion between two actions types A and B is: $\text{Proportion}(A,B) = \frac{NT_A - NT_B}{NT_A + NT_B}$, where NT_i is the Number of actions of Type i. In our case: an action is related to an actor. The transformation sequence from the primary trace to the indicator is explained in Fig. 29. We propose to make a *selection* on all observed actions related to the ActorX, and then select only those occurring in a chosen time interval. Then we prune the model to keep only the names and the type of the observed actions. Then we extract (using 2 selections) observed actions of types "A" and "B", which are used as input data for the computation rule, using the operators "Count" and "Proportion". This generic transformation sequence is not the unique solution. For example, selections on actor and time intervals may be built in a different order. Here, our aim is not to provide the transformation sequence for this indicator, but rather the authoring tools to build such sequences.

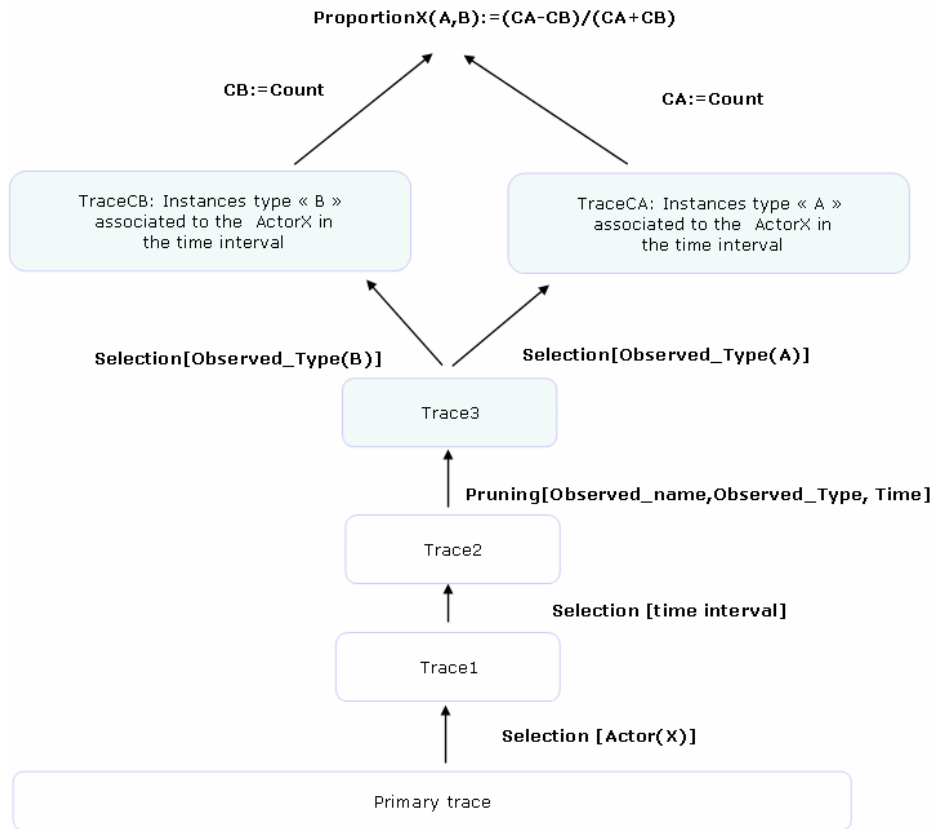


Figure 29 • Generic transformation Sequence to compute the indicator: "Proportion between two observed types A and B"

In the implemented SBT, we show how to build the proportion between the observed type "private message" and "chat message" made by the User15. Fig. 30 illustrates the construction of this sequence in the implemented SBT.

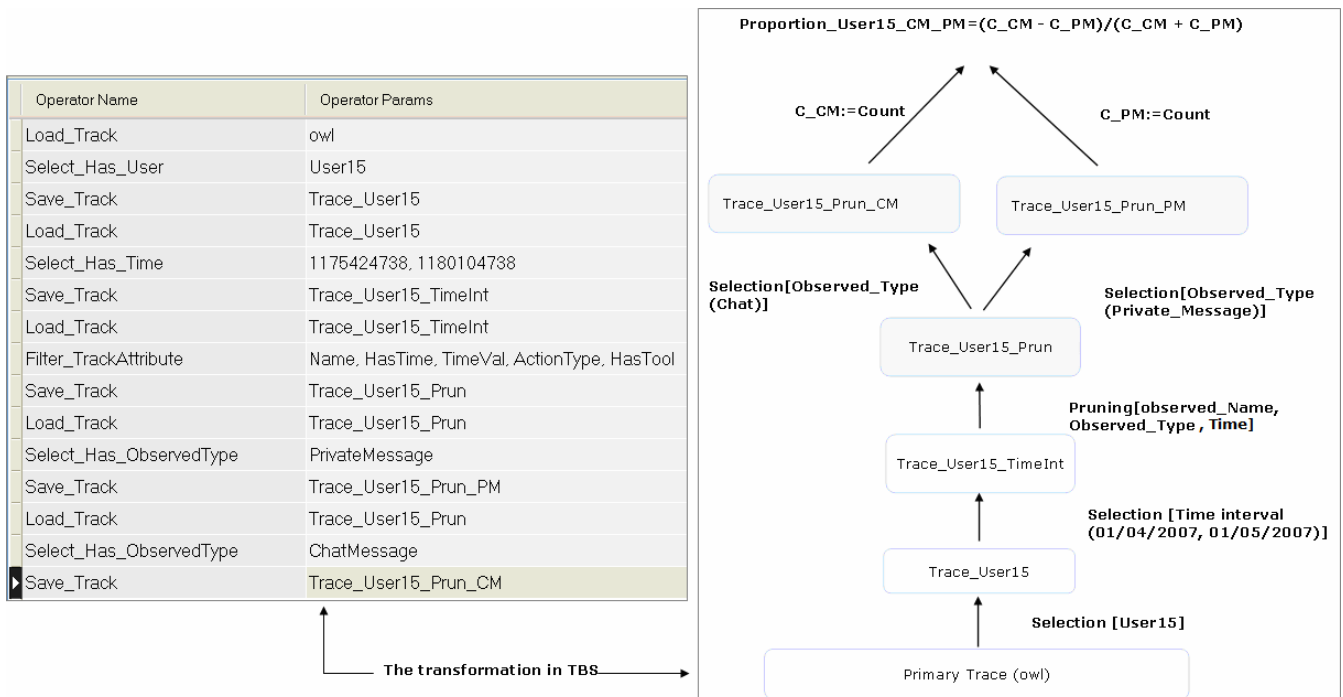


Figure 30 • Transformations sequence computing the proportion of private messages among all chat messages of User15

We propose to build the computation rule in a toolbox taking the transformed traces as input. This toolbox offers arithmetic operators; visual forms “Pies and histograms”; and calls the *Count* operator. Fig. 31 shows the computation rule implementation for the indicator *Proportion*.

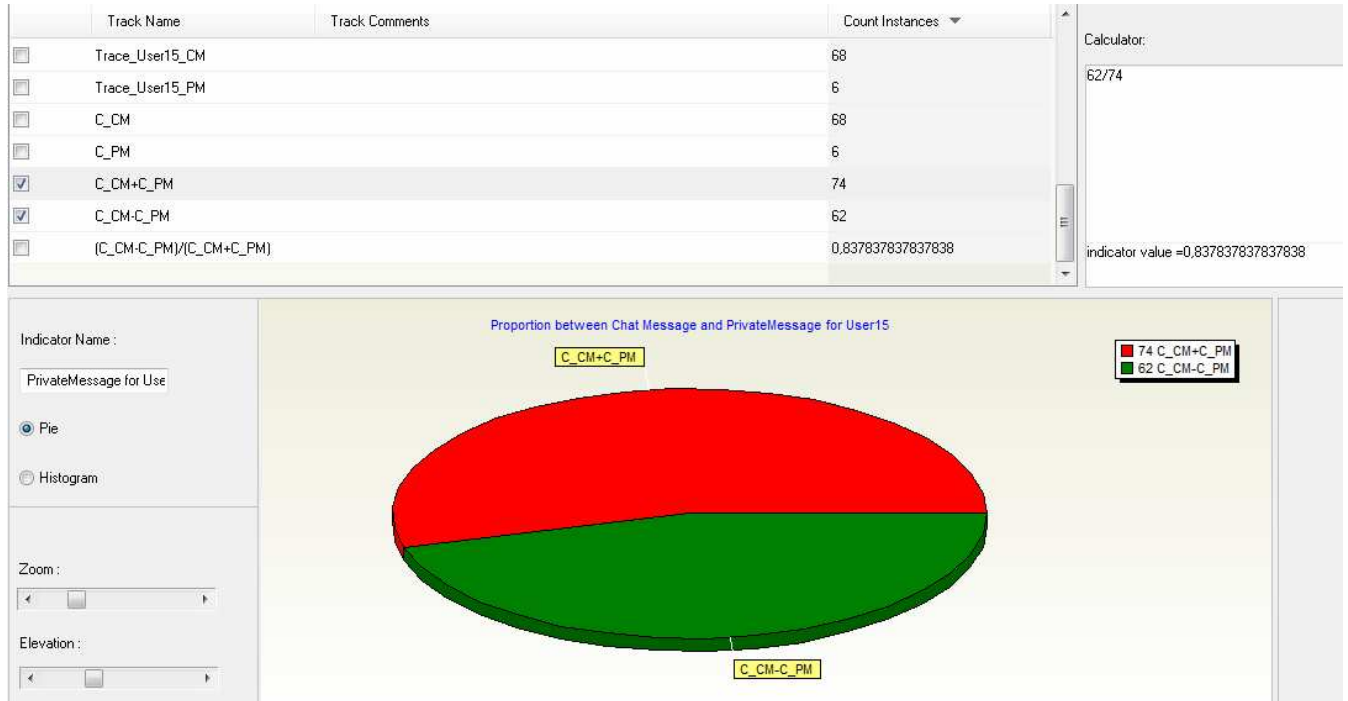


Figure 31 • Compute the Proportion indicator value of the last example in the implemented TBS

4.2.2. Example2: The division of labor

The division of labor computation is based on the sum of the differences (SD) of two instances associated with each user on each tool. There is no specific rule in the CSCL (Computer Supported Collaborative Learning) literature to compute this indicator. We propose the following transformation sequence to compute the indicator. We use the selection and pruning operators to build the sequence (Fig. 32).

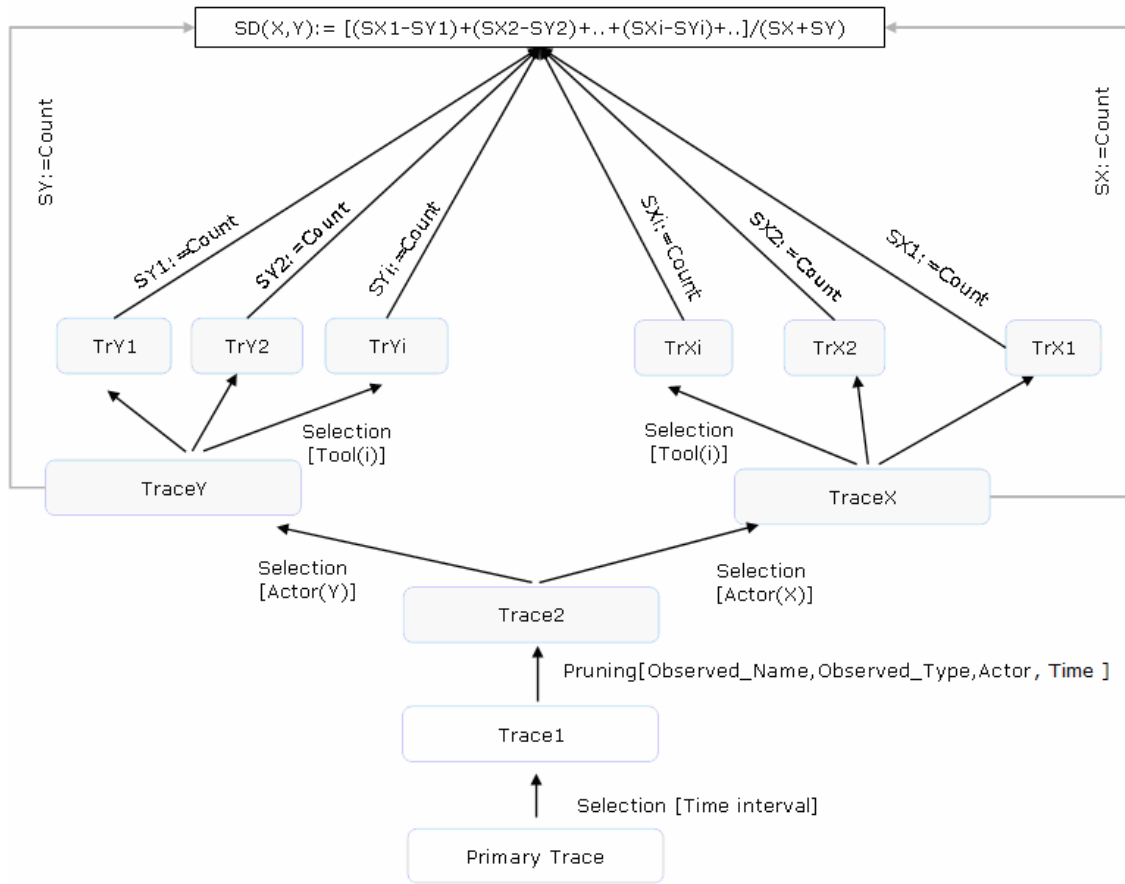


Figure 32 • Generic transformations Sequence to compute SD part of indicator: "Division of labor" between two actors ActorX and ActorY

The transformation sequences for ActorX and ActorY being the same, we can reuse the first one to build the second one, thanks to the TBS *ability to reuse* existing transformation sequences in other transformations. Fig. 33 shows the SD computation between actors "15" and "16" on tools "ToolChat1" and "Private Message".

Operator Name	Operator Params	Track Name	Track Comments
Save_Track	Trace1	OWL	
Load_Track	Trace1	Trace1	
Filter_TrackAttribute	Name, HasUser, TimeVal, ActionType, HasTool	Trace2	
Save_Track	Trace2	Trace_Actor15	
Load_Track	Trace2	Trace_Actor16	
Select_Has_User	User15	Tr_A15_PM	
Save_Track	Trace_Actor15	Tr_A15_CM	
Load_Track	Trace2	Tr_A16_PM	
Select_Has_User	User16	Tr_A16_CM	
Save_Track	Trace_Actor16		
Load_Track	Trace_Actor15		
Select_Has_ObservedType	PrivateMessage		
Save_Track	Tr_A15_PM		
Load_Track	Trace_Actor15		

The transformation in TBS

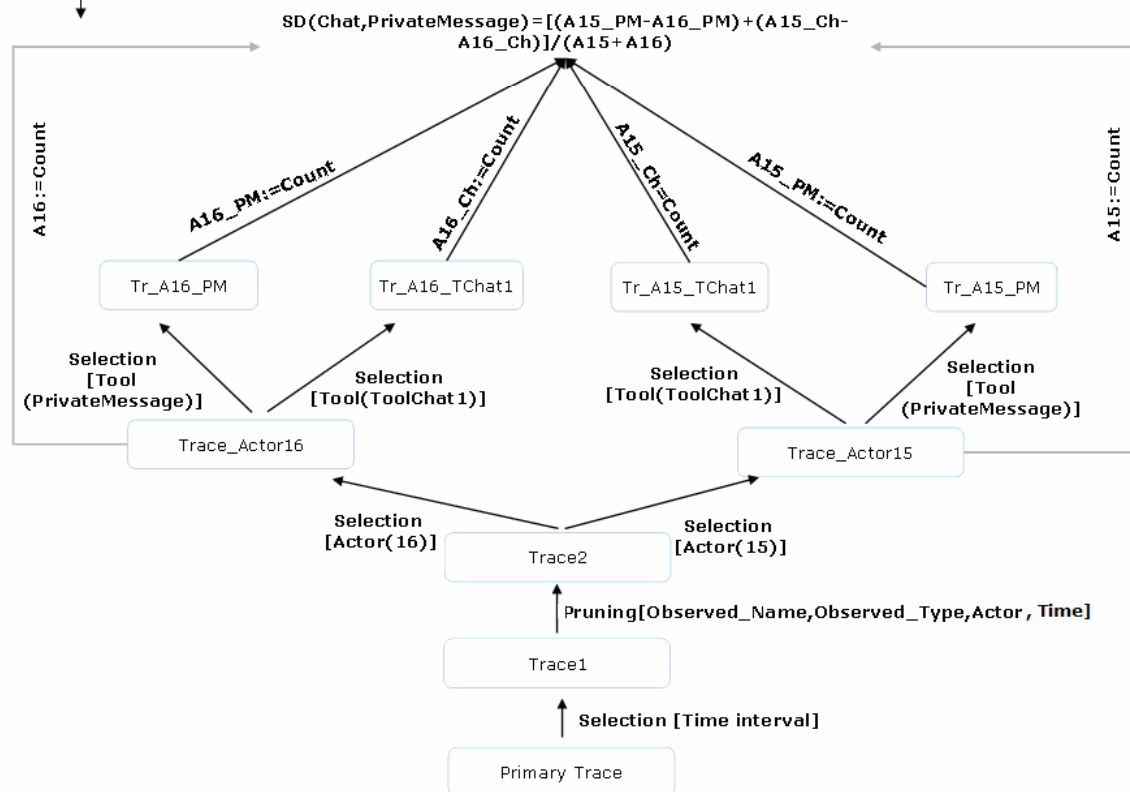


Figure 33 • Example of a transformation sequence to compute the SD part of the indicator “division of labor” (between actors 15 and 16) on the tools: “ToolChat1” and “Private Message”. Save_Track and Load_Track are two operators used by TBS-IM to keep m-tracks in memory and clarify which track the current transformation is applied to. These m-tracks management actions are not transformation.

4.2.3. Exemple3: An actor who reads his own contributions (auto self read) on a resource (I)

This indicator explains how often an actor reads his own contributions. In CSCL, there is no fixed rule to compute this indicator. We propose here to use a pattern with the matching operator. The pattern is defined as $[A+, B^*]$ where A represents the action type: "Edit Tool" and B is "Read in the same tool". The "+" means that the actor has modified his/her contribution at least once, while the "*" means that the actor may have read 0 or more times this contribution.

For example, the pattern ABB means that an actor has edited his/her contribution once, and read it twice. The pattern is illustrated by an automaton in Fig. 34.

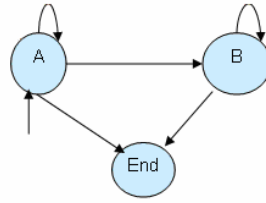


Figure 34 • Pattern to identify auto self read indicator

The associated transformation sequence for this indicator is based on this pattern, and is shown in Fig. 35. We propose in this figure, the same indicator computation with another transformation based on the selection and fusion operators. The fusion requires that the number of instances “Editing” is greater than 0.

This example shows the existence of some equivalence between transformation sequences. We can build two different sequences to compute the same indicator which also demonstrates the expressiveness of our method.

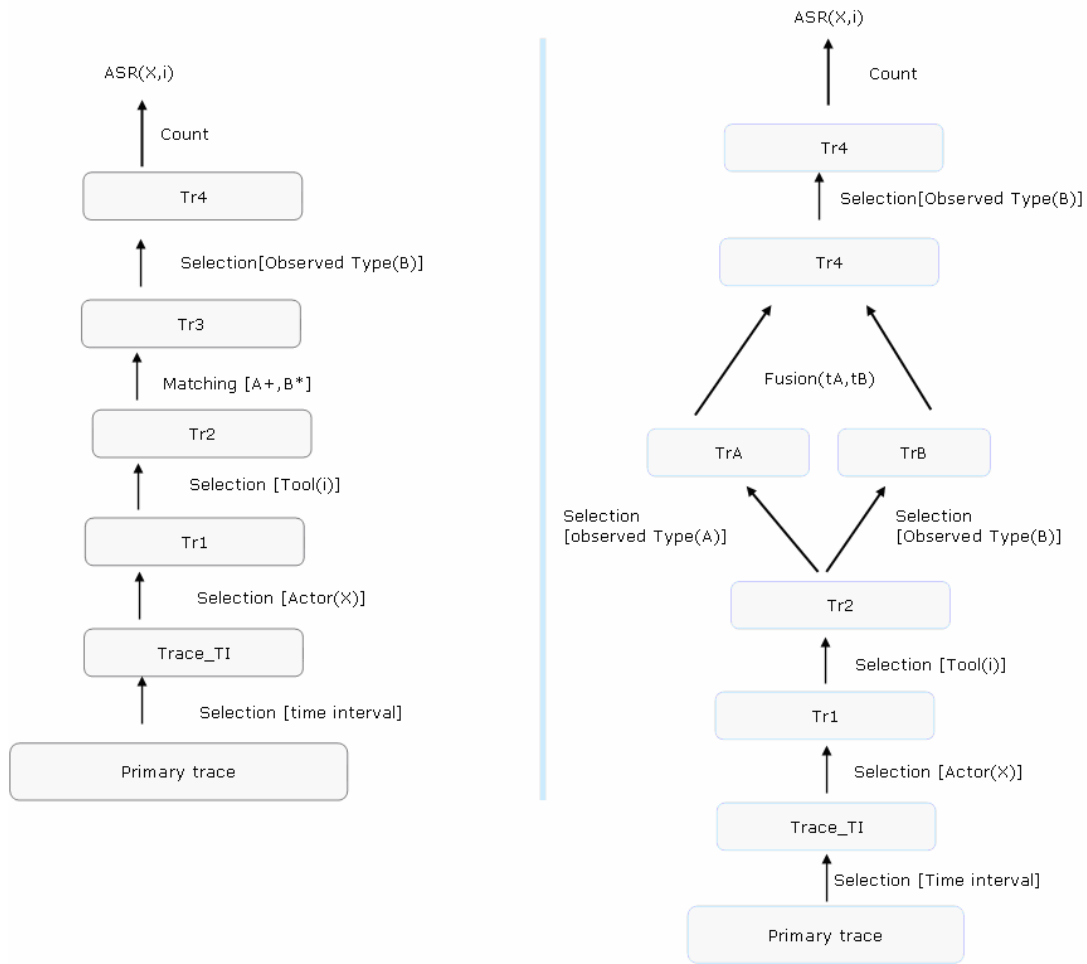


Figure 35 • Two sequences for the indicator (Auto-Self-read), using fusion or matching operator

We have chosen to implement the second transformations sequence (Fig 36) for the actor "2" and tool "ToolWiki1". The "A" in the generic sequence becomes "EditWikiPage" and "B" becomes "WikiView".

In this part, we have presented the implementation of our method with the proposal of generic operators that transform models or their instances. We have shown the existence of a certain class of equivalence between the transformation sequences, as well as the ability to reuse transformation subsequences in the same sequence or in new sequences. We have also shown the feasibility of our method with the implemented TBS which integrates a primary trace from Moodle. The indicators computation becomes an easy task, without going through the coding details.

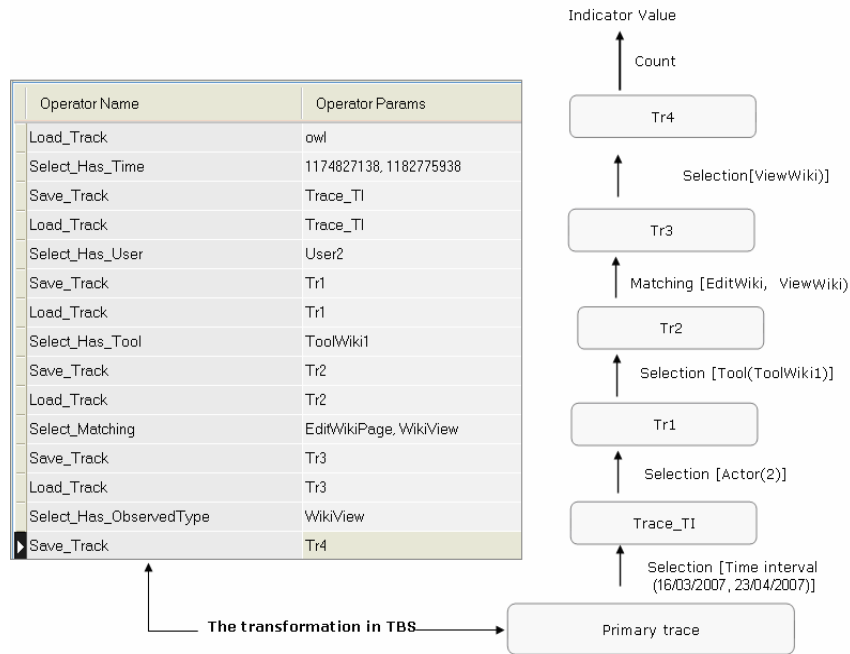


Figure 36 • Sequence changes for the actor 15 on tool Wiki in order to compute the readings of his own contributions on this wiki

4. CONCLUSION

We have presented a method and a tool to describe and compute collaborative indicators when using a Trace-Based System. The method is based on an MDE approach (trace models), and helps build transformation sequences of models to compute collaboration indicators. This method uses a Trace-Based System to manage the traces that are needed for computing indicators. The Trace-Based System also supports the reuse of models during the design and calculation steps. We propose an architecture and a tool to build and manage interaction traces used to compute indicators. It's essential and crucial for the proposed engineering to support the modelling process by facilitating models reuse for collecting observables as well as for the indicators computation themselves. Therefore, this work enables to describe explicitly the indicator construction (calculation expression, associated trace model and transformations' path from the primary trace) making ease the reuse of this knowledge and, moreover, proposes a concrete framework (a trace base system) to compute these indicators from interaction traces of the learning activity.

Then, we showed the benefit of this method compared to the ad-hoc indicators computation, where we compute indicator at an abstract level with transformations operators without using a specific coding. As recommended in [4], our method is widely independent from the learning platform. Indeed, it describes and executes the computation of a new indicator from the M-trace transformation sequences without using the learning platform. Only the initial collection of users' events is made specifically for the target platform, providing the primary modelled trace used by all transformation sequences to compute collaborative activities indicators.

The user can then reuse transformation sequences to compute the same indicators for other learning platforms, to ensure independence between the learning platform and the way to compute the collaborative activity indicator. The use and reuse of transformation sequences applied on the M-Trace is the originality of our proposed method. This method applies also to social systems that involve a big number of participants, as well as for collaborative activities in small groups (2-3 persons). The notion of modelled primary trace is generic and does not limit the number of learners. We must only associate an identifier with each learner, since we start with a primary M-Trace collecting event from any number of learners. We can also apply our method to compute individual activities indicators, but an important part of our work consists in studying collaborative activities indicators proposed in the literature, and define -for each of them- the abstract models of the collected traces. For the moment, we build a library of indicators with their associated trace models and focus on collaborative activities indicators. Publishing this library may be relevant for all learning platforms designers wishing to integrate it.

In a future work, our implemented TBS and the models transformations library will be tested by independent researchers with other learning platforms. We also plan to make the proposed transformation mechanism richer by adding new operators or by enriching existing ones and particularly the rewriting operator.

REFERENCES

- [1] A. Soller, A. Martinez, P. Jermann, M. Muehlenbrock, "From Mirroring to Guiding: A Review of State of the Art Technology for Supporting Collaborative Learning. IJAIED," *International Journal of Artificial Intelligence in Education*, vol. 15, pp. 261-290, 2005.
- [2] A. Mille and Y. Prié, "Une théorie de la trace informatique pour faciliter l'adaptation dans la confrontation logique d'utilisation/logique de conception," *Proc. 13èmes journées de Rochebrune –Traces-Enigmes-Problèmes*, Rochebrune, France, 2006.
- [3] A. Dimitracopoulou, "State of the art of interaction Analysis: Interaction Analysis Indicators. Interaction and Collaboration Analysis supporting Teachers' and Students' Self-regulation. (ICALTS)," JEIRP Deliverable D.26.1.1. Kaleidoscope NoE, pp. 153, December 2004.
- [4] A. Dimitracopoulou, "Computer based Interaction Analysis Supporting Self-regulation: Achievements and Prospects of an Emerging Research Direction," In Kinshuk, M.Spector, D.Sampson, P. Isaias (Guest editors). *Technology, Instruction, Cognition and Learning (TICL)*, 2008.
- [5] C. Reffay and T. Chanier, "How social network analysis can help to measure cohesion in collaborative distance-learning," *Procs. Computer Supported Collaborative Learning Conference (CSCL'2003)*, Bergen, Norway, pp. 343-352, June 2003.
- [6] A. Jaillet, "Peut-on repérer les effets de l'apprentissage collaboratif à distance", *Distances et savoirs*, vol. 3, no. 1, pp. 49-66; 2005.
- [7] O.C. Santos, A. Rodríguez, E. Gaudio, J-G. Boticario, "Helping the tutor to manage a collaborative task in a web-based learning environment," *Communication in the Workshop Towards Intelligent Learning Management Systems*, Sydney, Australia, p.72-81, 2003.
- [8] A. Martínez, Y. Dimitriadis, E. Gómez, B. Rubia, P. de la Fuente, "Combining qualitative and social network analysis for the study of classroom social interactions," *Computers and Education*, vol. 41, no. 4, pp.353-368, 2003.
- [9] P.A. Tedesco, "MARCo: Building an Artificial Conflict mediator to Support Group Planning Interactions," *International Journal of Artificial Intelligence in Education*, vol 13, pp.117-155, 2003.
- [10] M. May, S. George, P. Prévot, "Tracking, Analyzing, and Visualizing Learners' Activities on Discussion Forums," *Proc. International Conference on Web-based Education (WBE 2007)*, Chamonix, , pp. 649-656, 2007.
- [11] A. Lavallard, *Exploration interactive d'archives de forums : Le cas des jeux de rôle en ligne*, Doctorate thesis, Université de Caen, Juillet 2008.
- [12] F. Diagne, "Em-AGIR : un Environnement Multi-AGent ouvert pour la supervision à partir d'Indicateurs Réutilisés," *Proc. 2ième rencontre des jeunes chercheurs RJC-EIAH08*, Lille, France, pp. 65-70, 2008.
- [13] D. Pham Thi Ngoc, "Réingénierie des EIAH : automatiser et réutiliser le savoir-faire en analyse d'usage," *Proc. 2ième rencontre des jeunes chercheurs RJC-EIAH08*, Lille, France, pp. 99-104, 2008.
- [14] P.R. Jermann, *Computer Support for Interaction Regulation in Collaborative Problem-Solving*. Doctorate thesis, Genève, 2004.
- [15] N. Avouris, M. Margaritis, V. Komis, R. Melendez, A. Saez, "ModellingSpace: Interaction Design and Architecture of a collaborative modelling environment," *Proc. the 6th conference of Computer Based Learning in Science (CBLIS)*, Nicosia, Cyprus, pp. 993-1004, 2003.
- [16] P. Laforcade, T. Nodenot, C. Choquet, P.A. Caron, "Model-Driven Engineering (MDE) and Model-Driven Architecture (MDA) applied to the Modeling and Deployment of Technology Enhanced Learning (TEL) Systems: promises, challenges and issues," In: *Architecture Solutions for E-Learning Systems*, pp. 116-136, 2007.
- [17] J. Laflaquière, L.S. Settouti, Y. Prié, A. Mille, "A trace-based System Framework for Experience Management and Engineering," *Workshop on Experience Management and Engineering (EME 2006) in conjunction with KES2006*, Bournemouth UK. 2006.
- [18] L.S. Settouti, Y. Prié, J-C. Marty, A. Mille, "A Trace-Based System for Technology-Enhanced Learning Systems Personalisation," *Proc. The 9th IEEE International Conference on Advanced Learning Technologies*, Riga, Latvia. 2009.
- [19] E. Seidwitz, "What models Mean," *IEEE Software*, vol. 20, no. 5, pp.26-32.

- [20] J. Bézivin, O. Gerbé, "Towards a precise Definition of the OMG/MDA framework," Proc. the 16th Conference on Automated Software Engineering IEEE (ASE'2001), San Diego (USA), pp 273-280, 2001.
- [21] OMG. Meta-Object Facility (MOF) Specification Version 1.4. <http://www.omg.org/technology/documents/formal/mof.htm>
- [22] Moodle. Modular Object-Oriented Dynamic Learning Environment. <http://docs.moodle.org/fr/Accueil>
- [23] T. Djouad, "Analyser l'activité d'apprentissage collaboratif : Une approche par transformations spécialisées de traces d'interactions", Proc. 2ième rencontre des jeunes chercheurs RJC-EIAH08, Lille, France, pp. 93-98,2008.
- [24] Jena. A Semantic Web Framework for Java, <http://jena.sourceforge.net/>
- [25] A. Mille, B. Fuchs, B. Chiron, "Raisonnement fondé sur l'expérience : un nouveau paradigme en supervision industrielle". Revue d'intelligence artificielle, vol. 13, pp.97-128, 1999