HAL
open science

Supporting activity modelling from activity traces

Olivier L. Georgeon, Alain Mille, Thierry Bellet, Benoît Mathern, Frank E. Ritter

To cite this version:

Olivier L. Georgeon, Alain Mille, Thierry Bellet, Benoît Mathern, Frank E. Ritter. Supporting activity modelling from activity traces. Expert Systems, 2012, 29 (3), pp.261-275. 10.1111/j.1468-0394.2011.00584.x . hal-00980679

HAL Id: hal-00980679

https://hal.science/hal-00980679v1

Submitted on 8 Nov 2016

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Supporting Activity Modeling from Activity Traces

Olivier L. Georgeon [2], Alain Mille [2], Thierry Bellet [1],

Benoit Mathern [2], Frank E. Ritter [3]

[1] *Institut Français des Sciences et Technologies des Transports, de l'Aménagement et des Réseaux*

25, Avenue François Mitterrand, 69500 Bron, France

[2] *Université de Lyon*

86 Rue Pasteur 69007 Lyon, France

[3] *The Pennsylvania State University*

University Park. PA 16802, United States of America

Corresponding author: olivier.georgeon@liris.cnrs.fr

Keywords: Sequence-mining, Timeline-analysis, Activity-trace, Knowledge-Based-System, Activity-modeling.

ABSTRACT

We present a new method and tool for activity modeling through qualitative sequential data analysis. In particular, we address the question of constructing a symbolic abstract representation of an activity from an activity trace. We use knowledge engineering techniques to help the analyst build an ontology of the activity, i.e., a set of symbols and hierarchical semantics that supports the construction of activity models. The ontology construction is pragmatic, evolutionist and driven by the analyst in accordance with their modeling goals and their research questions. Our tool helps the analyst define transformation rules to process the raw trace into abstract traces based on the ontology. The analyst visualizes the abstract traces and iteratively tests the ontology, the transformation rules, and the visualization format to confirm the models of activity. With this tool and this method, we found innovative ways to represent a car driving activity at different levels of abstraction from activity traces collected from an instrumented vehicle. As examples, we report two new strategies of lane changing on motorways that we have found and modeled with this approach.

1. INTRODUCTION

We introduce here new principles based on knowledge engineering techniques for designing systems to help *analysts* create models of activity from *activity traces*. We illustrate these principles with a software tool that we have implemented, and with an example modeling analysis that we have performed using this tool.

By *activity trace* we mean a set of multiple streams of quantitative or symbolic data that record (at least partially) an activity performed by a *subject*. The *analysts* may be psychologists seeking to build theories of the subject's cognition, ergonomists seeking to design better user interfaces, analysts seeking to predict the subject's behavior in specific conditions, trainers seeking to improve training techniques, or even the subjects themselves seeking to improve their understanding of their actions. In each case, the created models of activity constitute micro-theories proposed by the analysts to describe, explain, and try to predict how the subject performs the activity.

The principles and the tool that we introduce here address three needs for helping analysts construct models of activity from activity traces. The first need is for helping the analyst learn previously unknown aspects or details of the subject's activity from the activity traces. The second need is for helping the analyst construct meaningful symbolic representations of interesting aspects of the activity. These representations, associated with the explanations proposed by the analyst, constitute the models of activity. The third

need is for helping the analyst test and support the created models of activity with regard to the activity trace.

Activity traces have also been called *protocols* (Ericsson and Simon, 1993) or simply *sequential data* (Sanderson and Fisher, 1994). We prefer the term *activity trace* because this term conveys the idea that the trace is intended to be interpreted by somebody (designated here as the analyst). We think of a trace as a footprint that helps who sees it understand what happened. Our activity traces yet differ from mere footprints in that they are not accidentally produced or unprocessed but they rather result from the analyst's choices and setup.

Many software tools have been implemented for activity trace analysis. A recent review (Hilbert and Redmiles, 2000) notes forty of them. These tools cannot autonomously generate a comprehensive explanation of human behavior but they interactively support analysis. This analysis consists of identifying, categorising, labelling, and transforming pieces of data and information in the activity trace. We summarise this process by the notion of *abstraction*. The analysts use their expertise and knowledge to formulate a whole set of tiny hypotheses and choices concerning how to collect the data, how to filter it, how to cluster and label it, and how to display and to report it so that it responds to the analysis purpose.

Although most of the existing tools acknowledge the central role of the analysts and the importance of their knowledge and expertise in the analysing process, these tools still

lack knowledge representation mechanisms to support the management of the analysts' knowledge. For instance, MacShapa (Sanderson et al., 1994), does help analysts label and cluster the behavioural data. It also acknowledges the usage of these labels as symbols to describe the activity. It, however, does not help the analyst formulate and manage the symbolic inferences she can make from these symbols.

We made the hypothesis that aspects of knowledge engineering can help design software systems that address the three needs identified above. We use ontology management facilities and rule engines to capture the hypotheses and choices made by the analyst. When interactively used by the analyst or a group of analysts, these facilities help formalize the way analysts find interesting symbolic patterns and infer models from them. Once this knowledge is formalized, the system uses it to automatically compute new representations of the activity from the activity traces. The system also helps analysts organize and store the different concepts and rules that summarize different studies and help capitalize on these studies.

To explain our principles and demonstrate the system and its design, we have organized this paper as follows: section 2 presents the principles of activity trace modelling, based on a pragmatic and evolutionist approach. Section 3 presents the prototype software tool that we have implemented from these principles, its technical features, its architecture, and its user interface. Section 4 presents an example study in which we have used this tool to create models of lane change on motorways from activity

traces generated with an instrumented car. The method is then summarized in the conclusion.

2. MODELING ACTIVITY TRACES

The notion of activity traces is widely used in the human behavior literature, and we cannot attribute its origin to a specific author. Only more specific related notions can be identified, such as pattern languages, as reviewed by Dearden and Finlay (2006), or grammar representations (Olson et al., 1994). Despite the wide usage of the term activity trace, we could not find a definition of it, which led us to propose the following definition:

An activity trace is a meaningful inscription, from the viewpoint of an analyst, of the flow of what has happened, from the viewpoint of a subject.

With this definition, we want to highlight that an activity trace always implies two viewpoints, situated in two different moments. It implies the subject's viewpoint, when he or she was performing the activity, and the analyst's viewpoint, when he or she is analyzing the activity trace. Indeed, an activity trace cannot be an inscription of all that happened (if that had any sense), because an activity only concerns what relates to the subject's perspective, goals, and intentions. Thus, inevitably, the analyst has to make assumptions about what is meaningful to the subject when the analyst sets up the tracing mechanism. In addition, the activity trace depends on what activity aspects interest the

analyst and what makes sense to her according to her previous knowledge and to her analysis goals.

Because an activity trace depends on the analyst's knowledge and assumptions, it can only be modeled in an iterative way, each iteration producing new knowledge leading to new hypotheses for the next iteration. Ericsson and Simon (1993) described this iterative nature of analyzing human behavior: "In designing our data-gathering schemes, we make minimal essential theoretical commitments, then try to use the data to test stronger theories" (p. 274). Moreover, none of the iterations can produce knowledge that could be proven to be true in an absolute sense, but only knowledge that is more efficient and useful with regard to the analyst's goals, and which is more convincing to the analyst's community than the knowledge from the previous iteration. More broadly, this conception of knowledge relates to a pragmatic epistemology (James, 1907) and an evolutionist epistemology (Popper, 1972).

These pragmatic and evolutionist aspects are crucial when defining a methodology and a tool for activity trace modeling. By fully acknowledging these aspects, we have designed a tool that facilitates and accelerates the evolutionist modeling process. The tool helps formulating a series of micro-hypotheses of possibly-useful symbols, possibly-useful transformation rules to transform the low-level data into higher-level data, and possibly-useful representations of the activity trace based on the micro-hypotheses. If the obtained representation does not help the analyst understand the activity better, then she rejects these micro-hypotheses; if it helps, then she keeps them.

The tool is designed to shorten this formulation / usage / validation-or-rejection loop. This process leads to the construction of a set of micro-hypotheses that are validated by the analyst. This set constitutes a formalization of the analyst's knowledge about how to understand the activity. The tool stores this knowledge, helps the analyst keep track of it, and helps the analysts' community discuss and question it.

The next section details how the tool reaches this goal by helping the analyst define sequences at the right level of abstraction, and simultaneously identify interesting subsequences, define them precisely, and query the whole trace in search for their occurrences.

2.1. Collecting a symbolic trace

A raw activity trace can be made of any kind of data describing a subject's activity flow and intended for an analyst's usage. In a broad sense, it can range from video or audio recording to computer logs. The only common point is that the data are temporally organized, meaning that each data piece is associated with a timestamp referring to a common time base. The first abstraction step consists of converting these raw traces into sequences of symbols. We refer to this step as the *discretization* of the raw trace into a symbolic trace. The symbols in the symbolic trace have to be meaningful to the analyst, and they are chosen on a pragmatic and evolutionist basis, in compliance with a pragmatic and evolutionist epistemology introduced above (introduction of section 2). The discretization process can be manual, semi-automatic, or automatic. The definition of

the symbols may evolve in parallel with the implementation of the discretization process. This is because the later interpretation of the symbols may differ from the meaning initially intended by the analyst when she specifies the discretization algorithm.

For example, in a study of car driving, we have used classical mathematical curve analysis method to generate symbols of interest from numerical values of the vehicle speed, the steering wheel angle, and the pedal positions. In this case, the symbols correspond to threshold crossing, local extremum, and inflexion points. Figure 1 illustrates this discretization process.

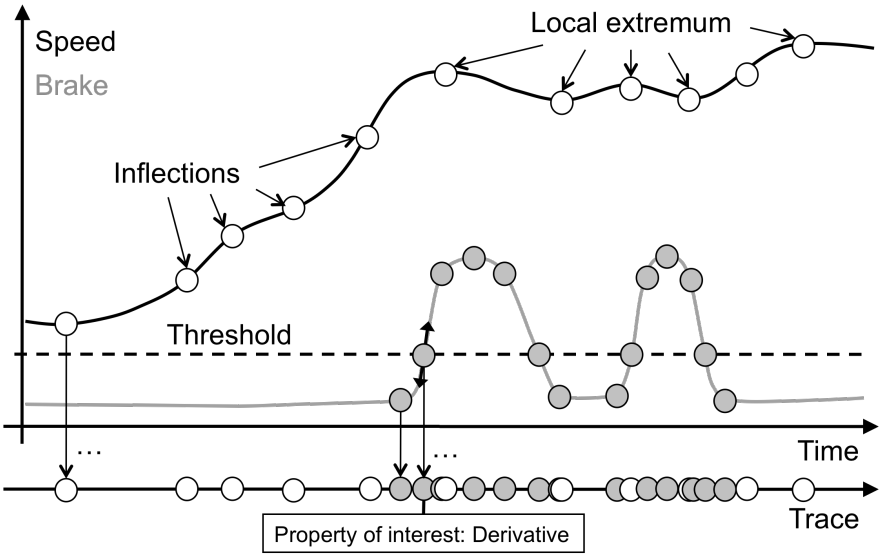

Figure 1. Discretization of analogical traces.

In Figure 1, the curves represent the vehicle speed in km/h, and the brake pedal position in percentage of range. Symbols of interest are shown on these curves as circles (threshold crossings, inflexion points, local extremums). The symbols are merged into the

symbolic trace that is represented at the bottom of Figure 1. The figure also shows a derivative value as an example property of interest associated with a symbol generated by a brake pedal threshold crossing. The analyst specifies the way to generate these symbols so that they correspond to meaningful events that describe the activity. In this example, the threshold crossing indicates the extent of the braking action and the derivative value indicates the abruptness of this action. Notably, while creating these symbols, the analyst claims the *existence* of the events that these symbols represent. So doing, the analyst defines an ontology of the activity.

Our experience has taught us that the system must keep a connection between the raw trace and the symbolic trace, and provide parallel displays of both of them. The analyst needs to tune many parameters of the discretization algorithms, like the threshold values or noise filters. The analyst validates the chosen symbols and algorithms by comparing the symbolic trace to the raw trace and ensuring that the symbolic trace represents what is happening. While she defines and validates these symbols, the analyst also supports her claim that the events represented by these symbols "exist". This support arises because the method to generate these symbols from the recorded data is formally specified and explained by the analyst. In this example, after we fully specified the discretization algorithm in accordance to our specific modeling goals, the discretization algorithm could then compute the symbols fully automatically.

2.2. Modeling the symbolic trace

At the symbolic level, analysts most often want to focus on relations between events. Indeed, events are not meaningful by themselves, but they become meaningful in the context where they relate to each other (Sanderson and Fisher, 1994). Examples of such relations include a "sequence following within a certain period of time", "co-occurrence within a certain period of time", and "causality with regard to a certain explanative theory". Building and understanding these relations between events is a part of the analyzing process. By definition, a set of elements connected through relations is a graph. Therefore, we model the symbolic traces with a graph structure.

More precisely, our trace graph structure has two parts: a sequence and an ontology. The sequence is a part of the graph that is made of event instances and of relation instances between event instances. The ontology is a part of the graph that is made of event classes and of relations between event classes. Figure 2 illustrates this graph structure with a simplified example taken from the car driving study.

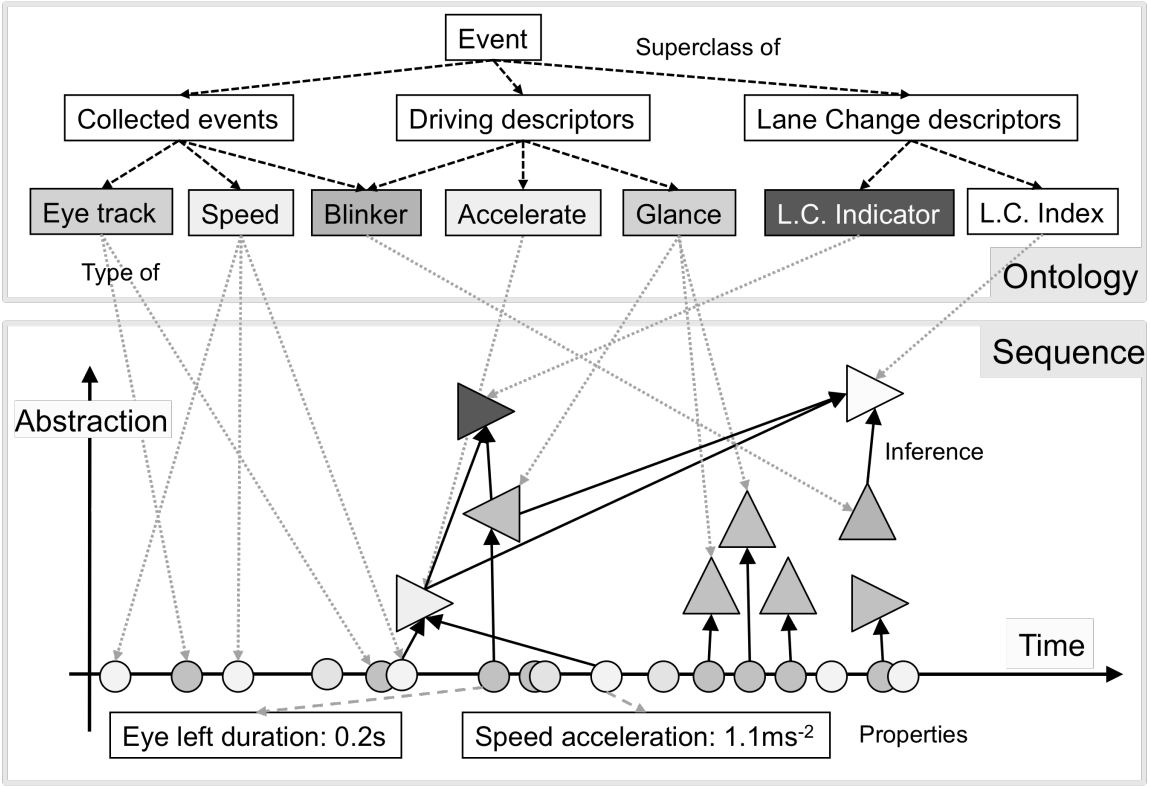

Figure 2: Activity trace modeled in a knowledge engineering system.

The sequence is represented in the bottom part of Figure 2 and the ontology in the top part. In the sequence graph, event instances are represented as circles and triangles. Relation instances between event instances are represented as solid arrows between these circles and triangles. Example properties of event instances are represented with gray dashed arrows pointing to their value at the bottom of the figure (e.g., the duration of an eye movement and acceleration value associated with an inflexion point of the speed curve).

Our tool displays the sequence in a similar form as shown in Figure 2; the exact form is shown in Figure 4. This display uses two axes: the time axis and the “abstraction” axis.

That is, the events' time-code attributes determine their "x" coordinates, and the analyst specifies their "y" coordinates when she configures these event's class in the ontology. The analyst can use the "y" coordinate to express different meanings; our recommendation is to use it to express an idea of abstraction level related to a specific analysis. In this example, the lowest level (circles) represents the events obtained from the discretization process; the intermediary level represents events that describe the activity in usual driving terms: accelerate, glance, turn signal on/off (blinker); and the higher level represents events describing lane change behavior: indicator of intention to change lane, and index of lane change. This example expresses the analyst's assumption that the conjunction of an acceleration and a glance to the left rear mirror can generate an indicator of the driver's intention to change lane (L.C. Indicator).

In the ontology graph, the nodes represent event classes, and the edges represent the relation "subclass of" (dashed black arrows). The analyst defines the ontology during the modeling process. For example, the "Collected events" class includes all the event classes that come from the discretization process. The "Driving descriptors" class gathers the intermediary event classes that describe the activity in usual driving terms. The "Lane Change descriptors" class gathers the most abstract event classes describing lane changes. The dotted gray arrows in the figure represent the relation "type of" going from event classes defined in the ontology to event instances in the sequence.

Again, like most ontologies, this ontology is made by the analyst on a pragmatic basis. It is likely that two analysts will create two different ontologies. While the software

cannot demonstrate that one is better than the other, it does help the analysts formalize and discuss them. This discussion leads to the construction of a language for describing the activity that represents an agreement about the terms that can be used to describe the activity. In addition, the ontology also supports the analysts' agreement about how the trace should be visualized, because the visualization properties of the symbols are stored in the ontology.

This trace formalism enables the analyst to conduct a hierarchical analysis of the activity. Event instances form a hierarchy where higher-level events are temporal patterns of lower-level events. The ontology also defines a hierarchy because some event classes are sub-classes of others. Notably, these two hierarchies are different because lower-level event instances do not necessarily belong to a sub-class of the class of higher-level event instances.

3. SYSTEM IMPLEMENTATION

We have implemented a prototype system based on an assemblage of open source knowledge-engineering tools: an ontology editor, an inference engine, visualization facilities, and documentation facilities. This system is named ABSTRACT (Analysis of Behavior and Situation for menTal Representation Assessment and Cognitive acTivity modeling). Figure 3 illustrates this assemblage.

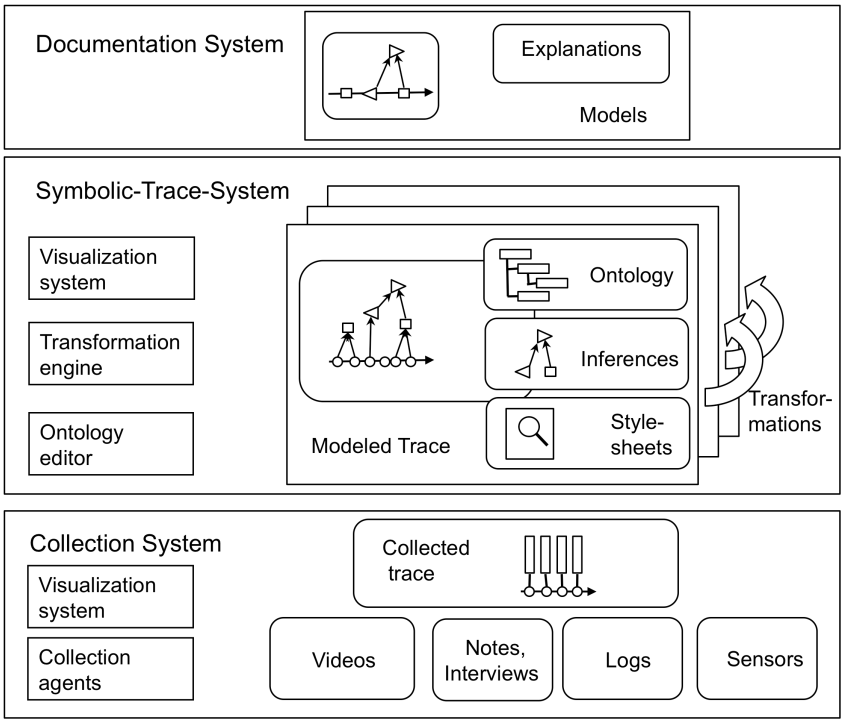

Figure 3: The architecture of the ABSTRACT activity analysis system.

The system can be split into three levels: a lower level, at the bottom of the figure, which is the *Collection System*; a core level, in the center of the figure, which is the *Symbolic Trace System* itself; and a higher level, on the top of the figure, which is a *Documentation* level.

3.1. The collection system

The collection system integrates tools to help the analyst prepare the symbolic trace. We call these tools collection agents. Collection agents may be automatic when specified once by the analyst, or may require the analyst's intervention. Automatic collection agents can be tools for preprocessing sensor data or computer logs, as in the example of

section 2.1. Semi-automatic collection agents can be tools for helping the analyst take notes, record interviews, or transcript video data. As noted, this discretization cannot be done blindly, but must be driven by the analyst. Hence, this level requires visualization facilities. We use Microsoft Excel with specific Visual Basic Application (VBA) macros as a visualization tool for the collection system. In this visualization, each event of the symbolic trace is displayed as a line in the spreadsheet. The lines are colored according to the event's type, and the event's properties are organized in different columns. We have implemented a specific video player and analogical data player that triggers VBA macros that automatically scrolls down the spreadsheet in synchronization. Some of these facilities are also available in commercial quantitative data analysis tools such as MacShapa (Sanderson et al., 1994) and NVivo. These facilities allow the analyst to check and validate or reject the symbolic trace, that is, refine the discretization algorithm and its diverse parameters until she gets a satisfying and meaningful symbolic trace including appropriate properties of interest.

3.2. The symbolic trace system

The symbolic trace system is the knowledge-engineering system itself. At this level, the traces are modeled as described in section 2.2. In addition, they are associated with a set of inference rules and a set of *style-sheets*. A style-sheet is a specification for displaying the trace on the screen. It specifies how semantic properties of the events that are defined in the ontology should be converted into visualization properties, such as shape and position. Style-sheets also implement particular time scales, and particular

filters to display only the interesting aspect of the trace for a particular analysis. They correspond to different ways of looking at the trace according to different modeling goals.

The inference rules produce inferred symbols from patterns of previously existing symbols. The principle is to query the graph in the search for subgraphs that match certain patterns, and to attach new nodes and arcs to the matching subgraphs. These new nodes represent the inferred symbols and the new arcs represent the *inference* relations. The usage of this inference mechanism is further described in Section 3.5.

Technically, the *sequence* part of the activity traces is encoded as Resource Description Framework (RDF) graphs. We choose RDF because it is the most widely used specification for graph encoding. We use XML as a serialization of RDF to store sequences, because XML makes RDF graphs easy to share with other applications. The ontology is encoded as RDF-Schema (RDFS), because RDFS is the simplest ontology language based on RDF. We use Protégé as a graphical ontology editor. That is, an installation of Protégé is embedded in our tool, and the analyst uses it to define the ontology of his traces. The graphical displays of our traces are encoded under the Scalable Vector Graphic (SVG) specification. Because Firefox natively supports SVG, we use it as a visualization tool, and we have implemented most of the tool as a web application in PHP. We use Extensible Stylesheet Language (XSL) as a transformation language for transforming RDF traces into their SVG graphical representation. We use SPARQL as a query language for graphs, as we will explain in section 3.5.

3.3. The documentation system

Analysts using our symbolic trace system expressed the need for a higher system layer providing a way to both index and attach documentation to episodes of interest. We implemented this by associating the symbolic trace system with a wiki. As we have made the choice of implementing ABSTRACT as a web application, analysts can reference each episode of interest by their URL, and easily paste this URL into a wiki page. Moreover, some new wiki implementations, like Semantic Mediawiki[1], include semantic facilities. We are still investigating how these semantic facilities can be used to merge the ontology editor with the documentation system into a single semantic documentation system.

3.4. System usage

The user interface is accessible as a web-page in any browser that supports SVG, such as Firefox. This interface is illustrated in Figure 4. It has four tabs: the *Open* tab that allows the analyst to select a trace in a list; the *Info* tab that displays general information about the selected trace, such as its creation date and its version, and the management of stylesheets; the *View* tab that displays graphical visualizations of the trace; and the *Edit*

---

[1] Semantic Mediawiki (http://semantic-mediawiki.org)

tab that allows the analyst to write queries to transform the trace. The interface also provides a link to the ontology editor, Protégé.

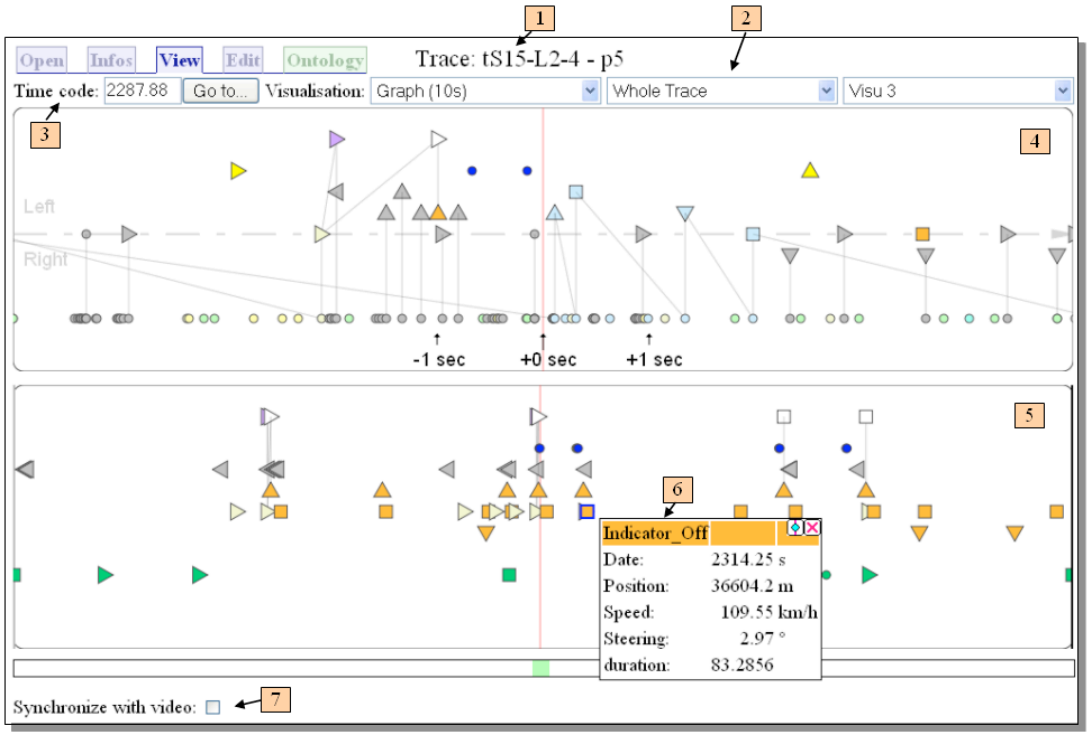

Figure 4: ABSTRACT User interface.

The *View* tab shown in Figure 4 provides the following functionalities (noted with numbered boxes):

1. Unique ID of the analysis, including aspects of the trace file and analyses done.
2. Selection of different visualization style sheets in drop-down lists. Different visualizations can be displayed simultaneously on the screen, and their time-code is synchronized.

3. Time code: this value corresponds to the cursor position in the visualization modules (vertical red line). The analyst can enter a time code and click *Go to* to focus on it.
4. Visualization sample with a time span of 10 seconds. The analyst can scroll the trace left and right with the mouse. This visualization example corresponds to the simplified description given in Figure 2.
5. Visualization sample of an entire trace (20 minutes), with only the high-level symbols displayed. These trace examples come from our car driving study and are further explained in section 4.
6. The analyst can show the symbols' properties by clicking on the symbols.
7. The system can synchronize with a video player. When this box is checked, the system gives the timecode control to the video player and automatically follows it.

3.5. The transformation mechanism

The analyst uses the *Edit* tab to write queries that infer higher-level symbols from patterns of lower-level symbols. For instance, Table 1 illustrates a query to infer the *Lane Change indicator symbol* shown in Figure 2, which indicates that a lane change is about to happen. In this example, the analyst wants to test the hypothesis that this indicator can be inferred from a conjunction of an *accelerate* event with an acceleration value greater or equal than 1 m/s^2, followed by a *glance* event pointing to the left mirror (generated by an eye-tracker), both within 1 second of each other.

The graph elements, either from the sequence or the ontology, are handled as triples [node, edge, node]. A query consists of a selection clause (WHERE) and a CONSTRUCT clause. The selection clause specifies a pattern of triples that should match the graph, and the CONSTRUCT clause specifies a pattern of triples that should be added to the graph wherever a pattern matches the selection clause. In addition, matching patterns can be restricted by a FILTER clause. The syntax shown in Table 1 has been simplified for clarity[2].

Table 1: Simplified inference query to infer a lane change.

```
CONSTRUCT
(?r1, infer , Indicator_Symbol)
(?r2, infer , Indicator_Symbol)
(Indicator_Symbol, type , Lane_Change_Indicator)
WHERE
(?r1, type , Accelerate)
(?r2, type , Left_Mirror_Glance)
(?r1, time-code , ?d1)
(?r2, tine-code , ?d2)
(?r2, Acceleration_Value , ?v1)
FILTER
( ?v1 > 1)
(sequence(?d1,?d2,1)) # in order, within 1 second
```

In this query, ?r1, ?r2, ?d1, ?d2, and ?v1 represent variables. Each of them must match the same graph element each time they appear in the query. The *sequence* function tests that the time codes ?d1 and ?d2 occur in order and within the parameter of 1 second of each other.

---

[2] The complete SPARQL syntax can be found in the SPARQL documentation, http://www.w3.org/TR/rdf-sparql-query/

In our implementation, the analyst has to know SPARQL to specify queries on the trace. To make it simpler, however, we have implemented a template mechanism that prepares skeletons of queries. We have also added some customized functions in our implementation of SPARQL, such as the *sequence* function presented above. These functions facilitate the specification of queries that compare time codes of events, and make it easier to specify temporal constraints. In so doing, we are implementing a semantics of time, for example, the semantics of the relation of co-occurrence and of sequential ordering. In the future, we plan to add options to let the analyst specify these queries from a graphical interface based on the visualization of the trace.

The *Edit* tab allows the analyst to visualize the resulting trace in a similar way as the *View* tab, but it also allows her to reject the trace if she is not satisfied by the result. This feature helps the analyst refine the query in search for the best symbols and inferences rules she can get to describe the activity from the actual data. Our system returns the number of times the pattern has matched in the trace. This number indicates the number of new symbols added. The system also provides an export function to a text file that can be imported into other tools like Microsoft Excel for further statistical computations. Queries are saved as independent files and the tool helps the user reference them. The database of queries associated to the ontology constitutes a representation of the analyst's understanding about how to make sense of the activity trace.

4. EXAMPLE ACTIVITY MODEL

We report here an example activity modeling analysis taken from a car driving study (Henning et al., 2007). Another example analysis—in a study of non-state political violence—is reported by Georgeon, Morgan, Horgan, and Braddock (2010). Figure 5 shows a 10 second section of a car-driving activity trace focusing on a lane change on a motorway. The legend is presented in Figure 6.

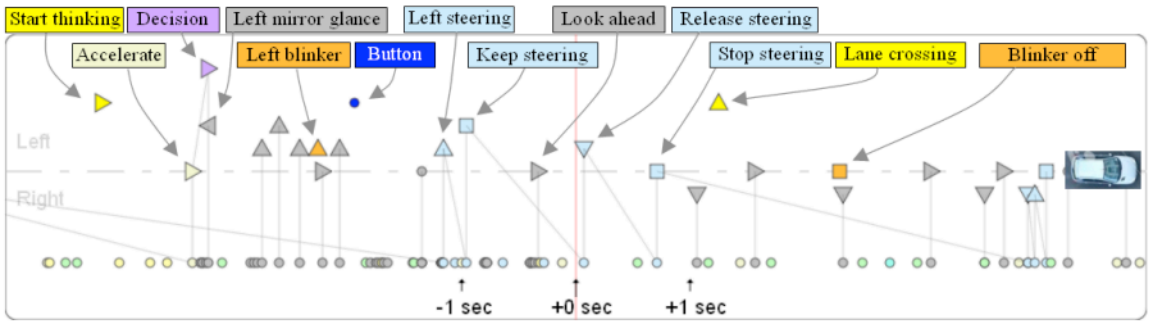

Figure 5: Example of lane change on a motorway (screenshot with text labels added at the top).

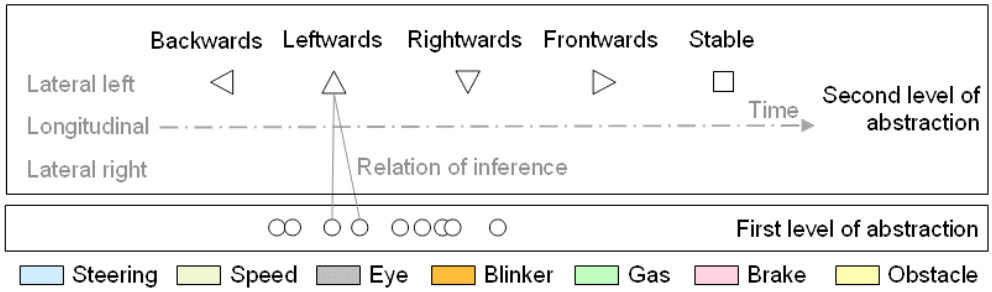

Figure 6: Legend of car driving symbolic trace.

In Figure 5, the "Button" is an index signal from the experimenter recorded during the experiment, the *start thinking* event comes from a verbal signal given by the driver in

a video-based post-experiment interview, and the *lane crossing* event is the moment when the left front wheel crosses the lane, manually encoded from the video. The representation of the driving episode given in Figure 5 is then automatically generated with the inference rules defined by the analyst.

Using the trace querying facilities of ABSTRACT on this driving data set, we could identify two categories of lane changes that we explain by the performance of two strategies (Figure 7 and Figure 8). In these descriptions, the lower part is a representative trace episode from our database while the upper part is drawn by hand as an abstract description of the strategies. This upper part also shows the car trajectory in the lanes, respecting the scale ratio length/width: about 300 m of 4 m wide lanes.

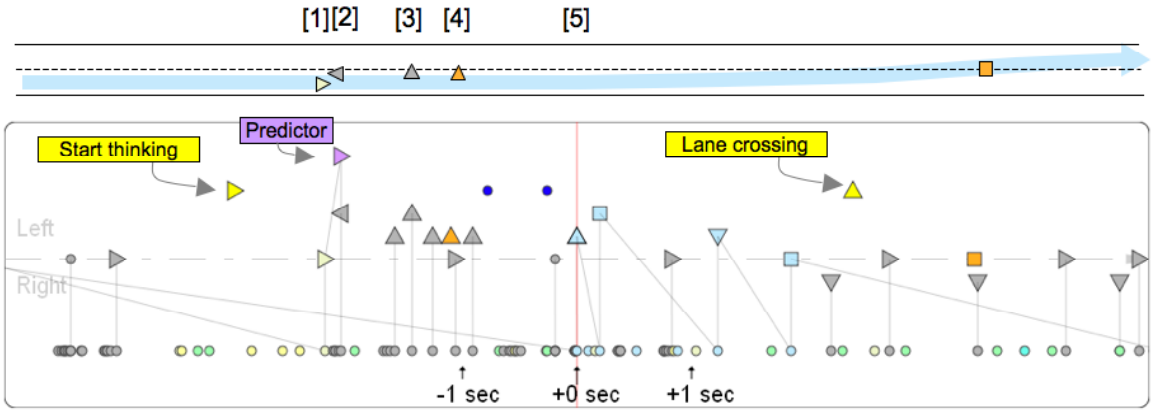

Figure 7: Lane change with acceleration (Lane_Change_Delayed).

The strategy displayed in Figure 7 is characterized by beginning in a situation where the subject is impeded by a slow vehicle. In this case, the subject starts accelerating [1] almost at the same time as he looks at his left mirror [2]. Then, if there is no vehicle

coming from behind, he starts looking at the left lane [3], he switches his blinker on [4], and he performs the lane change [5]. In this situation, the acceleration associated with a glance to the left mirror appears as a good predictor of the lane change. It occurs more than one second before the subject switches the blinker on.

In the situation of Figure 8, no slow vehicle impedes the subject, and he performs the lane change "on the fly". In this case, we can find no behavioral sign of his intention to change lane before the blinker is switched on [1]. Nevertheless, the blinker appears to be a sufficient predictor in that case, because it is switched in anticipation of the lane change, several seconds before the maneuver: looking to the left lane [2], looking to the left mirror [3], and starting steering [4].

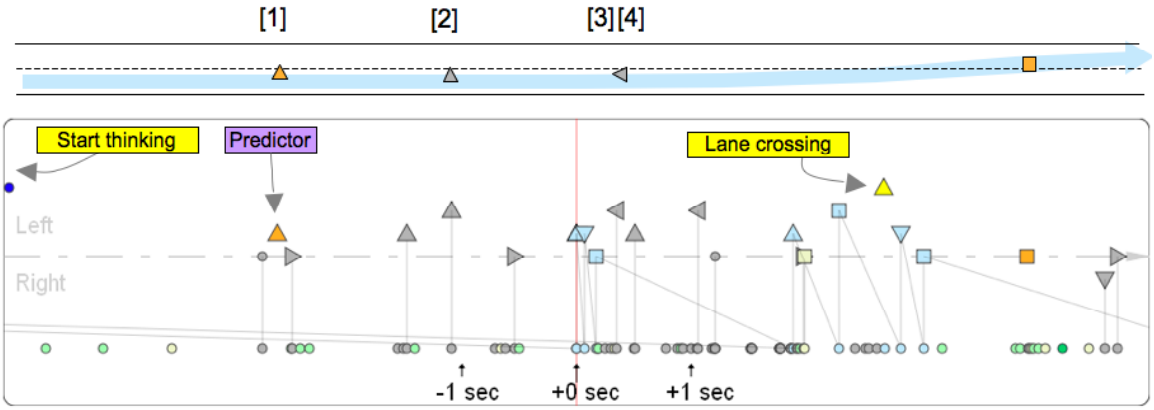

Figure 8: Lane change anticipated without acceleration (Lane_change_anticipated).

In parallel of searching and identifying these categories of situations and strategies, we define symbols to represent them and inference rules to generate these symbols. Finally, we have named the first strategy *Lane_change_delayed* and represented it with

white triangles, and the second strategy *Lane_change_anticipated* represented with white squares. Figure 9 shows that these two types of event occur four times in a representative 20-minute motorway ride of a subject.

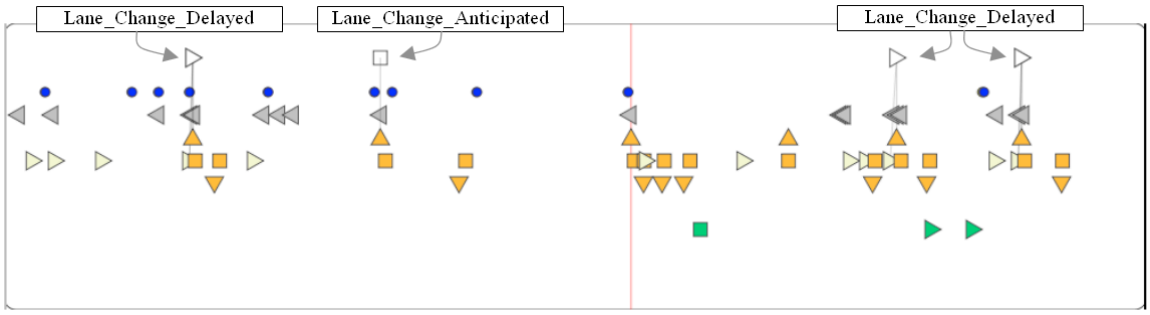

Figure 9: Categorization of lane changes between Lyon and airport (20 minutes long).

Figure 9 displays only the symbols that are useful to see the lane changes: the index button pressed by the experimenter (blue circles), the blinker (orange triangles up and down), the accelerations (ochre triangles to the right), the left mirror glances (gray triangles to the left), main junctions on motorways given by the GPS position (green square and triangles on the bottom of the display), and the lane change category symbols (white triangles and squares). In this example, one lane change (marked by the vertical cursor line in the figure) was not categorized. The uncategorized cases would require further study to understand their specificity. Once the analyst has finished her analysis, she can export the abstract traces into a spreadsheet to compute and report the statistics of the occurrences of events of interest.

Despite the extensive existing studies of car driving (e.g., Groeger, 2000) and on lane change maneuvers (e.g., Salvucci and Liu, 2002), we could find no representation of the

driving activity that could compare to ours in terms of comprehensiveness of the data and capacity to support higher-level understanding. In the case of lane changes, this innovative description helped us discover different strategies that have not been reported in the literature before. So doing, this study shows how our principles and tool have addressed the needs enounced in the introduction: our prototype tool helped us generate comprehensive symbolic representations of the activity at an appropriate abstraction level to discover previously unknown knowledge about the activity. It also helped us explain, report, and back up our models with the collected traces.

5. RELATED WORK

In this section, we situate our work in relation to two research areas, the area of qualitative data analysis and the area of trace-based reasoning.

In the area of qualitative data analysis, many of software tools address the need for supporting the transcription of raw data into sequences of encoded events, for example, Dismal (Ritter and Larkin, 1994, Ritter and Wood, 2005), NVivo, INTERACT, InfoScope, MORAE, MarShapa (Sanderson et al., 1994), and MacVisSTA. As such, these tools relate to our collection system as described in Section 3.1. Among these tools, MacVisSTA (Rose et al., 2004) particularly relates to this aspect of our work in that it supports merging multimodal data into a common timeline.

At the symbolic level—also called transcript level in some studies—tools like Theme (Magnusson, 2000) support the automatic discovery of temporal patterns based on

statistical properties. We consider our tool complementary to these tools because our tool helps find symbolic pattern based on the meaning they have to the analyst rather than on their statistical properties. In the car-driving example, our symbols of interest are not particularly frequent or infrequent nor do they obey pre-assumed statistical laws.

HyperRESEARCH appears to be the only qualitative data analysis tool that supports the validation of hypothetic theories through rule-based *expert system* techniques (Hesse-Biber et al., 2001). We find HyperRESEARCH's underlying principles for theory building very similar to ours. It, however, does not focus on temporal semantics and does not offer symbolic timeline visualization facilities to support activity modeling. Its rule engine also does not exploit elaborated hierarchical semantics defined in an ontology, as apposed to our solution based on SPARQL and RDF Schemas.

The other related research area, Trace-Based Reasoning (TBR) (Cordier et al., 2009) comes from the domain of knowledge representation, and more precisely from Case-Based Reasoning (CBR) (Aamodt and Plaza, 1994). CBR consists of helping users solve new problems by adapting solutions that have helped them solve previous problems. TBR extends CBR to retrieve useful cases as episodes from the stream of an activity trace (Mille, 2006). Our work relates to TBR because both address the question of modeling and representing a stream of activity for future usage. In particular, we pull lessons from the work of Settouti, Prié, Marty, and Mille (2009) that implemented a trace-based system to support the management and the transformation of traces. TBR has been used to implement companions that provide assistance to the user based upon previous usage

(e.g., Cram et al., 2008), or to support reflexive learning by providing the users with a dynamic display of his passed activity (Ollagnier-Beldame, 2006). As opposed to these works, our work helps a user (the analyst) understand the activity of another user (the subject). So doing, our work brings principles of qualitative data analysis to TBR and brings TBR techniques to address problems of quantitative data analysis.

6. CONCLUSION

We have defined the principles of a methodology and a software tool to help an analyst create models of activity from activity traces in an iterative and interactive fashion. These principles relate to the notion of *abductive reasoning* in that they consist of helping analysts form, organize, and test micro-hypotheses to explain and represent the activity. Specifically, they help investigate what concepts and semantics describe the activity the best and represent these concepts in an ontology and apply this semantics through a rule engine. These principles are summarized in Figure 10.

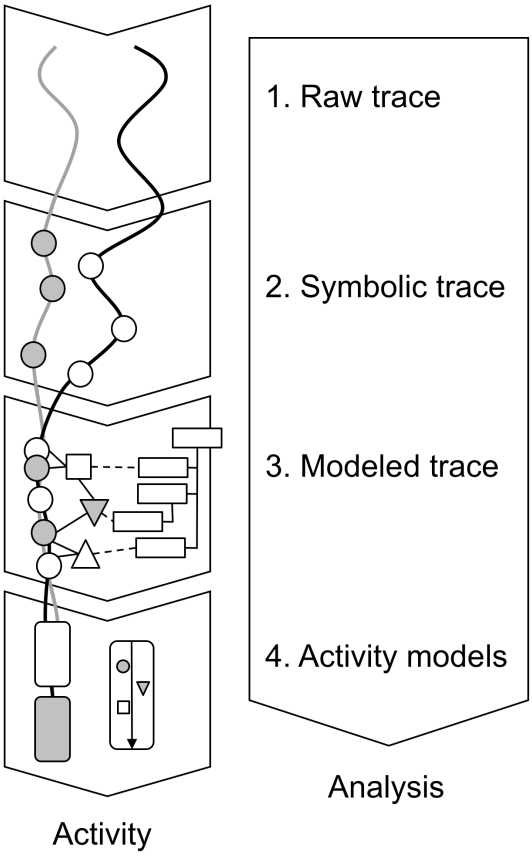

Figure 10: Activity modeling from activity traces

In Figure 10, the collected raw data is represented as curves along the activity axis. The first analysis step consists of collecting this raw trace. The second step consists of producing a symbolic trace (symbols represented by circles) through the discretization of the raw trace. The third step consists of modeling the symbolic trace by inferring more abstract symbols (represented as squares and triangles) and organizing these symbols in an ontology (hierarchy of white rectangles). The fourth step consists of producing explained models of activity (round-angled rectangles) that are backed up by the abstract trace. During the modeling process, the analyst formalizes her understanding of the

activity in the form of transformation rules, ontology, and documentations that are stored in the system, which allows capitalizing on the analyst's knowledge across studies.

To illustrate these principles, we have implemented a prototype software tool through an assemblage of open source knowledge engineering software modules. With this tool, we have modeled car-driving activity traces collected with an instrumented vehicle. This analysis allowed us to identify and describe two strategies of lane change on motorways. This example shows that our knowledge engineering approach of activity modeling from activity traces offers answers to the needs for tools to help analysts understand better an observed activity, create models of this activity, report, and back up these models with the observational data.

The abstract activity traces that we have constructed constitute a model of the car driver in their own, in that the analysts can use our tool to query these traces to answer new questions they may have about the driving activity. Our future developments will consist of simulating the activity, for instance, in the field of car driving, generating realistic driving behavior in a driving simulator, based on our abstract activity traces.

ACKNOWLEDGMENTS

We would like to acknowledge support for this project from the European Commission through the HumanIST Network of Excellence. Partial support for this report was provided by ONR (contracts N00014-06-1-0164 and N00014-08-1-0481). We also gratefully thank Jean-Marc Trémeaux for his participation in the software implementation, and Matthias Henning for his contribution to the car-driving experiment. We appreciate the comments on this report from Jonathan Morgan and Ryan Kaulakis.

REFERENCES

AAMODT, A. & PLAZA, E. (1994) Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AI Communications,* 7**,** 39-59.

CORDIER, A., MASCRET, B. & MILLE, A. (2009) Extending Case-Based Reasoning with Traces. *Grand Challenges for reasoning from experiences, Workshop at IJCAI.* Pasadena, CA, 23-32.

CRAM, D., FUCHS, B., PRIÉ, Y. & MILLE, A. (2008) An approach to User-Centric Context-Aware Assistance based on Interaction Traces. *Modeling and Reasoning in Context. Human Centered Processes.* Delft, The Netherlands.

DEARDEN, A. & FINLAY, J. (2006) Pattern Languages in HCI: A Critical Review. *Human-Computer Interaction,* 21**,** 49-102.

ERICSSON, K. A. & SIMON, H. A. (1993) *Protocol analysis: Verbal reports as data,* Cambridge, MA, MIT Press.

GEORGEON, O., MORGAN, J., HORGAN, J. & BRADDOCK, K. (2010) Process Modeling for the Study of Non-State Political Violence. *19th Annual Conference on Behavior Representation in Modeling Simulation.* Charleston, NC, Brims Society, 240-247.

GROEGER, J. (2000) *Understanding driving,* Hove, UK, Psychology Press.

HENNING, M. J., GEORGEON, O. & KREMS, J. F. (2007) The quality of behavioral and environmental indicators used to infer the intention to change lanes. *4th International Driving Symposium on Human Factors in Driver Assessment.* Stevenson, Washington USA, 231-237.

HESSE-BIBER, S., DUPUIS, P. & KINDER, S. (2001) Testing hypotheses on qualitative data: the use of HyperRESEARCH computer-assisted software. *Social Science Computer Review,* 18**,** 320-328.

HILBERT, D. M. & REDMILES, D. F. (2000) Extracting Usability Information from User Interface Events. *ACM Computing Surveys (CSUR),* 32**,** 384-421.

JAMES, W. (1907) *Pragmatism*.

MAGNUSSON, M. S. (2000) Discovering hidden time patterns in behavior: T-Patterns and their detection. *Behavior Research Methods, Instruments and Computers,* 32**,** 93-110.

MILLE, A. (2006) From case-based reasoning to traces-based reasoning. *Annual Reviews in Control,* 30**,** 223-232.

OLLAGNIER-BELDAME, M. (2006) Traces d'interactions et processus cognitifs en activité conjointe :Le cas d'une co-rédaction médiée par un artefact numérique. *CNRS - LIRIS.* Lyon, Université Lumière Lyon 2.

OLSON, G. M., HERBSLEB, J. D. & REUTER, H. H. (1994) Characterizing the sequential structure of interactive behaviors through statistical and grammatical techniques. *Human -Computer Interaction,* 9**,** 427-472.

POPPER, K. (1972) *Objective knowledge,* Oxford, Oxford University Press.

RITTER, F. E. & LARKIN, J. H. (1994) Developing process models as summaries of HCI action sequences. *Human-Computer Interaction,* 9**,** 345-383.

RITTER, F. E. & WOOD, A. B. (2005) Dismal: A spreadsheet for sequential data analysis and HCI experimentation. *Behavior Research Methods, Instruments, and Computers,* 37**,** 71-81.

ROSE, R. T., QUEK, F. & SHI, Y. (2004) MacVisSTA: a system for multimodal analysis. *International Conference on Multimodal Interfaces.* State College, PA, USA, 259 - 264.

SALVUCCI, D. D. & LIU, A. (2002) The time course of a lane change: Driver control and eye-movement behavior. *Transportation Research,* F**,** 123-132.

SANDERSON, P. M. & FISHER, C. A. (1994) Exploratory sequential data analysis: Foundations. *Human-Computer Interaction,* 9**,** 251-317.

SANDERSON, P. M., MCNEESE, M. D. & ZAFF, B. S. (1994) Handling complex real-word data with two cognitive engineering tools: COGENT and MacSHAPA. *Behavior Research Methods, Instruments, and Computers,* 26**,** 117-124.

SETTOUTI, L. S., PRIÉ, Y., MARTY, J.-C. & MILLE, A. (2009) A Trace-Based System for Technology-Enhanced Learning Systems Personalisation *The*

9th IEEE International Conference on Advanced Learning Technologies. Riga, 93-97.

OLIVIER GEORGEON is a research associate in the LIRIS department at the Claude Bernard University (Lyon, France). He previously had a 12-year industrial experience as a software engineer, developer, and project manager in the domain of automatization of industrial processes. He received a PhD in cognitive psychology from the Université Lumière (Lyon, France) in 2008. His research interests are in *learning through activity*. His work both produces practical applications and addresses epistemological questions concerning learning from and about an activity.

ALAIN MILLE has been a professor at the Claude Bernard University (Lyon, France) since 2000. He is a scientific director in the LIRIS department (UMR 5205 CNRS), and leads the research group *Supporting Interactions and Learning through Experience*. After a first career as a computer engineer and a project manager (Hospices Civils de Lyon), Alain Mille was asked to build a Computer Science Department in a college of engineering (CPE-Lyon). In this latter position, he created a research team on Case Based Reasoning. The main topics were decision helping and complex tasks assistance. Alain Mille is in several national and international conference program committees, and on the editorial board of Intellectica (a cognitive sciences journal). He is

specifically interested in dynamic modeling of knowledge during computer-mediated activities.

THIERRY BELLET has been a researcher at the Ergonomics and Cognitive Sciences Laboratory of IFSTTAR (Institut Français des Sciences et Technologies des Transports, de l'Aménagement et des Réseaux) since 1999. His main areas of research concern human activity analysis and cognitive processes modeling (e.g., Situational Awareness, decision making, cognitive schemas). His research has produced computational simulations of mental activities of the car driver (e.g., COSMODRIVE: COgnitive Simulation MOdel of the DRIVEr).

BENOIT MATHERN is a PhD student in computer science. He first worked as a computer engineer at the Ergonomics and Cognitive Sciences Laboratory of IFSTTAR (Institut Français des Sciences et Technologies des Transports, de l'Aménagement et des Réseaux), then he started his PhD work in collaboration with the LIRIS department of the Claude Bernard University (Lyon, France). His main research interests focus on knowledge engineering, knowledge discovery and human-machine interaction. His work is applied to cognitive science in the field of transportation.

FRANK RITTER is one of the founding faculty of the College of IST, an interdisciplinary academic unit at Penn State to study how people process information using technology. Frank Ritter's current research is in the development, application, and methodology of cognitive models, particularly as applied to interface design, predicting

the effect of behavioral moderators, and understanding learning. He edits the Oxford Series on Cognitive Models and Architectures, is associate editor of *Cognitive Systems Research*, editorial board member of *Human Factors*, and the *Journal of Educational Psychology*, and technical program co-chair for the BRIMS 2009, 2010, and 2011 conferences and the associated special issues in *Computational and Mathematical Organizational Theory*.