Knowledge Acquisition by Abduction for Skills Monitoring: Application to Surgical Skills

C. Monserrat[1], J. Hernández-Orallo[1], J.F. Dolz[2], M.J. Rupérez[3], P. Flach[4]

[1]DSIC, Universitat Politècnica de València, Spain
[2]Área de Simulación Clínica y Seguridad del Paciente, Hospital Universitari i Politècnic La Fe de València, Spain
[3]Centro de Investigación en Ingeniería Mecánica, DIMM, Universitat Politècnica de València, Spain
[4]Intelligent Systems Laboratory, University of Bristol, UK
`cmonserr@dsic.upv.es`
`https://github.com/cmonserr/SUPERVASION`

Abstract. We present work in progress with the goal of automated monitoring by learning procedures from one high-level explanation given by an expert (a narrative) and a few video-recorded executions of the procedure (one in most cases). We refer to this problem as *automated supervision by observation*. For the acquisition of the procedural knowledge, we use the *induction by abduction* technique implemented in XHAIL, and Event Calculus for logical reasoning about events in time. The newly developed tool has been used to learn and monitor the training of novel surgeons in minimally invasive surgery. The results show the correct behaviour of the developed tool in the experiments developed until now.

Keywords: Automated Skills Supervision; Inductive Programming; Knowledge Modelling.

1 Introduction

We envision technologies that assist humans when they are learning or performing skilled tasks. We would like to create automated assistants that, after the observation of how an expert performs a task, are able to supervise whether other humans are performing the task correctly, also by observation. Unlike the usual interpretation of the term 'automated supervision', which is based on a human coding of a particular supervising checklist, we are aiming at a more challenging technology: the learning process should take place from one (or at most a few) demonstration(s), the tasks should be sequences of actions with a hierarchical structure and the acquired knowledge must be intelligible to humans at some of the levels of the hierarchy, especially, at the abstract ones.

According to these characteristics, an inductive programming approach [1] seems more appropriate. Given the tradition of representing states and actions in a logical setting, we will explore an inductive logic programming approach. In order to capture skills from video sequences in a sufficiently abstract way, we need to represent the sequences of actions detected during training. There are two main approximations using a logical representation: Situation Calculus [2] and Event Calculus (EC) [3]. We choose EC because the changes caused by the events are treated locally, affecting only the fluents or properties directly related to each event. In EC, a fluent is a property that is true along a period of time (e.g., closed) and an event (e.g., close_instrument) is anything that starts or ends a fluent.

To demonstrate the potential of the new paradigm of supervision by observation, we focus on Minimally Invasive Surgery (MIS) application domain. MIS requires long periods of surgeon training, requiring time of experts to give immediate feedback and guidance during each exercise [4].

These limitations have been significantly overcome by the introduction of virtual surgery simulators [5]. However, the metrics used in these virtual environments only evaluate but do not supervise [6]. Although new approaches are making it possible to detect wrong gestures and immediately suggest gesture corrections during a surgical training [6], these supervision systems are very low level, quite rigid and inefficient.

In the next sections we will describe how we have used ILP techniques in the surgical domain to first learn and then supervise the acquisition of skills by novice surgeons at a sufficiently abstract level to avoid these limitations.

2 Materials and methods

The supervision by observation has two sequential phases. The first phase is related to knowledge acquisition and the second phase uses this acquired knowledge to monitor the training of new practitioners. The software tool we use for the knowledge acquisition is XHAIL [7, 8], which makes "Induction by Abduction" and allows the acquisition of new knowledge using one narrative (high-level description provided by an expert) and some base knowledge (low-level events detected from a video sequence). Once knowledge is acquired, it can be used for monitoring new sequences of events and extract the high-level description from them. The latter is done by RTEC [10, 11] that allows fluents to be detected of any level of abstraction from sequences of low-level events and fluents in real time. We now explain each of the phases in detail.

PHASE 1.- Knowledge acquisition phase. Fig. 1 shows a schema of the knowledge acquisition process, with three inputs: the low-level events (directly observable), the narrative (high-level fluents) and the general and specific knowledge written in Discrete Event Calculus [12].

The sequence of events is the sequence of directly observable events usually extracted from video recordings, corresponding to videos of expert performing the training task. From these videos, the Event Builder constructs the sequence of observed events in EC language.

As Fig. 1 shows, the Event Builder has two elements: the Event Detector and the Sequencer. The Event Detector consists of a set of modules, each one

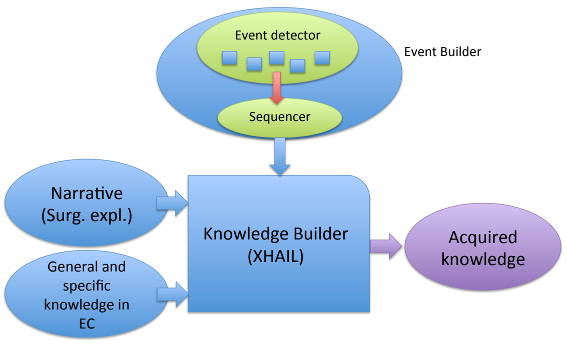

Fig. 1. Knowledge acquisition schema.

specialized in detecting one (or several) kind(s) of events. Each event detector module implements the best technique for the event to be detected. When Event Detector program detects an event E_i, it is sent with the time-frame information TF_i (a real value that represents the instant of time in which the event has been detected) to the Sequencer.

For each event E_i that arrives to the Sequencer, it generates a triplet $\{E_i$, T_i, $TF_i\}$, where T_i is a value that states the order in which one event happens and has associated one time-frame TF_i. This time-frame is the ones in which the event E_i was detected. The T_i values are computed as follows:

$$\begin{aligned}\{E_i, T_i, TF_i\} = \{E_i \in \mathit{events}, T_i \in \mathbb{N}, TF_i \in \mathbb{R} \mid \nexists j (\{E_j, T_j, TF_j\} \wedge \\ T_i < T_j \wedge TF_i > TF_j)\}\end{aligned} \tag{1}$$

Besides, in this implementation, the T_i's are generated strictly ordered:

$$\begin{aligned}T_i = T_j + 1 \leftrightarrow \exists i, j \Big(\{E_i, T_i, TF_i\} \wedge \{E_j, T_j, TF_j\} \wedge T_i < T_j \Big) \wedge \\ \nexists k \Big(\{E_k, T_k, TF_k\} \wedge T_i > T_k > T_j \Big)\end{aligned} \tag{2}$$

In the end, the Event Builder translates the sequence of triplets to an EC sequence of happened events. In this EC sequence, the T_i's are used as the time parameter in the EC predicates (see Table 1). The time-frame information (TF_i) is used internally for synchronizing the high-level narrative with the low-level events.

Back to Fig. 1, the Narrative consists of the properties that hold or not at any instant of time extracted from the expert high-level explanation and translated to a sequence of fluents in EC. Finally, the General and Specific Knowledge input really consists of two different dialects of Discrete Event Calculus [12]. One dialect is used for managing low-level events and the other is used to manage high-level fluents. Table 1 shows the predicates and axioms used in both EC dialects.

Table 1. The EC dialects used: predicates and axioms of both dialects are separated by dash lines.

Predicates:	
holdsAt(F,T)	Fluent F is true at time T.
happensAt(E,T)	Event E occurs at time T.
initiates(E,F,T)	Event E initiates (makes true) the fluent F from time T+1.
terminates(E,F,T)	Event E terminates (makes false) the fluent F from time T+1.
initially(F)	Fluent F is initially true.
stoppedIn(T1,F,T2)	Fluent F is terminated in an instant of time between T1 and T2.
....	....
holdsAtF(FF,T)	High-level fluent FF is true at time T.
initiatesF(FF,T)	High-level fluent FF is true from time T+1.
terminatesF(FF,T)	High-level fluent FF is false from time T+1.
General Axioms from EC:	
holdsAt(F,T) :- initially(F), not stoppedIn(0,F,T).	
holdsAt(F,T) :- happensAt(E,T1), initiates(E,F,T1), not stoppedIn(T1,F,T), T1¡T.	
stoppedIn(T1,F,T2) :- happensAt(E,T), T1¡T, T¡T2, terminates(E,F,T).	
....	
holdsAtF(FF,T+1) :- not holdsAtF(FF,T),initiatesF(FF,T).	
holdsAtF(FF,T+1) :- holdsAtF(FF,T), not terminatesF(FF,T).	

These three inputs are sent to XHAIL in order to acquire new knowledge. The hypotheses to be obtained are biased [7] by the predicates that detect when one high-level fluent is initiated (initiatesF/2) or terminated (terminatesF/2) (head of the new rules) based on which low-level fluents hold (holdsAt/2) or not (body of the new rules).

PHASE 2.- Online supervision phase. Fig. 2 shows a schema of the training monitoring process. As this schema shows, the training monitoring process has three inputs: the real-time sequence of events, the knowledge obtained in phase 1 and the specific knowledge written in Discrete Event Calculus. The general knowledge is already integrated in RTEC.

The Event Builder works exactly in the same manner as in Phase 1 except that, in this case, it detects the events in real time while the novice is doing the training. The Specific Knowledge used in Phase 2 is also based on Discrete Event Calculus but managing fluents through their Maximum Validity Interval [13]. The last input, the Acquired Knowledge, is formed by the rules learned in Phase 1 adapted to RTEC software.

The output of the Online supervision phase is the detection of when one high-level fluent starts and when this high-level fluent ends using the acquired knowledge over executions different from the one used in the learning process (Phase 1). RTEC provides *temporal-windows* inside which the learned rules are true. The success of the detection process depends on whether the initiatesF/2 or terminatesF/2 of the new executions are inside the corresponding *temporal-windows*.

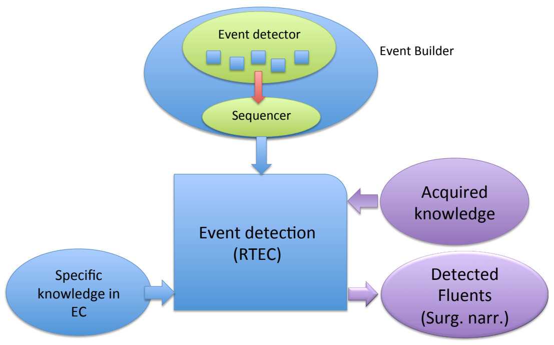

Fig. 2. Online supervision schema.

3 Results

The developed tool is being used in monitoring the training of novel surgeons. With this goal, in this very initial implementation, we have analysed the ability of the system to learn and monitor one high-level fluent: *clipping*. According to the expert description, the clipping phase starts just before one object $o1$ (contained in one bin $g1$) is clamped by an instrument $i1$ and finishes just when this object is released (in a second bin $g2$). The described process is applied to the execution of a basic training of MIS skills in two scenarios (scenario A and scenario B). The learning process will be used in both executions. Then, the new knowledge acquired from execution in the scenario A is used to detect the high level fluents in the execution in the scenario B and vice versa.

According to the described process, the rules learned for instrument $i1$ from the execution in scenario A were:

initiatesF(clip_object(i1,o1),T) :- holdsAt(open(i1),T), hold-sAt(in5(i1,g1),T).
terminatesF(clip_object(i1,o1),T) :- holdsAt(waiting(i1),T), holdsAt(in5(i1,g2),T).

where the fluent clip_object/2 is a high-level fluent with two parameters: instrument ($i1$) and object ($o1$). This fluent is true when the process of $i1$ clipping $o1$ holds. open/1 is a low-level fluent that is true when the instrument that appears as parameter is open (in this case is $i1$). Finally, in5/2 is a low-level fluent that is true when the tip of one instrument (in this case $i1$) is less than 5 mm from one bin (in this case it could be $g1$ or $g2$). On the other hand, the rules learned for instrument $i1$ in the execution in the scenario B were:

initiatesF(clip_object(i1,o1),T):-holdsAt(in5(i1,g1),T),not holdsAt(towards(i1,g1),T).
terminatesF(clip_object(i1,o1),T):-holdsAt(in5(i1,g2),T),not holdsAt(towards(i1,g2),T).

where towards/2 is a low-level fluent that is true when an instrument (in this case $i1$) is approaching the bin passed as parameter (it could be $g1$ or $g2$).

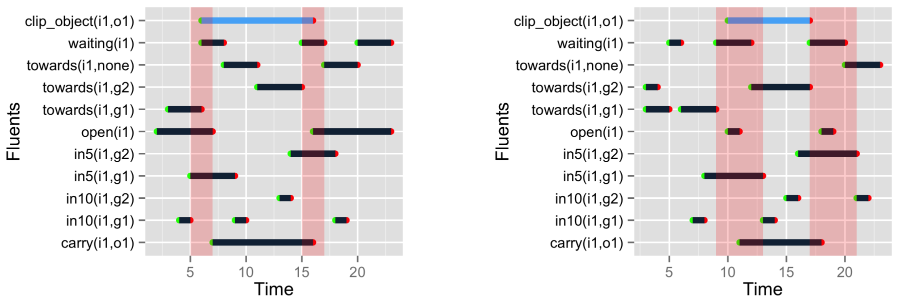

Fig. 3. Initial and ending points of the fluent clip_object/2 in (left) one execution en scenario A; (right) one execution in scenario B.

Fig. 3 shows the fluents that corresponds to one execution of the clip process in both scenarios. The Y-axis shows the low-level (represented by black segments in the graph) and high-level (represented by light segments in the graph) fluents of each execution. The vertical bands are the temporal windows for the initiatesF/2 (the left one of each graph) and for the terminatesF/2 (the right one) obtained from the learned rules. As can be seen, the high-level initiatesF/2 and terminatesF/2 for clipping are inside of the corresponding temporal windows.

The results shown correspond to a first simple example. More examples have been developed. The analysis of one more complex (with two instruments $i1$ and $i2$) is shown in `https://github.com/cmonserr/SUPERVASION/wiki`. The source code of these experiments can be found in the same GitHub entry.

4 Conclusions and future work.

The initial results show that the new knowledge acquired in one training can be used to recognize behaviours in different executions of the same exercise. The same process has also been applied to a second tool and inverted process and with both tools at the same time (none of these experiments are included due to lack of space) having similar results. However, more (complex) learning exercises should be used to fully validate the approach. Besides, the knowledge acquired tends to overfit the exercise from which it is obtained. An option to avoid overfitting could be the use of weights in the narrative provided to XHAIL [9].

Aknowledgements. This research is partially founded by a research stay at the University of Bristol (PRX15/00079) by the MECD and the project SUPERVASION (TIN2014-61716-EXP) by the MINECO of Spain. We also would like to give thanks to Dr. Alexander Artikis for his help in the use of RTEC and Dr. Oliver Ray for his support in the use of XHAIL.

References

1. Gulwani, S., Hernández-Orallo, J., Kitzelmann, E., Muggleton, S.H., Schmid, U., Zorn, B.: Inductive programming meets the real world. Communications of the ACM, pp. 90–99, (2015)
2. McCarthy, J., Hayes, P.J.: Some philosophical problems from the standpoint of artificial intelligence. In: Meltzer and Michie (eds.) Machine Intelligence, vol. 4, pp. 463–505. University Press (1969)
3. Kowalski, R, Sergot, M.: A logic-based calculus of event. In: Foundation Knowledge Based Management, pp. 23–55. Springer, Berln Heilderberg (1986)
4. Porte, M.C., Xeroulis, G., Reznick, R.K., Dubrowski, A.: Verbal feedback from an expert is more effective than self-accessed feedback about motion efficiency in learning new surgical skills. Am. J. Surg. 193(1), 105–110 (2007)
5. Satava, R.M.: Virtual reality surgical simulator. Sug. Endosc. 7, 203–205 (1993)
6. Monserrat, C., Lucas, A., Hernández-Orallo, J., Rupérez, M.J.: Automatic supervision of gestures to guide novice surgeons during training. Sug. Endosc. 28(4), 1360–1370 (2014)
7. Ray, O.: Nonmonotonic abductive inductive learning. J. Appl. Logic 7(3), 329–340 (2009)
8. System for eXtended Hybrid Abductive Inductive Learning, `https://github.com/stefano-bragaglia/XHAIL`
9. Bragaglia, S., Ray, O.: Nonmonotonic Learning in Large Biological Networks. In: Davis, J., Ramon, J., (eds.) Proc. 24th Int. Conf. on Inductive Logic Programming (ILP), LNAI 9046:33–48, (2015)
10. Artikis, A., Sergot, M., Paliouras, G.: An Event Calculus for Event Recognition. IEEE Transactions on Knowledge and Data Engineering, 22(4), 895–908, (2015)
11. Run-Time Event Calculus, `https://github.com/aartikis/RTEC`
12. Mueller, E.T.: Event Calculus. In: Harmelen, F., Lifschitz, V., Porter, B. (eds.) Handbook of Knowledge Representation 2008, pp. 671–708. Elsevier (2008)
13. Chittaro, L., Montanari, A.: Efficient Temporal Reasoning in the Catched Event Calculus. Computational Intelligence, 12 (3), 359–382 (1996)