A Learning System for Decision Support in Telecommunications

Filip Železný[1], Jiří Zídek[2], Olga Štěpánková[3]

[1] Center for Applied Cybernetics, [3] The Gerstner Laboratory
[1,3] Faculty of Electrotechnics, Czech Technical University
Prague, Czech Republic
Technická 2, CZ 166 27, Prague 6
{zelezny,step}@labe.felk.cvut.cz
[2] Atlantis Telecom s.r.o.
Žirovnická 2389, CZ 106 00, Prague 10
zidek@atlantis.cz

Abstract. We present a system for decision support in telecommunications. History data describing the operation of a telephone exchange are analyzed by the system to reconstruct understandable event descriptions. The event descriptions are processed by an algorithm inducing rules describing regularities in the events. The rules can be used as decision support rules (for the exchange operator) or directly to automate the operation of the exchange.

1 Introduction

In spite of the explosion of information technologies based on written communication, the most common and most frequently used tool is the telephone. Up-to-date private branch exchanges (PBX) provide comfort in managing the telephone traffic, namely regarding calls coming into an enterprise from the outside world. Communication proceeds smoothly provided that the caller knows with whom she wants to communicate and the person is available. In the opposite case, there is a secretary, receptionist, operator or colleague that can for instance help to find a substituting person. The operator is a person with no direct product, but with strong impact on productivity of other people. Despite that, a wide range of companies cancel the post of the telephone operator. The reason is that it is not easy to find a person who is intelligent enough to be good operator and to be modest enough to be just an operator. This opens area for computers - the computer is paid for only once so no fix costs set in. Moreover, the machine can work non-stop and provide additional data suitable for analysis allowing for improvements of the telecommunication traffic.

Currently there are several domains where computers are used in the PBX area (neglecting the fact that PBX itself is a kind of computer):

– *Automated attendant* - a device that welcomes a caller in a unified manner and allows him usually to reach a person, or choose a person from a spoken list; in both cases the calling party is required to co-operate

- *Voice mail* - a device allowing to leave a spoken message to an unavailable person, and some rather sophisticated methods of delivering the messages are available
- *Information service* - the machine substitutes a person in providing some basic information usually organized into an information tree; the calling party is required to co-operate

The aim of the above listed tools is to satisfy a caller even if there is no human service available at the moment. But all such devices are designed in a static, simple manner - they act always the same way. The reason is simple - they do not consider who is calling nor what they usually want - as opposed to the human operator. Comparing a human operator/receptionist to a computer, we can imagine the following improvements of the automated telephony:

1. Considering who is calling (by the identified calling party number) and what number was dialled by the caller, the system can learn to determine the person most probably desired by the caller; knowledge can be obtained either from previous cases (taking into account other data like daytime, explicit information - long absence of some of the company's employee, etc.) or by 'observing' the way how the caller was handled by humans before; this could shorten the caller's way to get the information she needs.
2. The caller can be informed by a machine in a spoken language about the state of the call and suggested most likely alternatives; messages should be 'context sensitive'.

Naturally, the finite goal of a computerized telephony is a fully 'duplex' machine that can both speak and comprehend spoken language so that the feedback with the caller can proceed in a natural dialog.

We present a methodology where the goal is to satisfy the goal 1. The task was defined by a telecommunication company that installs PBX switchboards in various enterprizes. Our experiments are based on the PBX logging data coming from one of the enterprises. The methodology is reflected in a unified system with inductive (learning) capabilities employed to produce decision support rules based on the data describing the previous PBX switching traffic. The system can be naturally adapted to the conditions of a specific company (by including a formally defined enterprise-related background knowledge) as well as in the case of a change in the PBX firmware (again via an inductive learning process).

We employ the language of Prolog [3, 5] (a subset of the language of first-order logic) as a unified formalism to represent the input data, the background knowledge, the reasoning mechanism and the output decision support rules. The reason for this is the structured nature of the data with important dependencies between individual records, and the fact that sophisticated paradigms are available for learning in first-order logic. These paradigms are known as Inductive Logic Programming (ILP) [9, 7]. The fundamental goal of ILP is the induction of first-order logic theories from logic facts and background knowledge. In the recent years, two streams of ILP have developed, called the *normal* setting (where - roughly - theories with a 'predictive' nature are seeked) and the *non-monotonic* setting (where the theories have a 'descriptive' character). We employ both of the settings in the system and their brief description will be given in the respective sections.

The paper is further organized as follows. The next section describes the data produced by the PBX. In Sections 3 and 4 we deal with the question of how to reconstruct events from the data, i.e. how to find out what actions the callers performed. In Section 5 we shall describe the way we induce decision support rules from the event database and appropriate background knowledge. Section 6 shows the overall interconnection of the individual learning/reasoning mechanisms into an integrated system.

A rough knowledge of the syntax of Prolog clauses (rules) is needed to understand the presented examples of the learning and reasoning system parts.

2 The Exchange and its Data

The raw logging file of the PBX (MC 7500) is an ASCII file composed of metering receipts (tickets) describing 'atomic events'. The structure of such a ticket is e.g.

```
4AB000609193638V1LO       1                  12193650EDILBRDD
EX          0602330533      005000 1FEFE
```

This ticket describes a single unanswered ring from the external number 0602330533 on the internal line 12. To make the information carried by the ticket accessible to both the human user and the reasoning mechanisms, we convert the tickets description into a relational-table form obtained means of the data transformation tool Sumatra TT [2] developed at CTU Prague. A window into the relational table is shown in Figure 1. The numbered columns denote the following attributes extracted from the ticket and related to the corresponding event: 1: date, 2: starting time, 3: monitored line, 4: end time, 5: call type (E - incoming, S - outgoing), 6: release type (LB - event terminated, LI - event continues in another ticket), 7: release cause (e.g. TR - call has been transfered), 8: call setup (D - direct, A - result of a previous transfer), 11: call nature (EX - external, LO - local, i.e. between internal lines etc.), 12: corresponding party number, 14: PBX port used, 17: unique ticket key. Attributes not mentioned are not crucial for the explanation sakes.

A complete event, i.e. the sequence of actions (e.g. transfers between lines) starting with an answered ring from an outside party and ending with the call termination, is reflected by two or more tickets. For example, a simple event such as an external answered (non-transferred) call will produce two tickets in the database (one for the ring, another for the talk). Figure 1 contains records related to two simultaneous external calls, each of which was transferred to another line after a conversation on the originally called line. The first problem of the data-analysis is apparent: tickets related to different events are mixed and not trivially separable. Moreover, although calls originating from a transfer from

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
000802	085151	32	085151	E	LI		D			EX	0405353377		005001	FE	FE	17664
000802	085158	10	085201	E	LI		D			LO	32			0	4	17665
000802	085201	10	085205	E	LI		D	DR	32	LO	32		005001	0	4	17666
000802	085151	32	085205	E	LB	TR	D	DR	06	EX	0405353377		005001	FE	FE	17667
000802	085158	32	085205	S	LB	TR	D		6	LO	10	10	10	0	0	17668
000802	085207	31	085207	E	LI		D			EX	85131111		005009	FE	FE	17669
000802	085218	11	085218	E	LI		D			LO	31			0	3	17670
000802	085218	11	085223	E	LI		D	DR	31	LO	31			0	3	17671
000802	085207	31	085223	E	LB	TR	D	DR	72	EX	85131111		005009	FE	FE	17672
000802	085214	31	085223	S	LB	TR	D			LO	11	11		0	0	17673
000802	085223	11	085339	E	LB		A	DR	31	EX	85131111		005009	FE	FE	17674
000802	085205	10	085424	E	LB		A	DR	32	EX	0400000000		005001	FE	FE	17675

Fig. 1. A window into the PBX logging data containing two simultaneous calls.

a preceding call can be identified (those labelled A in the attribute 8), it cannot be immediately seen from which call they originate. Moreover, we have to deal with an erroneous way of logging some instances of the external numbers by the PBX, e.g. the number 0400000000 in Figure 1 actually refers to the caller previously identified as 0405353377. This problem will be discussed in Section 4.1.

3 Event Extraction

The table in Figure 1 can be visualized graphically as shown in Figure 2. The figure visually distinguishes the two recorded simultaneous calls, although they are not distinguished by any attribute in the data. The two events are as follows: caller 0405353377 (EX1) is connected to the receptionist on the line 32, asks to be transferred onto line 10. After a spoke notification from 32 to 10, the redirection occurs. During the transferred call between EX1 and 10, a similar event proceeds for caller 85131111 (EX2), receptionist 31 and the desired line 11. It can be seen that the duration of each of the two events (transferred calls) is covered by the durations of the external-call tickets related to the event. Furthermore, the answering port (attribute 14) recorded for each of the external-call ticket is constant within one event and different for different simultaneous events. In other words, a single external caller remains connected to one port until she hangs up, whether or not she gets transferred to different internal lines.

This is an expert-formulated, generally valid rule which can be used to delimit the duration of particular events (Figure 3). The sequences of connected external-call tickets are taken as a base for the event recognition. We have implemented the event extractor both as a Prolog program and as a set of SQL queries. However, additional tickets (besides the external-call tickets) related to an event have to be found in the data to recognize the event. For instance, both of the events reflected in Figure 2 contain in fact a transfer-with-notification action (e.g. line 32 informs line 10 about the forthcoming transfer before it takes place), which can be deduced from the three tickets related to the internal line communication within each of the events.

23.35 % of all tickets in the experimental database fall each into one of the extracted events. The rest of the communication traffic thus consists of internal or outgoing calls.

4 Event Reconstruction

Having obtained an event delimitation from the event extractor as a sequence of external-call tickets, we need to look up the database for all other tickets related to that event. According to the these tickets we can decide on what sequence of actions occurred during the event, such as different kinds of call transfers (direct, with notification), their outcome (refusal, no answer, line busy), returns to the previous attendant etc. The way such actions are reflected in the ticket database depends on the current setting of the PBX firmware and an appropriate formal mapping $\{\textit{events}\} \rightarrow \{\textit{sets of inter-related tickets}\}$ is not available.

Such mapping can however be obtained via an inductive learning process which will discover ticket patterns for individual actions from classified examples, i.e. completely described events. These classified examples were produced by intentionally performing a set of actions on the PBX and storing separately the tickets generated for each of the actions. The discovered (first-order) patterns will then be used to recognize transitions of an automaton formally describing the course of actions within an event.

4.1 Learning Action Patterns

The goal of the action-pattern learner is to discover ticket patterns, i.e. the character of the set of tickets produced by the PBX in the logging data as a result of performing a specific action. This regards the occurrence of individual tickets in the set, their mutual order and/or (partial) overlapping in time etc.

For this purpose, tickets are represented in the Prolog fact syntax[1] as

$$\texttt{t(...,}a_n,a_{n+1}\texttt{,...)}$$

where a_i are ticket attributes described earlier and the irrelevant date attribute is omitted. The constant `empty` will stand for a blank field in the data. The learner has two inputs: the classified event examples (sets of Prolog facts) and a *general background knowledge* (GBK). Following is a single instance of the example set, composed of facts bound to a single event, namely two facts representing tickets, and two facts representing the actions in the event: an external call from the number 0602330533 answered by the internal line 12 and the call

[1] Such a representation is obtained simply by a single pass of the Sumatra TT transformation tool on the original data

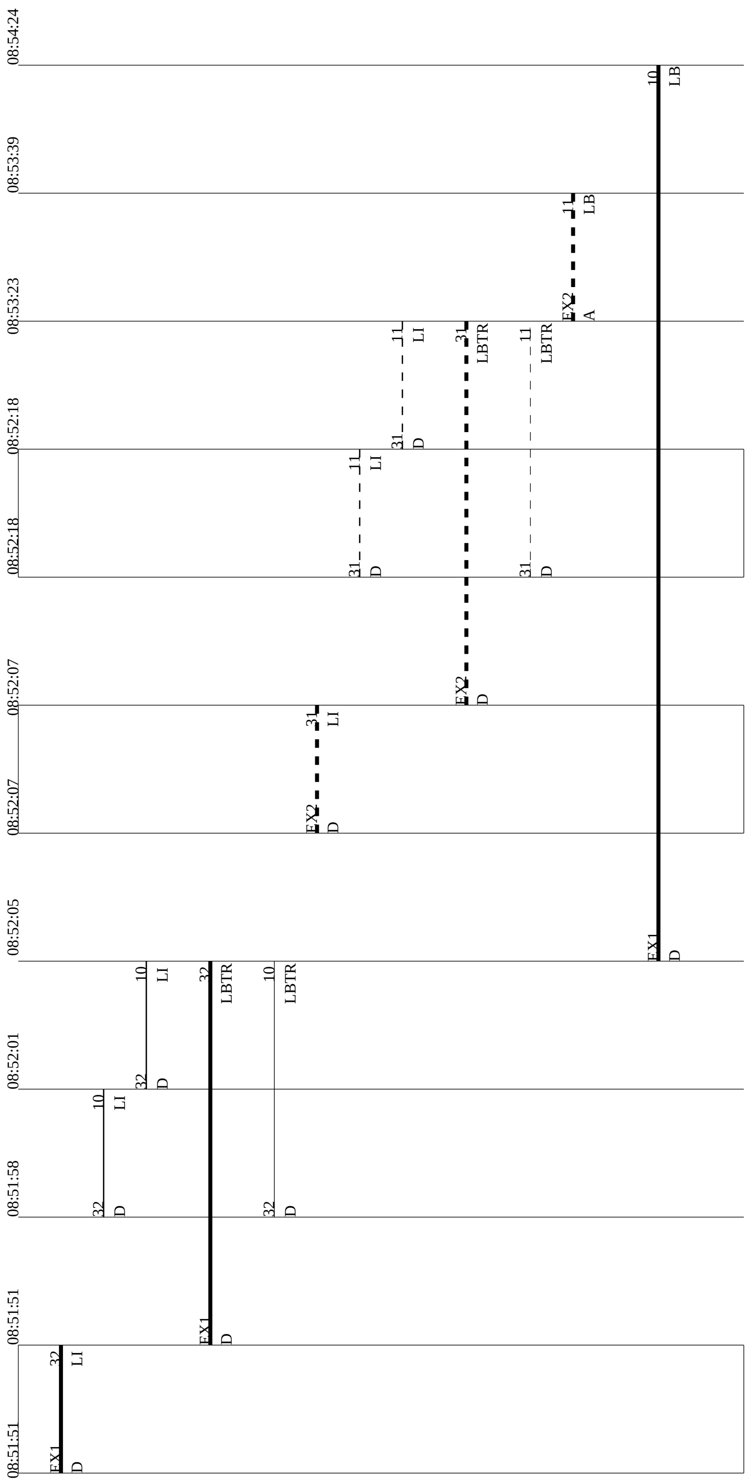

Fig. 2. Visualizing the chronology of the telecommunication traffic contained in the table in Figure 1. Vertical lines denote time instants, labelled horizontal lines denote the duration and attributes given by one ticket. For each such line, the upper-left / upper-right / lower-left / lower-right attributes denote the calling line, called line, call setup attribute and release type + cause, respectively. The abbreviations EX1 and EX2 represent two different external numbers. Thin lines stand for internal call tickets while thick lines represent external call tickets. The vertical position of the horizontal lines reflects the order of the tickets in the database. For ease of insight, tickets represented by dashed lines are related to a different call than those with full lines

TICKETS
EVENT EXTRACTOR
EVENT DELIMITATIONS

Fig. 3. The event extraction.

termination caused by the external number hanging up.[2]

```
t(time(19,43,48),[1,2],time(19,43,48),e,li,empty,d,empty,empty,ex
   [0,6,0,2,3,3,0,5,3,3],empty,anstr([0,0,5,0,0,0]),fe,fe,id(4)).
t(time(19,43,48),[1,2],time(19,43,50),e,lb,e(relcause),d,dr,06,ex
   [0,6,0,0,0,0,0,0,0,0],empty,anstr([0,0,5,0,0,0]),fe,fe,id(5)).
ex_ans([0,6,0,2,3,3,0,5,3,3],[1,2]).
hangsup([0,6,0,2,3,3,0,5,3,3]).
```

The general background knowledge GBK describes certain apriori known properties of the PBX. For example, due to a malfunction, the PBX occasionally substitutes a suffix of an identified external caller number by a sequence of zeros (such as in the second fact above). The correct and substituted numbers have to be treated as identical in the corresponding patterns. Therefore one of the rules (predicate-definitions) in GBK is `samenum(NUM1,NUM2)` that unifies two numbers with identical prefixes and different suffixes, one of which is a sequence of zeros.

To induce patterns from examples of the above form and the first-order background knowledge GBK, we constructed an ILP system working in the non-monotonic ILP setting (known also as *learning from interpretations*). The principle of this setting is that given a first order theory B (background knowledge), a set of interpretations (sets of logic facts) E and a grammar G, we have to find all first-order clauses (rules) c included in the language defined by the grammar G, such that c is true in $B\&e$ for all $e \in E$.[3] In our case, E is the set of classified events, $B =$ GBK and G is defined so that it produces rules where tickets and their mutual relations are expressed in the rule's Body and the action is identified in the rule's Head. To specify G we integrated the freely available DLAB [4] grammar-definition tool into our ILP system. Besides grammar specification, DLAB also provides methods of clausal refinement, so we could only concentrate on clause validity evaluation and implementing the (pruning) search through the space of clauses.

An example of a generated pattern found to be valid for all of the collected examples is the following, describing which combination of tickets and their relationship specified by the equalities in the rule's body reflects the action of

[2] Both external and internal line numbers are represented as *Prolog lists* to allow easy access to their substrings.

[3] A clause $c = Head : -Body$ is true in $B\&e$ if both B and e are stored in a Prolog database and the Prolog query $? - Body, not\ Head$ against that database does not succeed.

answering a direct (non-transferred) external call.[4]

```
ex_ans(RNCA1,DN1):-
   t(IT1,DN1,ET1,e,li,empty,d,EF1,FI1,ex,RNCA1,empty,ANTR1,CO1,DE1,ID1),
   IT2=ET1,
   ANTR2=ANTR1,
   t(IT2,DN2,ET2,e,lb,RC2,d,EF2,FI2,ex,RNCA2,empty,ANTR2,CO2,DE2,ID2),
   samenum(RNCA1,RNCA2).
```

The time order of the involved tickets is determined by the equality $IT2 = ET1$ in the rule (with the variables $IT2, ET1$ referring to the initial time of the second ticket and the end time of the first ticket, respectively).

Using the described approach to generate rules for other actions as well, we create a database of action patterns (as shown in Figure 4). Since we had known some of the patterns from experience in the manual data analysis, this process was both a theory discovery and theory revision. The final action pattern database is thus a combination of induction results and explicit knowledge representation. The database can be kept static as long as the PBX exchange firmware (i.e. the exact manner of logging) remains unchanged. The process should be repeated when the firmware is modified and the logging procedures change.

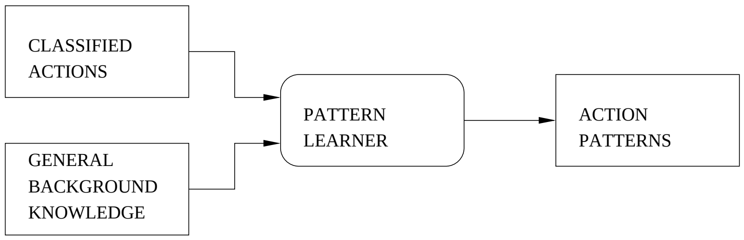

Fig. 4. Learning action patterns.

4.2 Event Recognizing Automaton

To discover the sequence of actions in an event, we assume that every event (starting with an incoming call) can be viewed as a simple finite-state automaton shown in Figure 5. Each transition corresponds to one or more actions defined in the action pattern database (e.g. 'Attempt to transfer' corresponds to more kinds of the transfer procedure). The automaton (event reconstructor) is encoded in

[4] Remind that capital letters stand for universally quantified variables in the Prolog syntax.

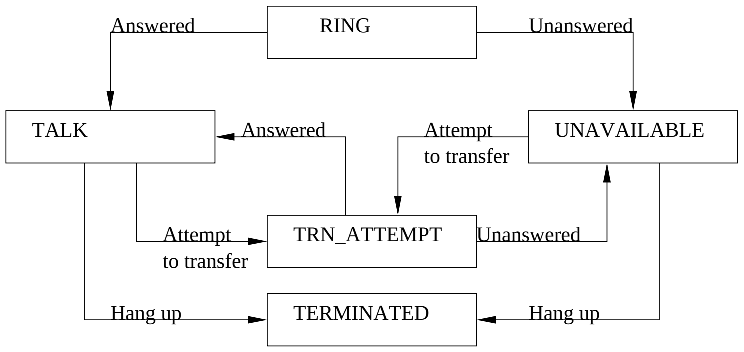

Fig. 5. The states and transitions of the event automaton. In this representation of the PBX operation, the sequence *RING* → *Unanswered* → *UNAVAILABLE* → *Attempt to transfer* cannot occur because the caller is not assisted by a person on an internal line.

Prolog. It takes as input an event delimiting sequence S (produced by the event extractor) and the action-pattern database. In parsing S, the patterns are used to recognize transitions between the states. Since the patterns may refer to GBK and also to tickets not present in S (such as transfer-with-notification patterns - see Figure 2), both GBK and the ticket database must be available to the automaton. This dataflow is depicted in Figure 6.

Regarding the output, one version of the reconstructor produces human-understandable descriptions of the event, such as in the following example.

```
? - recognize([id(60216),id(60218),id(60224),id(60228),id(60232),id(60239)])
EVENT STARTS.
648256849 rings on 32 - call accepted,
32 attempts to transfer 0600000000 to 16 with notification, but 16 refused,
32 notifies 12 and transfers 0648256849 to 12,
12 attempts to transfer 0600000000 to 28 with notification, but 28 does not
respond,
12 notifies 26 and transfers 0600000000 to 26,
call terminated.
EVENT STOPS.
```

An alternative version of the reconstructor produces the descriptions in the form of structured (recursive) Prolog facts of the recursive form

`incoming(DATE,TIME,CALLER,FIRST_CALLED_LINE,RESULT)`,

where

$$\texttt{RESULT} \in \{talk, unavailable, \texttt{transfer}([t_1, t_2, ..., t_n], \texttt{RESULT})\} \qquad (1)$$

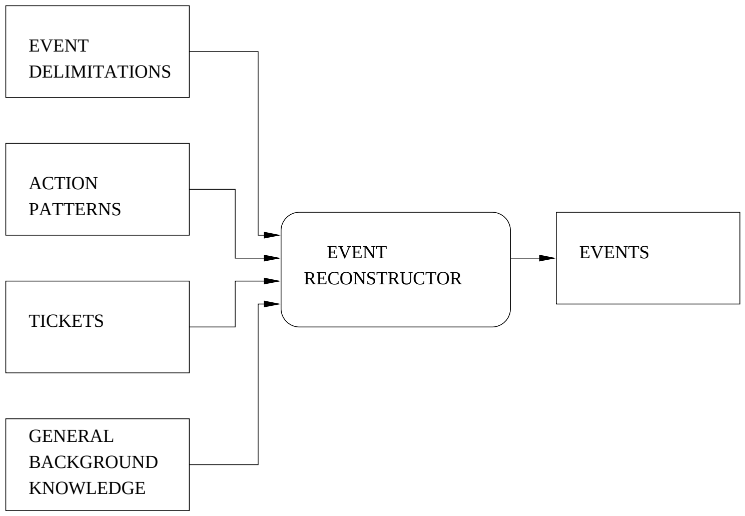

Fig. 6. The event reconstruction.

and $t_1 ... t_{n-1}$ denote line numbers to which unsuccessful attempts to transfer were made, and the transfer result refers to the last transfer attempt (to t_n). According to this syntax, the previous example output will be encoded as

```
incoming(date(10,18),time(13,37,29),[0,6,4,8,2,5,6,8,4,9],[3,2],
   transfer([[1,6],[1,2]],transfer([[2,8],[2,6]],talk))).
```

(2)

and in this form used as the input to the inductive process described in the next section.

The effectiveness of the described recognition procedure is illustrated in Figure 7. It can be seen that the method results in a very good coverage (recognition) of events containing more than 2 tickets. For the shorter events, more training examples will have to be produced and employed in learning to improve the coverage.

5 Decision Support

Having reconstructed the events from the logging file, i.e. knowing how different external callers have been handled in the history, the system can find regularities in the data, according to which some future events may be partially predicted (extrapolated) or even automatized. For example, it may be found that if the caller identified with her number N calls the receptionist, she always desires to be transferred to line L. Then it makes sense transfer N to L automatically

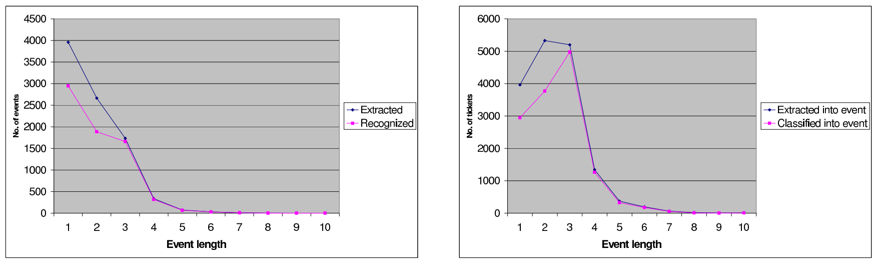

Fig. 7. Portions of extracted events of individual lengths (number of tickets) that are recognized by the recongition automaton (left). Portions of tickets classified into a recongized event of all tickets extracted into events of different lenghts (right). The complete ticket database contains about 70.000 tickets ranging within a 3 month operation of the PBX.

upon the ring without the assistance of the receptionist (provided that N can be transferred to another line by L if the prediction turns out to be wrong). Or, it may be a regular observation that if the person on line L_1 is not available, line L_2 is always provided as a substitute - this again offers an automation rule, or at least a decision support advice to the receptionist.

A suitable methodology for inducing predictive rules from our data is the normal inductive logic programming setting, where the goal is (typically) the following. Given a first-order theory (background knowledge) B, two sets of logic facts E^+, E^- (positive and negative examples), find a theory T such that

1. $B\&T \vdash e^+$ for each $e^+ \in E^+$
2. $B\&T \nvdash e^-$ for each $e^- \in E^-$

I.e. we require that any positive (negative) example can (cannot) be logically derived from the background knowledge and the resulting theory.

In our case, B is composed of the GBK described earlier and an *enterprise-related* background knowledge (EBK). EBK may describe e.g. the regular (un) availability of employees. The E^+ set contains the event descriptions given by the predicate `incoming` such as the example 2. The negative example set is in our case substituted by *integrity constraints* that express for instance that a call cannot be transferred to two different lines etc. The resulting theory is bound not to violate the integrity constraints.

Our experiments in the decision support part of the system have been reported in detail elsewhere [10], therefore we just mention one example of a resulting rule valid with accuracy 1 on the training set. The rule

```
incoming(D,T,EX,31,transfer([10|R],RES)) :- day_is(monday,D),branch(EX,[5,0]).
```

employs the predicates `day_is` whose meaning is obvious and `branch` which identifies external numbers by a prefix. Both of the predicates are defined in

EBK. The rule's meaning is that if a number starting with 50- calls on Monday on the (reception) line 31, the caller always desires to be transferred to line 10 (whatever the transfer result is).

See the sources [10] for a detailed overview of the predictive-rule induction experiments.

Figure 8 summarizes the data-flow in the system's decision support part. The performance of the decision-support rules is being tested so far only in the experimental environment and the implementation in the enterprize is currently under construction.

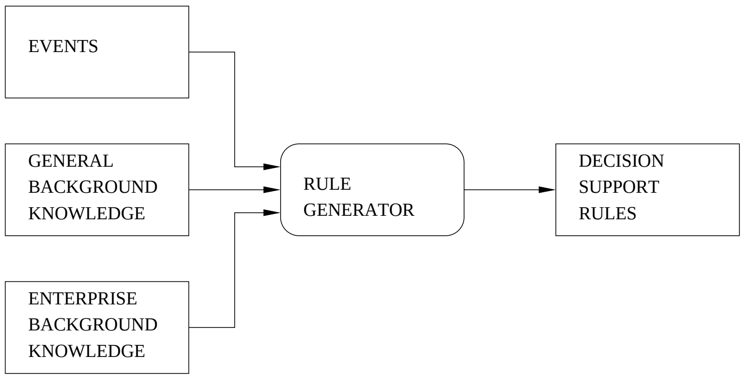

Fig. 8. Generating decision support rules.

6 System Integration Overview

Figure 9 shows how the previously described individual system parts are integrated. The fundamental cycle is the following: the PBX generates data that are analyzed to produce decision support rules which then again influence the operation of the PBX (with or without a human assistence).

7 Conclusions

We have presented a system for decision support in telecommunications. The system analyzes data stored by a private branch exchange to reconstruct understandable event descriptions. For this purpose, action patterns are learned from classified examples of actions. The event descriptions are processed by an algorithm that induces rules describing event regularities which can be used as decision support rules (for the exchange operator) or directly to automate the PBX operation.

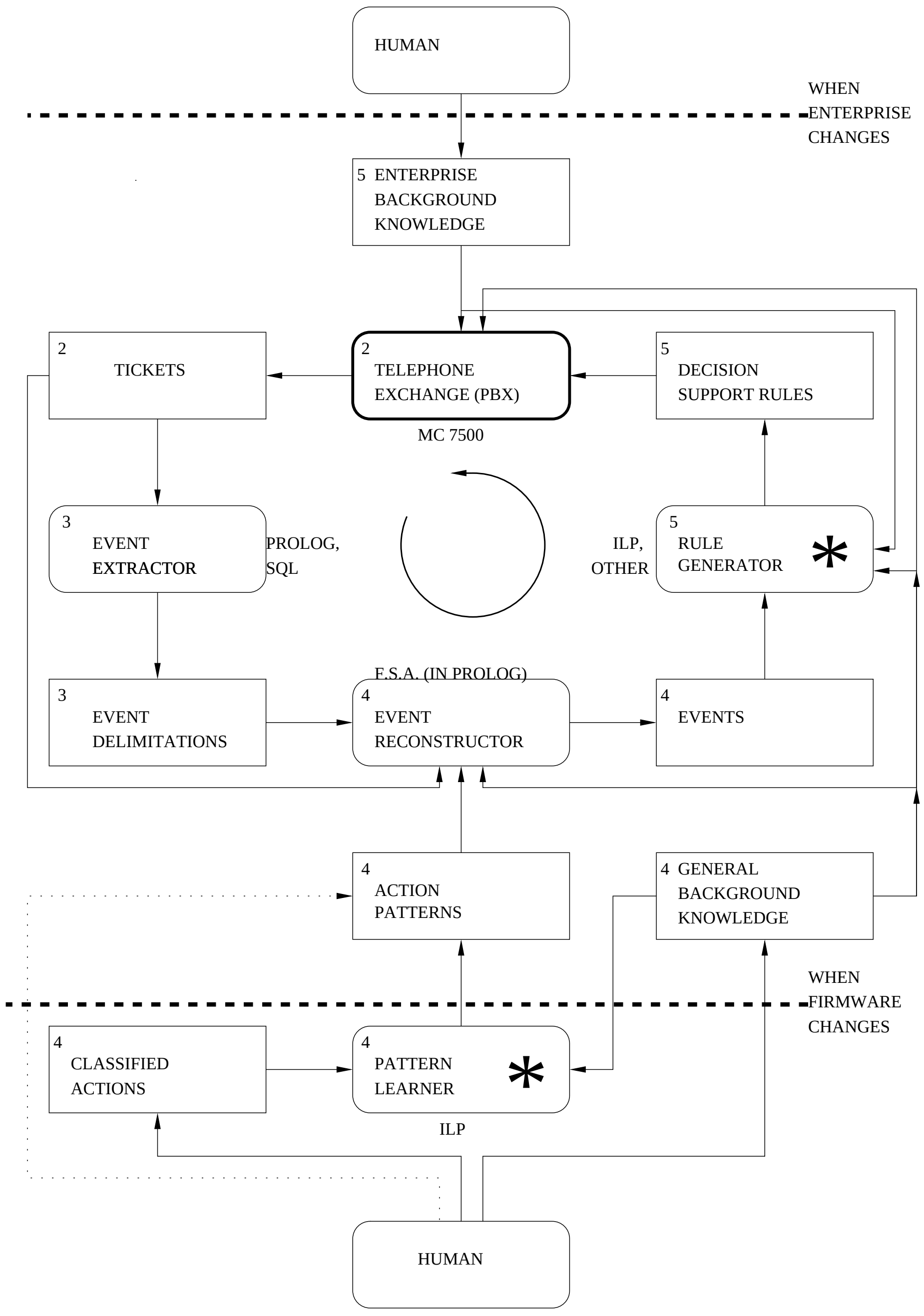

Fig. 9. The system integration overview. The dataflow over the upper (lower) dashed line is needed only when the enterprise conditions are (PBX firmware is) modified, respectively. The dotted arrow represents the optional manual formulation of the expert knowledge about the action representation in the logging data. The star-labelled processes are those where learning/induction takes place. The digit in the upper-left corners in boxes refers the section where more detail on the respective process/database is given.

In other words, we have performed a *data-mining* task on the input data and tried to integrate the results into a *decision support* system. The methods of data-mining have been currently receiving a lot of attention [6], especially those allowing for intelligent mining from multiple-relation databases [9]. By employing the techniques of inductive logic programming, we are in fact conducting a multi-relational data-mining task. Although there is previous work on data-mining in telecommunications [8], we are not aware of another published approach utilizing multi-relational data-mining methods in this field. The integration of data-mining and decision-support systems is currently also an opening and discussed topic [1], and research projects are being initiated in the scientific community to lay out a conceptual framework for such integration. We hope to have contributed to that research by this application-oriented paper.

8 Acknowledgements

This work has been supported by the project MSM 212300013 (Decision Making and Control in Manufacturing, Research programme funded by the Czech Ministry of Education, 1999-2003), the Czech Technical University internal grant No. 3 021 087 333 and the Czech Ministry of Education grant FRVS 23 21036 333.

References

1. Workshop papers. In *Integrating Aspects of Data Mining, Decision Support and Meta-Learning*. Freiburg, Germany,, 2001.
2. Petr Aubrecht. Sumatra Basics. Technical report GL–121/00 1, Czech Technical University, Department of Cybernetics, Technická 2, 166 27 Prague 6, December 2000.
3. Ivan Bratko. *Prolog: Programming for Artifical Intelligence*. Computing Series. Addison-Wesley Publishing Company, 1993. ISBN 0-201-41606-9.
4. L. Dehaspe and L. De Raedt. DLAB: A declarative language bias formalism. In *Proceedings of the 10th International Symposium on Methodologies for Intelligent Systems*, volume 1079 of *Lecture Notes in Artificial Intelligence*, pages 613–622. Springer-Verlag, 1996.
5. P.A. Flach. *Simply Logical: Intelligent Reasoning by Example*. John Wiley, 1994.
6. D. Hand, H. Mannila, and Smyth P. *Principles of Data Mining*. MIT, 2000.
7. N. Lavrač and S. Džeroski. *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood, 1994.
8. R. Mattison. *Data Warehousing and Data Mining for Telecommunications*. Artech House, 1997.
9. N. Lavrac S. Dzeroski, editor. *Relational Data Mining*. Springer-Verlag, Berlin, September 2001.
10. F. Železný, P. Mikšovský, O. Štěpánková, and J. Zídek. ILP for automated telephony. In J. Cussens and A. Frisch, editors, *Proceedings of the Work-in-Progress Track at the 10th International Conference on Inductive Logic Programming*, pages 276–286, 2000.