Modeling and understanding time-evolving scenarios

Riccardo Melen, Fabio Sartori and Luca Grazioli
Department of Informatics, Systems and Communication - University of Milano-Bicocca
Viale Sarca 336/14, Milan, 20126, Italy

ABSTRACT

In this paper, we consider the problem of modeling application scenarios characterized by variability over time and involving heterogeneous kinds of knowledge. The evolution of distributed technologies creates new and challenging possibilities of integrating different kinds of problem solving methods, obtaining many benefits from the user point of view. In particular, we propose here a multilayer modeling system and adopt the Knowledge Artifact concept to tie together statistical and Artificial Intelligence rule-based methods to tackle problems in ubiquitous and distributed scenarios.

Keywords: Bayesian Network, Rule-Based Systems, Time-Evolving Scenarios, Knowledge Artifacts, Shadow Facts.

1. INTRODUCTION

Rule-based systems are the technology of choice for solving a wide variety of problems involving the understanding of complex phenomena and the planning of the consequent actions.
A generic rule-based system is made of an inference engine, a knowledge base made of rules and a set of facts to be analyzed. The set of rules embodies the knowledge available about the particular scenario we want to model; therefore "understanding" means the interpretation of a pattern of events/facts obtained by matching the left part of some rules contained in the knowledge base and deriving the appropriate inferences.
In many cases the set of applicable rules is static, i.e. it does not change in time. The applications of this simple, well-established form of rule-based system are many: from medical diagnosis to network fault management, from environment monitoring to security risk analysis, we have thousands of commercial applications of such kind of technology.
In this paper we are concerned with a more complex problem, that of modeling time-varying scenarios.
In this case, the observed system and its reference environment change in time, passing through a series of macroscopic states, each one characterized by a specific set of relevant rules. Moving from one state to another, the meaning and importance of some events can change drastically, therefore the applicable inferences, as described by the rule set, must change accordingly.
The crucial point from the system point of view is the difficulty for production rules to capture in a precise way the knowledge involved in decision making processes which are variable in an unpredictable way. The resulting rules set must be obtained as the product of an intensive knowledge engineering activity, being able to generate new portions of the knowledge base effectively and efficiently with respect to the changes in the application domain.
Some examples of these application scenarios can help in clarifying the characteristics of the problems we intend to tackle.
A first example is the evolution of the state of an elderly patient affected by a neurologic degenerative disease. Quite often the development of the disease does not proceed in a linear, predictable way; instead long periods of stationary conditions are followed by rapid changes, which lead to another, worse, long lasting state. In this case, the interpretation of some events (such as a fall, or a change in the normal order in which some routine actions are taken) can differ substantially depending on the macro-state of reference.
Another case would be an application analyzing urban traffic, with the purpose to help a driver to take the best route to destination. The scenario being analyzed changes significantly with the hour of the day and the day of the week, as well as in response to events modifying the available routes, such as an accident or a street closure due to traffic works.
In these situations, an efficient response of the system is very important, since the elaboration must be necessarily "real-time", and it is mandatory for the system to check continuously the knowledge-base to understand if it is consistent or not. In this paper we present an approach to the development of rule-based systems which change their behavior dynamically according to the change in number and value of the problem variables. The approach is based on the notion of Knowledge Artifact (KA), a conceptual and computational tool for the acquisition and representation of heterogeneous knowledge involved in complex domains.
For the sake of simplicity, in the rest of the paper we shall use the term "state" when referring to the macroscopic states described above, and the expression "evolving scenario" to indicate the situation where a system and its reference environment evolve across a sequence of states, that is the case of interest for our study.

2. RELATED WORK

Our concept of state of an evolving scenario has some similarities with the *situations* studied in [1]. The main differences are that our scenarios develop on a longer time scale, one state can turn gradually into another one and its characterization may include components which are not immediately measurable, such as the mental conditions of an individual. However we employ some of the techniques described in [1].
The use of ontologies in a layered modeling approach has been discussed in [2]; in that paper, however, ontologies are employed to reason about contexts in a deterministic way, without modeling uncertainties and transitions across contexts.
One of the cornerstones of our work is the concept of Knowledge Artifact.

In Computer Science, artifacts have been widely used in many fields like Distributed Cognition [3], CSCW [4] and MAS paradigm [5].
According to those definitions, artifacts are typically considered *passive entities* in literature: they can support or influence human and artificial agents reasoning, but they are not part of it, i.e. they don't specify how a product can be realized or a result can be achieved. In the Knowledge Management research field, Knowledge Artifacts are specializations of artifacts. According to Holsapple and Joshi [6], *``A knowledge artifact is an object that conveys or holds usable representations of knowledge."* Salazar-Torres et al. [7] argued that, according to this definition, KAs are artifacts which represent *``[...] executable-encodings of knowledge, which can be suitably embodied as computer programs, written in programming languages such as C, Java, or declarative modeling languages such as XML, OWL or SQL".*

Thus, Knowledge Management provides artifacts with the capability to become *active entities*, through the possibility to describe entire decision making processes, or parts of them. In this sense, Knowledge Artifacts can be meant as guides to the development of complete knowledge-based systems.

3. OUR APPROACH – PART I: THE PROBLEM REQUIREMENTS

A direct solution to the problem of building a knowledge base coping with an evolving scenario consists in defining one or more state variables, whose values describe the present state, and putting a check on the state in the left parts of all the rules. In this way the knowledge base is partitioned into disjoint subsets, each one valid for a specific state, and the system tracks the scenario evolution by asserting the state variables.
Such an approach leads to an unwieldy number of rules and to the risk of building an ambiguous/inconsistent knowledge base. Even more serious, however, is the problem of representing gradual changes: a set of "hard coded" descriptions of the possible states is not sufficient to model the transition period between contiguous states in the scenario evolution: as a matter of fact, it is assumed that the analyzed scenario jumps abruptly from one state to another, and this transition is reflected in the model by the firing of the rules which assert the new value(s) of the state variable(s).
In some applications this model is perfectly adequate to reflect the reality. Consider for instance the application analyzing urban traffic mentioned above: its reference environment changes almost immediately when a street is closed due to traffic works, and a sharp state transition is perfectly justified.
In many other cases, however, we need to track a more gradual evolution. An example is the evolution of neurologic degenerative diseases mentioned in the introduction: the transition from a given state of cognitive impairment to a worse condition may follow a non-linear pattern, where the patient switches back and forth between two states for some time, or shows initially the symptoms of the worse condition only with respect to some specific tasks or abilities.
Another case is the change in traffic patterns with the hour of the day: here we have a different kind of transition, because the traffic flows typical of the morning rush hour transform smoothly into the flows typical of the late morning, giving rise to a sequence of intermediate states.
All these examples refer to cases where the various possible states are known and can be modeled in advance, either by defining heuristically a set of rules or by some automatic knowledge base construction technique.
In other situations, however, only the present state is embodied in the knowledge base as a set of rules, while we do not have a precise formalization of the new states where the scenario could evolve. This may be due either to a lack of knowledge about the characteristics of the scenario which is being considered, or to its intrinsic nature: as a matter of fact in some cases it is not possible to identify a set of distinct macroscopic states, because the scenario evolves across a continuum. Examples of this kind arise, for instance, in marketing studies, when we want to follow the evolution of the preferences of a large number of potential customers.
In these cases, if we want to maintain the approach of using a rule-based system, we would need a solution capable of assessing the adequacy of the present set of rules, and modifying it by generating new rules dynamically, while the scenario evolves. However other technical solutions, for instance based on statistical decision making, are possible and may be preferable. Although this class of problems is outside the scope of our research, we will note when the techniques we have developed can be applied also to it.
To summarize the discussion, we attempt to classify the evolving scenarios into some categories, and to select the appropriate technical approach for each one.
Figure 1 depicts the various kinds of evolving scenarios we have discussed up to this point, which we will call as follows: sharp transition ("street closed"), morphing (from rush hour to mid-morning traffic), bouncing (for instance the evolution of the Alzheimer disease) and continuum (customer preferences).

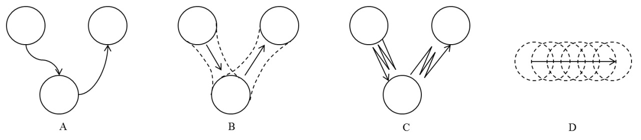

Figure 1: several kinds of evolving environments

Note that this classification is somewhat simplified: for instance the traffic understanding application that we have hypothesized must face a scenario which exhibits both sharp transitions and morphing.

4. OUR APPROACH – PART II: THE LAYERED MODEL

The basic principle on which our approach rests is a two-layered approach to scenario modeling. The present state of the scenario is represented by the knowledge base of a standard rule-based system, while the set of all possible states is modeled by a higher level abstraction, namely by an ontology.
With this approach we can distinguish clearly between the active set of rules which applies to the ongoing flow of events, providing a semantic interpretation of the current state and determining the reactive actions to be taken, and an (implicit) representation of all the possible rules which make sense, given a coherent view of the objects and relations which are admissible within the scenario evolutions.
While the separation of two different modeling layers is a quite natural way to deal with evolving environments such as those represented in Figure 1, it is important to stress that the interaction of the two layers can follow different mechanisms. The upper layer can be considered just as an offline tool, producing several "fixed" scenario representations, which are fed into the rule-based system when some specific event occurs. In our approach, instead, both levels cooperate in real time: the upper layer has the task of maintaining and updating

the “running” knowledge base employed by the lower layer. This can also be done gradually, in order to track morphing transitions such as the one in Figure 1b.
In the next section we will describe how the upper layer evaluates the adequacy of the current knowledge base and how the updating can be performed in the various transition scenarios. Let us conclude the present discussion with a review of the motivations justifying our layered approach.
A first kind of motivations regards two modeling issues: timing and handling of unexpected events.
The two layers respond to the modifications of the scenario according to two different time scales: the lower layer responds to single events occurring on a short time scale, while the upper layer tracks the long term evolutions, and typically responds to longer sequences of events, bearing some statistical significance. This separation allows dealing properly with the timing requirements, trading off precision with timeliness in the upper layer transitions and introducing, if necessary, real time capabilities at the lower layer (we will elaborate on this point later on).
The rule-based system implementing the lower layer of the model is designed to work, in line of principle, in an open world (this expression should be interpreted here in a sense close to [8] and is quite different from the Open World Assumption in formal logic): this means that events that do not cause any rule to fire are simply discarded after a while, without any modification to the subsequent operations at the lower layer. On the other side, the upper layer assumes a more complete view of the world, and considers all the possible events, i.e. all the events allowed by the underlying ontology: for instance, the occurrence of very unlikely sequences of events can be seen as an indication of the lack of adequacy of the running knowledge base.
Another kind of motivations is related to knowledge engineering issues. As a first observation, providing a high level of abstraction in the form of an ontology allows a human expert to perform a simpler verification of its correctness with respect to the specific domain of application. A second advantage regards the formal consistency of the running knowledge base, which is easily verifiable at the runtime.
Finally we have the motivations related to implementation and performance. We are especially interested in distributed deployments of our architecture. For instance, in the case of the monitoring of elderly patients, it would be useful to implement the lower layer on portable wireless devices, such as tablets and smartphones, in order to provide a reaction to events which is both faster and more reliable (there are no risks and delays associated with temporary losses of connectivity); however the upper layer processing, which can be more computationally expensive and has less real time constraints, is more suited to a centralized implementation.
Furthermore, the implementation of the lower layer as a separate entity can be optimized in various ways: as an example, time-consuming verifications of facts (e.g. measurements of a physical quantity) can be postponed until all the other left-side conditions of a rule are verified so that the rule could fire: this optimization is supported for instance by the “shadow facts” construct of Jess [9].
Figure 2 summarizes the observations we made about the layered modeling, and sketches a possible supporting architecture: note that we introduced specific functions which monitor the adequacy of the running knowledge base and manage the necessary updates. The precise nature of these functions will be described in the next paragraph.

5. OUR APPROACH PART III – TRACKING A CHANGING SCENARIO

A Knowledge Artifact for Evolving Scenarios

The most specific characteristics of our approach regard the methods we use for tracking the evolving scenarios.

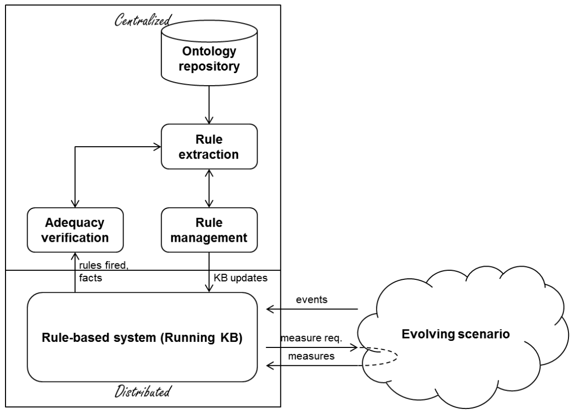

Figure 2: implementation of a layered model

There are three main elements to be described: adequacy verification, new rule production and knowledge base management. The crucial aspect of them is that they must be correlated in order to capture the scenario variability without the risk of being inconsistent. To this aim, from the conceptual point of view, we adopted the notion of Knowledge Artifact (KA).
More specifically, in our framework (see Figure 3) the KA is made of three main components:

- an ontology-based description of the possible entities and of their possible relations in the considered scenario;
- a Bayesian Network, employed to select the causal relations which are applicable in the present state of scenario evolution;
- Production Rules, embodying the knowledge necessary to implement the Knowledge Base rules.

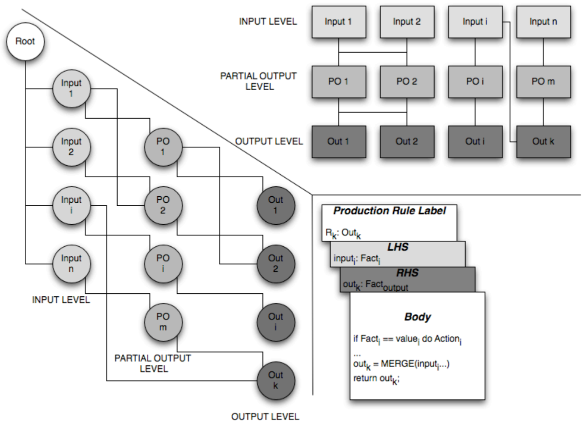

Figure 3: graphical representation of KA elements. Relationships among them are shown by means of gray scale coloring

The role of ontology

With respect to the scenario in Figure 2, ontology is responsible for identifying the system inputs, outputs and

partial outputs, as well as the relationship among them. In other words, the ontology describes the structure of the application domain and identifies which are the important elements to consider for solving understanding problems.
The ontology role is to check that the domain representation in terms of inputs, outputs and partial outputs is coherent: if new elements must be added to the scenario, they are added at the ontological layer. Moreover, the ontology allows deleting or modifying inconsistent nodes or relationships, according to the temporal evolution of the system.

Rule extraction and Knowledge Base management
A Bayesian Network is used for implementing the *rule extraction* functional block in Figure 2: given an output or partial output from the ontology, the related BN describes the causal flow from inputs to outputs, moving through partial outputs.
BN allows checking the state of the scenario from the procedural point of view: if some modifications happen at the ontological layer, the BN tries to forecast the consequences in terms of new behaviors, i.e. new causal relationships to add. Moreover, it is responsible for the verification of the correct behavior of the system from the statistical point of view: it is able to generate new sets of most probable rules to extend the global system behavior according to the variations occurred at the ontological level.
Finally, Production Rules allow defining the causal flow of a given BN in terms of rule-based constructs. A rule is made of a Left Hand Side (LHS) that is a logical clause involving one or more facts from the knowledge base and a Right Hand Side (RHS), which specifies actions to do in case the LHS is true. These actions could be modifications of the knowledge base, like insertion/deletion of new/obsolete facts or I/O operation, to get/return input/output values from/to the user.
A collection of rules is produced for each output of the system, while partial outputs (i.e. results of a computation useful to obtain an output, but not interesting for the user) are managed in the same way: of course, partial output must be executed before the outputs that they influence, following the causal relationships introduced by the BN. In this way, the correct division of the system into computational layers is defined.
The *rule management* functional block in Figure 2, handles the insertion of the new rules in the running system and the elimination of obsolete rules (those suited to a past state of the environment, possibly conflicting with the new ones). In a non-automatic implementation of the rule management a further task which could be carried out is the modification of the set of rules, performed by a human expert.

Adequacy verification
Dealing with time-evolving scenarios means that the set of rules employed by the KBS (the running KB) may become inadequate to our purpose due to a transition.
In the most straightforward solution to this problem, the updating function can be accomplished by keeping the BN working on the stream of events/facts which are being fed into the KBS. In this way the BN is continuously re-computing the set of most probable rules, and the rule management functional block compares this set with the one being employed by the KBS, and performs the necessary updates. Figure 4 depicts this approach, taken in our case study (see below).

The outlined procedure has only one critical parameter, which is the duration of the time window containing the events/facts which are taken in account for the computations of the BN. A large time windows causes a slow reaction by the system, but the updating process cannot be led astray by unlikely and isolated combinations of events, not indicating a state transition.

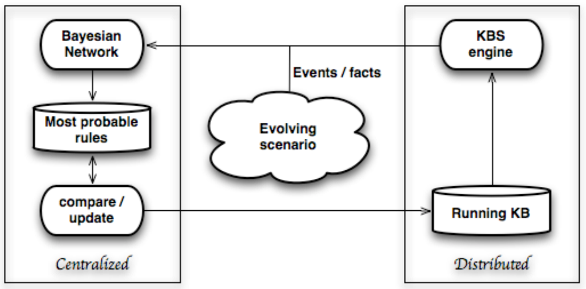

Figure 4: a straightforward solution to the adequacy problem

Conversely, a narrow time window is less significant from a statistical point of view, but allows a faster system reaction.
The correct duration of the time window can be established with the help of a domain expert; quite obviously, shorter time windows are suited to environments which exhibit sharp transitions, larger windows are suited to morphing and continuum transitions, while bouncing ones require a more sophisticated approach (for instance handling different groups of rules with different time windows).
Although the approach we have just discussed is perfectly reasonable in several practical applications, it is ill-suited to distributed implementations and real-time, data intensive problems, because it requires a large quantity of data being continuously sent from the distributed to the centralized level.
An alternative approach consists in separating the task to realize that a transition is occurring (or is likely to occur) from the computation of the new set of rules. In this way the transition detection is performed by a dedicated functional block (*adequacy verification* in Figure 2), offloading the BN from a task which is not appropriate to it. The BN can then intervene for the computation of a more adequate set of rules, requiring the direct access to the monitored events for a much shorter time span. Moreover, the process of rule extraction can take advantage of the preliminary indications of the adequacy verification function.
The crucial point in this approach is the implementation of the verification function. A possibility we are investigating is defining a very small set of rules (called monitoring rules), executed on a separate system, which can be considered reliable and timely indicators of an occurring transition. We have been able to identify monitoring rules for some practical problems, but the implementation of this alternative approach to adequacy verification requires some further work.

Real Time behavior
As stated above, one of the advantages of a layered approach is performance. More specifically, we want to obtain an architecture which can react in real-time to the modifications of the environment: technically, it must be possible to enforce predefined deadlines for the firing of a rule after the occurrence of relevant events.
Of course, this requirement is significant for the KBS while an analogous requirement imposed to other parts of our system, and in particular to the BN, would not make sense in practical situations because the timing of the BN operations is constrained by the need to collect statistically significant data.
The technology we employ for the implementation of the KBS,

based on the Jess platform, provides a very effective features for the support of real time requirements, the *shadow facts*.
Shadow facts are *Java Beans* objects treated as facts in the knowledge-base. If they change their value, the inference engine is able to detect this change and execute again the KBS, being sure that no inconsistency between outputs and inputs will be generated.
This implies that the presence in the left side of a rule of a shadow fact whose code embodies an update period of, say, 100 milliseconds, guarantees that such rule will be ready to fire every 100 ms, possibly using new information about the environment gathered by the shadow fact. This feature therefore allows implementing real-time requirements directly into the KB rules.

6. IMPLEMENTATION

This section describes our implementation of the mechanism to generate, in real time, new rules according to the new observations. The model is implemented in JAVA and includes four main components responsible for the generation of new rules:

- **Monitoring agent (MA)**: a Thread whose task is to understand when a rule update is necessary. When this condition is expressed, it starts the updating procedure.
- **Expert system manager (ESM)**: it is the component responsible for the rule-based system's management. It communicates with a JESS engine and with the manager of the "upper level".
- **Bayesian Network manager (BNM)**: the Bayesian Network executor. It manages the net structure and its update.
- **JESS engine (JE):** the manager of the JESS components.

The MA component is the place where the adequacy verification algorithm can be implemented (see Figure 2). Moreover, it is responsible for the management of the domain ontology, managing the set of variables necessary to describe the problem. This set can vary over time, both in the number of variables involved and in their values, causing a transition from a state S_i to a new state S_j. When such a transition occurs, MA starts the rule updating process.
The updating procedure is accomplished through an asynchronous call to ESM: after this call ESM needs to have the list of posterior probabilities updated to extract new rules; for this reason it launches a synchronous call to the BNM waiting for the complete list of posterior probabilities for each node of the net. BNM computes an inference procedure for each possibility, given the evidences of the system. That is the heaviest operation from the computational point of view.
Once the new posterior probabilities are processed, ESM is able to extract new rules. Only at this moment it interrupts JE, storing the new rules in the knowledge base.
From the computational point of view, the three-tuple (MA, BNM, ESM) constitutes the KA implementation in our case study. According to the description above, the three components are related one another, as introduced in Section 4.
The adoption of shadow facts in the communication between MA and the Jess Engine allows capturing the real time behavior of the framework: the state transition from S_i to S_j causes the variation of a shadow fact object value or the instantiation/deletion of a new/existing shadow fact object.

7. CASE STUDY AND EXPERIMENTAL EVALUATION

The case study we present in this section is related to the analysis of urban traffic. Nowadays, the need of having recommendations about mobility in the urban context is greater than ever, due to the ever-growing metropolitan areas, with higher population density. To satisfy this need, the diffusion of personal wireless devices, such as smartphones, allows monitoring of different variables, like the traffic conditions, itineraries calculus with distance and timing, and so on.
Our application scenario uses a set of information that ranges from physical and psychological condition of the user to weather condition, day of the week, traffic condition, and so on. Each observation is collected from personal devices, like smartphones and wearable devices.

Using these different kinds of information, we realized a simulator composed by different kinds of agents, whose goal is to reach the upper town of Bergamo from the lower town using one of the three possibilities available (i.e. *bus*, *funicular* and *stairs*).
The goal of the case study is to compare a new version of the simulator, which makes use of the model introduced in this paper with a previous version, where a "classical" Bayesian inference procedure was adopted for taking the decision.
Each agent is equipped with a decision engine, realized according to the model described so far, in particular the adopted BN is depicted in **Figure 5**.

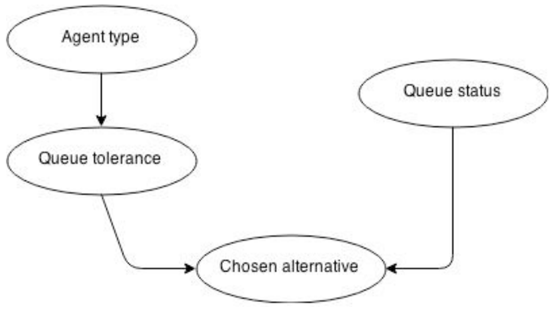

Figure 5: the case study BN nodes and relations

This structure shows a high level representation of the causal model and it can be seen as the responsible for the generation of rules used by the "running" knowledge base (the lower level of the model). Every time an agent is created, the system provides it with the decision engine described earlier in this section (essentially a rule-based system exhaustive of all the possible situations expected from the scenario). There are two inputs used by each agent: the agent's type is retrieved by the attributes of the agent itself, and the queue status is provided by the environment. The decision engine is now ready to be executed given the inputs provided.
Please note that the rule-based systems produced are always different from each other, generating, as a consequence, heterogeneous agents.
Only the chosen alternative is used by the agent as output of the decision engine, and it represents his choice in the environment.
The details of the simulator are beyond the scope of this paper, however it is important to know that every agent is introduced in a controlled environment, and it has to take a decision about which transport to take
The simulator creates three different types of agents: *citizen agent* (this type of agent is characterized by a deep knowledge of the territory and by a low tolerance to queue), *nearby agents* (this second type of agent is characterized by a moderate knowledge of the territory and a rather high tolerance to queue) and *tourist agents* (with no knowledge of the territory and a

high tolerance to queue). Assuming the nodes “Agent type” and “Queue status” as evidences, the BNM, as a first operation, generates all the possible LHS configurations: in the example there are 12 possible alternatives for each output node. After this preliminary computation the BNM generates a posterior distribution of probability for each configuration; for instance, one possible posterior distribution of probability obtained after the inference procedure is the following one:

```
Node: Vehicle
Configuration exploited:
...... Node: Queue Outcome: none
...... Node: Agent_Type Outcome: citizen
Posterior probability distribution:
--------- Outcome: funicolar Probability:
0.7275
--------- Outcome: bus Probability: 0.2515
--------- Outcome: stairs Probability:
0.021000000000000005
```

Once the 24 probability distributions have been obtained, the ESM uses one of the three alternatives modeled to extract new rules, and then stores them in the JE; for instance the rule extracted by the previous distribution is the following, where LHS and RHS are separated by the => symbol:

```
(defrule VEHICLEfunicolar13 (Queue none)
(Agent_Type citizen)
=>
(assert (Result_Vehicle (Vehicle
funicolar) (Reliability 0.7275))))
```

This is the complete sequence used for asserting the new rule in the knowledge base. Note that the RHS of the rule reports also a reliability value, retrieved from the posterior distribution of probability. The reliability of the rule is the value used by the model to understand when something is changing around the user: if the same rule, in a further updating procedure, obtains a different reliability value, the model realizes that something in the reliability of the rule is changed (for instance, if the value has grown, the given rule is more reliable). The new rule is now ready to be executed. What we observed, after various executions of the simulator, is quite interesting: compared to the “classical” decision mechanism, this new version provides different behaviors over the time.
The original simulator version did not show a plausible behavior in all cases. Examining the queue composition of the most critical transport vehicle (the funicular), we noticed that the behaviors of citizens agents and nearby agents was often too similar. This conduct wasn’t expected. Initially we thought that the reason of this unexpected behavior was the BN’s structure itself: a more complex BN, capable of modeling more environment variables, would have been able to discern in a better way the agents’ behaviors.
This conclusion is denied by the new version of the simulator. Analyzing the new funicular queue composition plot (see **Figure 6**) it’s clear how the agents’ behaviors are now very separated and more adherent to reality. The reason of this improvement is in the decision model itself: the rule execution, instead of a simple Bayesian inference whose result is not predictable, generates more characteristic agents, or rather better distinguishable between each other.

8. CONCLUSIONS AND FUTURE WORK

In this paper, we have discussed the problem of modeling time evolving scenarios. This is a very important research trend in Computer Science, involving heterogeneous competencies. Indeed, the continuous evolution of mobile devices and

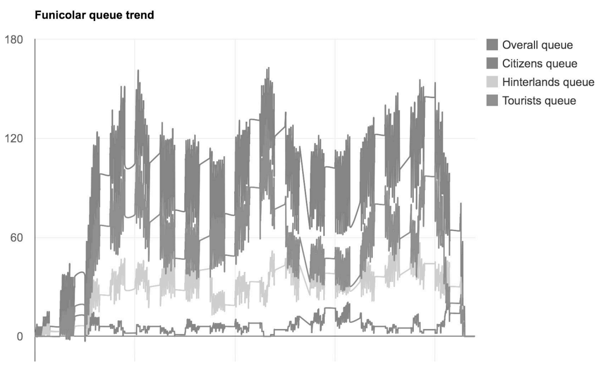

Figure 6: a line chart representing the funicular queue composition overtime.

applications offers new and stimulating challenges as well as opens to new possibilities to tackle it, allowing integrating easily and profitably different kinds of systems into unique conceptual and computational frameworks, such as the Knowledge Artifact concept in our approach.
This is the motivation of our work, to understand how statistical and Artificial Intelligence methods like Bayesian Networks and rule-based systems can be exploited to automatically generate and use new knowledge when necessary.
Future works will be mainly devoted to testing the applicability of the framework to develop systems characterized by variable conditions and parameters.

9. REFERENCES

[1] Ye, J., Dobson, S., & McKeever, S. (2012). **Situation identification techniques in pervasive computing: A review.** Pervasive and mobile computing, 8(1), 36-66.
[2] Gu, T., Pung, H. K., & Zhang, D. Q. (2005). **A service-oriented middleware for building context-aware services.** Journal of Network and computer applications, 28(1), 1-18.
[3] Norman, D. A. 1991. **Cognitive Artifacts**. Design Interaction, pp. 17-38
[4] Shmidt, K. and Simone, C. 2000. “Mind the gap”, Towards a unified view of CSCW. COOP, pp 205-221.
[5] Omicini, A., Ricci, A., Viroli, M. 2008. **Artifacts in the A&A meta-model for multi-agent systems**. Auton Agent Multi-Agent Syst, 17:432-456.
[6] Holsapple, C., and Joshi, K. 2001. **Organizational Knowledge Resources**. Decision Support Systems, vol.1, pp. 39-54.
[7] Salazar-Torres, G., E. Colombo, F. S. C. da Silva, C. A. Noriega, and S. Bandini. 2008. **Design issues for knowledge artifacts**. Knowledge-Based Syst., vol. 21, no. 8, pp. 856–867.
[8] Baresi, L., Di Nitto, E., & Ghezzi, C. (2006). **Toward open-world software: Issue and challenges**. IEEE Computer, 39(10), 36-43.
[9] Friedman-Hill, E. (2003). **JESS in Action.** Greenwich, CT: Manning.
[10] Sartori, F., Manenti, L., & Grazioli, L. (2013). **A Conceptual and Computational Model for Knowledge-based Agents in ANDROID.** WOA@ AI* IA, 2013, 41-46.