Supervised Learning of an Abstract Context Model for an Intelligent Environment

Oliver Brdiczka, Patrick Reignier & James L. Crowley

GRAVIR/IMAG, 655 Av. de l'Europe - 38330 Montbonnot-St. Martin, France
{brdiczka,reignier,crowley}@inrialpes.fr

Abstract

This paper addresses the problem of supervised learning in intelligent environments. An intelligent environment perceives user activity and offers a number of services according to the perceived information about the user. An abstract context model in the form of a situation network is used to represent the intelligent environment, its occupants and their activities. The context model consists of situations, roles played by entities and relations between these entities. The objective is to adapt the system services, which are associated to the situations of the model, to the changing needs of the user. For this, a supervisor gives feedback by correcting system services that are found to be inappropriate to user needs. The situation network can be developed by exchanging the system service-situation association, by splitting the situation, or by learning new roles. The situation split is interpreted as a replacement of the former situation by sub-situations whose number and characteristics are determined using conceptual or decision tree algorithms. Different algorithms have been tested on a context model within the SmartOffice environment of the PRIMA research group. The decision tree algorithm (ID3) has been found to give the best results.

1. Introduction

An environment is called "perceptive" if it is capable of maintaining a model of its occupants and their activities. An environment becomes "active" when it is capable of (re)actions (system services). An "interactive" environment is based on the capacity of perception, (re)action and communication with the users. "Intelligent" environments should provide services while minimizing disruptions, such as explicit man-machine communication. This requires that the intelligent environment perceives user activities and identifies user needs correctly in order to react in an appropriate way. However, user needs and system services evolve in the course of time. Further, different users may have different needs for the same activities. Thus an intelligent environment must be capable of adapting and developing its services automatically to meet the specific needs of a user.

In this paper, we describe a method for evolving and developing the abstract context model for an intelligent environment. Our abstract context model is based on a situation network. The situations of this network are altered and split in order to meet changing user needs. These evolutions of the model are intended to maximize the correctness of the executed system services concerning the user needs perceived by a supervisor.

2. Problem Statement

The problem addressed in this paper is machine learning in an intelligent environment. The intelligent environment is a computer system that executes a number of services according to perceptual information on user actions or activity. As we know, user behavior changes in the course of time. The automatic adaptation of system (re)actions according to changing user needs is seen as machine learning process. We need information in the form of feedback on executed system services in order to guide the learning process. Further, we can define several qualities that this machine learning process should have:

- *Understandable Representation and Reasoning:* We consider that a user is only willing to accept an intelligent environment offering services implicitly if he understands and foresees its decisions. Thus we want the user to be able to understand the system perceptions and their representation in the model of the interactive environment. Further, the learning process, i.e. the reasoning and the development of the model necessary to cover changing user needs, should also be understandable for the user.
- *Supervisor corrections (feedback):* We want to minimize the frequency with which the system offers inappropriate services, while minimizing disruption. This means that the feedback given to the system is to be minimal to achieve the wanted changes of system services. We assume that a person, denoted supervisor in the following, is capable of specifying system services to be executed by the system and that his feedback is always consistent. The user himself or another person can act as this supervisor. In this paper, we distinguish three forms of supervisor feedback:
  - (Re)action correction: the service or (re)action executed by the system is wrong and a different service must be executed instead. The supervisor gives the different system (re)action as feedback to the system. This includes the case where the supervisor wants the system to execute a (re)action while the system does not execute anything.
  - (Re)action deletion: the service executed by the system is wrong and no system service must be executed instead. The supervisor gives a particular (re)action, the "erase" (re)action, as feedback to the system.
  - (Re)action preservation: the service executed by the system is correct. The supervisor does not give any information to the system. As we assume that the supervisor is always consistent, we can interpret the absence of his corrections or deletions as positive feedback for the currently executed system (re)actions.

The supervisor will not give any further feedback on the model representing the intelligent environment. The learning process must integrate the information given on the system (re)actions by evolving and developing the model.

3. Previous Approaches and Related Work

An early example of an interactive environment is the KidsRoom [1], a perceptually-based, interactive, narrative playspace for children. The KidsRoom environment interacts with the children in order to narrate a story. It does not have, however, any learning capacity concerning the automatic development of its (re)actions. The perception modules and the (re)actions in the environment are integrated in a preprogrammed story context and need to be adapted by hand (in general by a programmer).

The ContAct project [2] realized a system deriving user activity and availability from sensor data in an office environment. The system uses a naive Bayesian Classifier to learn user activity and availability directly from sensor data according to given user feedback. This system is, however, only perceptive and not (inter)active as there are no (re)actions defined or planned by the system. Further, no understandable representation of the environment is provided.

The Lumiere project [3] at Microsoft Research realized methods and an architecture for reasoning about the goals and needs of software users as they work with software. The objective is to learn appropriate interactions with the user according to perceived user activities in order to offer assistance within a software environment. At the heart of Lumiere are Bayesian models, which are based on a large amount of example data from users and experts. Although these models could identify a number of (understandable) pertinent variables of human behavior (like task history or assistance history), their reasoning is not obvious to the user.

In this paper, we want to focus on an understandable model of the environment whose initial construction can be done by hand (without large amounts of example data). The context model [4] is a non-Bayesian model inspired by concepts of planning and knowledge representation used in robotics and artificial intelligence. The central concept of this model is the notion of situation. A situation refers to a particular state of the environment. It is composed of a particular configuration of entities, roles and relations (see Fig. 1 for an example).

An entity may generally be understood as corresponding to a physical object or person. It is created accompanied by a set of numerical or symbolic properties. We say that an entity is observed to play a role if it passes a role acceptance test on its properties. This role acceptance test may be seen as a predicate function defined over entities and their properties. A person may, for example, play the role lecturer if he or she stands next to the presentation screen, which is tested by comparing the person's position to predefined values (acceptance test).

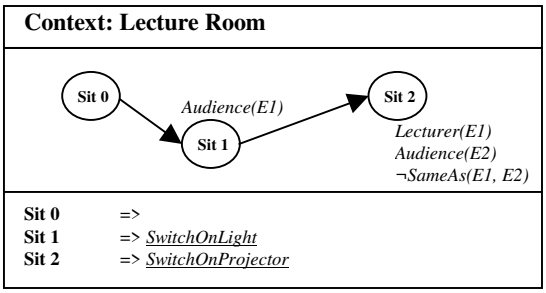

Figure 1: Example of a context model for a lecture room. **Sit 0**, **Sit 1** and **Sit 2** are the available situations. *Lecturer*, *Audience* are the available roles and *SameAs* the available relation. SwitchOnLight, SwitchOnProjector are system (re)actions. E1 and E2 are entities.

A relation is defined as a predicate function on several entities playing roles. The identity relation may, for example, be created by comparing the names of two entities playing different roles. A context is determined by the available roles and relations. A situation stands for a particular assignment of entities to roles completed by a set of relations between these entities. Thus a change in the relations between entities or in the assignment of entities to roles results in a change of situation. This (possible) change of situation is represented by an arc connecting these situations. The context can then be represented by a network of situations. System (re)actions (system services) are directly associated to the situations in the network.

4. Method

The understandable form of the context model allows an easy predefinition of situations and associated system (re)actions by a supervisor. The learning process must adapt the predefined context model according to the given supervisor feedback on the system (re)actions. As the different layers of a context model (entities - roles, relations - situations) influence each other, the learning process cannot adapt them simultaneously. Thus we will focus on the development of the situation network and the associated system (re)actions.

Bayesian models (in particular Hidden Markov Models [5]) as well as algorithms based on first-order logic like [6] have been considered to represent and adapt the situation network. However, these approaches do not have good properties concerning the extension of the number of situations, which is essential for developing a situation network. Bayesian models require a large amount of example data to extend the number of states. First-order logic algorithms cannot create new predicates (problem of higher order logic), which corresponds to the extension of the number of situations. Thus the method proposed in this paper is based on algorithmic changes of the structure of the situation network.

4.1. Algorithm

Fig. 2 shows an overview of the proposed algorithm. The input of the algorithm is a predefined situation network (context model) and feedback given by a supervisor. The supervisor corrects, deletes or preserves the (re)actions executed by the system while observing a user in the environment. Each correction, deletion, or preservation generates a training example for the learning algorithm containing current situation, roles and relations configuration, and the (correct) (re)action. The differences between the (re)actions given in the training examples and the (re)actions provided in the predefined situation network will drive the different steps of the algorithm.

In the first step, the algorithm tries to directly modify its (re)actions using the existing situation network. If (re)action A is associated to situation S, and all training examples indicate that (re)action B must be executed instead of A, then B is associated to S and the association between A and S is deleted.

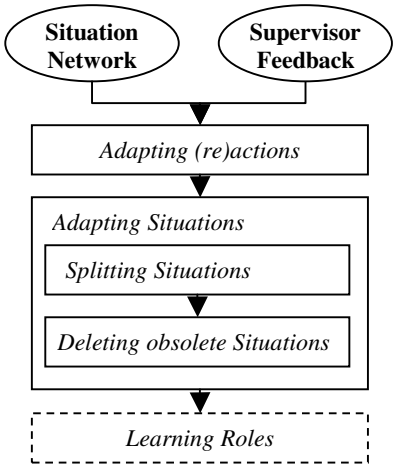

Figure 2: Overview of the different steps of the algorithm.

In the second step, the algorithm tries to modify the situation network. The situation split is executed when the supervisor perceives several situations (expressed by different (re)actions in the training examples) while the predefined situation network only perceives one situation (expressed by one (re)action). Thus the situation perceived by the predefined situation network may be too general and the algorithm tries to split it in sub-situations. The role, relations configuration of these sub-situations needs to be determined according to the given training examples (Section 4.2). After the situation split, sub-situations whose associated (re)action is the "erase" (re)action are deleted.

4.2. Splitting Situations

When splitting situations, a number of training examples indicate different (re)actions for one situation of the predefined situation network. Several sub-situations need to be created for these (re)actions. We must determine the characteristic role, relation configurations of these sub-situations (Fig. 3).

As a context is defined by a finite number of available roles and relations, the situations within this context can be represented as a fixed-sized vector containing one 0/1 value for each available role and several 0/1 values for each available relation. The value 1 means that the corresponding role or relation is valid; the value 0 means that the role or relation is not valid. As a relation is applied on entities playing roles, it is represented by one 1/0 value for each different role combination it can be applied to. A characteristic role, relation configuration for one situation may contain blanks ("-") for those roles or relations that are not characteristic for this situation. A training example contains a vector with specific values reflecting the current role, relation configuration when recording the training example and the corresponding (re)action (given by the supervisor). As the context is defined by the available roles and relations, the description of the situations within this context

The determination of the characteristic role, relation configurations of the sub-situations can be seen as classification problem. The (re)action labels of the training examples can be interpreted as class labels. For each class, we need then to determine the concepts or hypotheses based on the given role, relation vectors of the class. These concepts or hypotheses are then used to construct the characteristic role, relation configurations of the corresponding sub-situation.

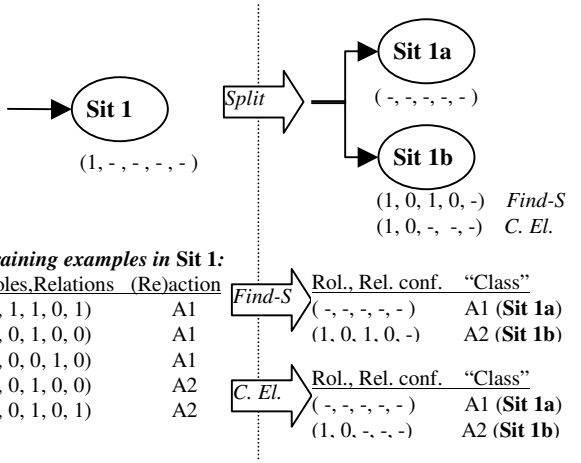

Figure 3: Splitting Situations with Find-S and Candidate Elimination algorithm.

The first considered learning method is the conceptual learning algorithm Find-S ([7] chapter 2.4) which constructs the most specific hypothesis for each (re)action based on the role, relation configurations in the given training examples (Fig. 3). The resulting hypotheses for the created sub-situations often contain, however, specific values for the existence or non-existence of roles or relations that are not necessary or characteristic. As a consequence, small variations in the role, relation configuration may not be covered by the created sub-situations because their hypotheses are too specific.

To produce more general hypotheses for the sub-situations, we consider the conceptual learning algorithm Candidate Elimination ([7] chapter 2.5). This algorithm constructs the most specific and the most general hypotheses for each (re)action based on the role, relation configurations in the given training examples. By combining the most general hypotheses for each (re)action, we construct the role, relation configuration for the corresponding sub-situations (Fig. 3).

Both algorithms Find-S and Candidate Elimination have, however, the restriction that they can only find one conjunctive concept for each (re)action, i.e. if the training examples indicate that a (re)action is to be executed in two different complementary role, relation configurations, Find-S and Candidate Elimination will fail to construct several hypotheses (and thus sub-situations) for this one (re)action. This is due to the fact that neither algorithm can construct disjunctive hypotheses.

We consider Decision Tree learning methods, in particular the algorithm ID3 [8], in order to address the limitation of conceptual learning methods. The idea is to construct a decision tree that classifies the different (re)actions found in the training examples of one situation (Fig. 4).

The attributes of this decision tree are the roles and relation values (0/1 values of the vector). Each leaf of the tree is labeled with a (re)action (class). The path from the root of the tree to the leaf gives the characteristic role, relation configuration for the sub-situation to be created for this (re)action. We can have several leaves with the same (re)action, which corresponds to the creation of several sub-situations for this (re)action (disjunctive hypotheses).

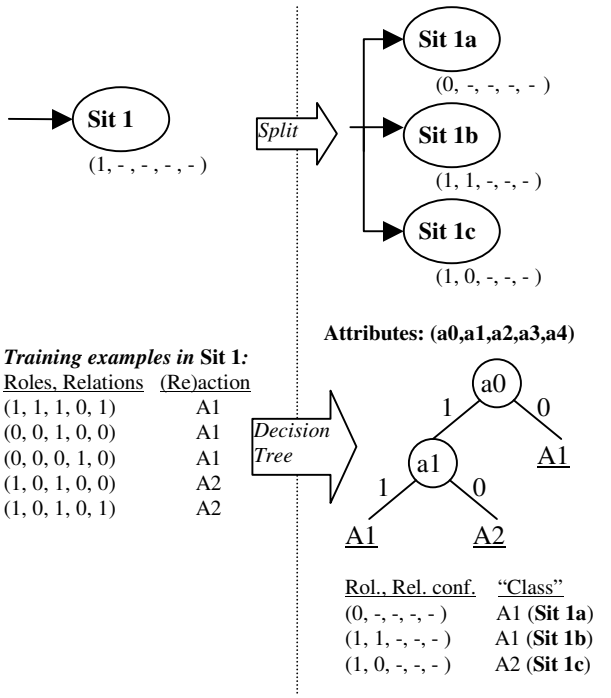

Figure 4: Splitting Situations with Decision Tree algorithm (ID3).

4.3. Learning Roles

If the information supplied by training examples is not sufficient to discriminate characteristic configurations for the sub-situations during the situation split, the creation and learning of new roles need to be considered. This is the case when the supervisor gives different feedback while the system perceives same situation, role and relations configurations.

Table 1: An example for learning a role. The role acceptance test is based on the calculus of the probability of the role value, given the entity position. A Bayesian Classifier could be used.

Roles, Relations	**Feedback**	**Observed Entity Properties**	Associated Role Configuration
(1,0,0,1)	A1	(Entity1, 101, 18) (Entity1, 105, 20) (Entity1, 108, 22)	**NewRole1 = 0**
(1,0,0,1)	**A2**	**(Entity1, 25, 0)** **(Entity1, 21, 2)** **(Entity1, 18, 5)**	**NewRole1 = 1**

When creating a new role, we need to learn the corresponding acceptance test on the properties of the available entities. Learning a role acceptance test can be seen as classification problem. The different supervisor feedback items (different (re)actions) need to be distinguished based on data given on the properties of the entities. Table 1 gives an example referring to the system described in section 5. The entities and their properties are created by a tracking system running on video images of one wide-angle camera. The properties of an entity are its name and its current position in the image. Learning a role acceptance test corresponds here to learning a new characteristic entity position in the image. Given a high amount of sensor-based position data, a Bayesian learning approach for learning the role acceptance test seems to be appropriate for this example.

A problem is to decide which entity or entities to chose for learning the role acceptance test. In the example we only refer to one available entity. If there are several entities available, the entity that allows distinguishing the supervisor feedback items in the best way needs to be chosen. When using a Bayesian approach, the maximum likelihood can be used for determining this entity.

While the development of the situation network, i.e. adapting (re)actions and situations, can be seen as generic approach that is independent of specific perceptual components, learning new roles relies on the properties generated by these components. Thus the choice of the algorithms for learning the acceptance test as well as for determining the relevant entities depends on the available perceptions representing the entity properties. Algorithms for learning role acceptance test have been considered (Table 1) but not been implemented yet.

5. Implementation

A context model for office activity within the SmartOffice environment [9] of the PRIMA group has been designed and implemented. In this environment, entities are created by a robust tracking system [10].

Figure 5: Video image of the wide-angle camera of SmartOffice. Four presence detection zones of the tracking system are indicated. A white box next to the door is used for the creation of new targets (entities). One person is currently tracked.

The position of the created entities determines several roles like comes_in or works_on_PC (Fig. 5). Additional roles are determined by the login of an entity (person) to a computer in the environment or specific appointments marked in the agenda of the logged entity (person). The not_same_entity_as relation is used to distinguish entities in the environment. The (re)actions of the system are based on the control of the Linux music player and the projection of different messages or presentations on different surfaces in the environment. The learning algorithms run on data base tables containing a representation of the current situation network and the training examples. A control process programmed in the forward chaining rule programming environment Jess [11] is used to execute the situation

network. This situation network represented by rules is automatically generated from the data base tables of the learning algorithms. The supervisor feedback cannot be given while the user is acting in the environment (i.e. while the control process is running). Thus the control process and the learning algorithms need, at present, to run sequentially and not in parallel.

6. Evaluation and Results

To evaluate our method, two experiments have been executed on the predefined context model of the SmartOffice environment (figure 6). The experiments have the same goal concerning the evolution of the system services. The supervisor gives feedback based on these goals during the experiments. As we focus on the correct execution of the system services, we do a cross-validation by adapting the predefined situation network using the supervisor feedback of the first experiment and by evaluating the second experiment on the adapted situation network (and inverse). The evaluation is done on the number of correctly classified training examples, i.e. correctly executed (re)actions, as well as on the review of the adaptations of the predefined situation network.

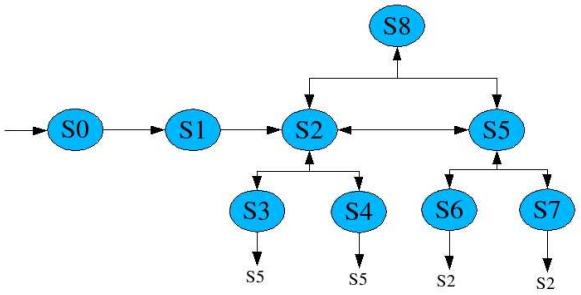

Figure 6: Context model of the SmartOffice environment. Important Situations are **S0** (empty room), **S1** (newcomer enters SmartOffice), **S2** (Person connects to and works on PC), **S5** (Connected Person sits on couch) and **S8** (Presentation in SmartOffice).

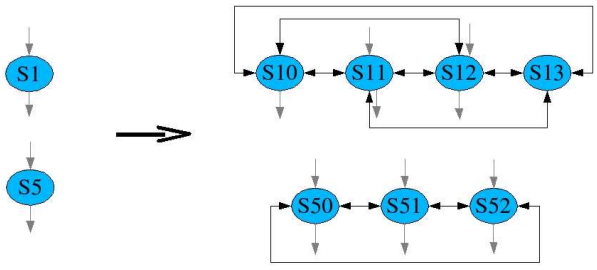

Figure 7: Structural adaptations performed on the predefined context model of the SmartOffice environment by the method (Find-S, Candidate Elimination and Decision Tree algorithm). Situations **S1** and **S5** have been split into sub-situations.

The goal of both experiments was to integrate the correct turn-on and turn-off of the Linux music player depending on the activities (=roles, relations) of the user. The music player should be switched on when a newcomer sits on the couch to have a rest, and switched off when the newcomer starts speaking or leaves the couch (concerned situation: **S1**). The music player should similarly be switched on and off for a connected person (concerned situation: **S5**). Figure 7 shows the adaptations of the concerned situations after the integration of the supervisor feedback. **S1** has been split into additional sub-situations integrating sitting down on couch (**S11**), speaking on couch (**S12**) and leaving couch (**S10**). The additional sub-situations of **S5** integrate sitting down on couch (**S51**) and speaking on couch (**S52**).

Table 2: Confusion matrix for (re)action execution (Find-S).

Find-S	A0	A8	A9
A0	0.87	0.04	0.09
A8	0.50	0.50	0.00
A9	0.50	0.00	0.50

Table 2, 3 and 4 show the results of the (re)action execution in the form of confusion matrices. (A8 switches on the music player, A9 switches off the music player, and A0 is the "do nothing" (re)action).

Table 3: Confusion matrix for (re)action execution (Candidate Elimination).

C. El.	A0	A8	A9
A0	0.91	0.04	0.04
A8	0.66	0.33	0.00
A9	0.75	0.00	0.25

In all experiments, the structural development of the situation network corresponds to the expected changes. Concerning the correct classification of the training examples, i.e. the correct execution of the (re)actions, the Decision Tree algorithm (ID3) gives the best results.

Table 4: Confusion matrix for (re)action execution (Decision Tree, ID3).

D Tr.	A0	A8	A9
A0	0.83	0.09	0.09
A8	0.00	1.00	0.00
A9	0.00	0.00	1.00

The improved results of Decision Tree approach are due to the fact that this algorithm supports disjunctive hypotheses. However, the Decision Tree algorithm tends to construct "too general" hypotheses for the sub-situations, which can lead to several inappropriate classifications. This is due to the fact that the Decision Tree algorithm prefers small trees to large trees, which means that general hypotheses are preferred to specific hypotheses for the sub-situations.

7. Conclusions

We have presented a learning method for evolving system services to changing user needs in an intelligent environment. The intelligent environment has been modeled as a situation network. This network is adapted according to feedback given by a supervisor using an algorithmic learning method. The results of the method are encouraging. The system services desired by the human supervisor are correctly integrated into the situation network structure.

Given supervisor feedback and generated training examples are often not sufficient to decide which adaptation must be done to the situation network. The proposed role learning concept can help extending the training examples by additional roles and hence discriminating the necessary

adaptations of the situation network. However, especially graph optimization opens a wide range of possible adaptations. Two different adaptations may cover the same (optimal) number of training examples. The two corresponding situation networks will, however, not have the same “meaning” for the supervisor. A possible solution is the extension of the learning to an interactive process. The learning system will verify ambiguous choices by asking the supervisor and the supervisor can intervene and correct when decisions of the learning system are wrong.

The method presented in this paper relies on the correct detection of roles and relations, which are seen as perception modules encapsulating perception error handling. Further, the supervisor feedback needs to be consistent, which is not always the case in reality. Thus the focus of our future research will concern the extension of the context model and of the learning algorithms to fuzzy or probabilistic values by integrating the confidence values of the perceptual processes.

References

[1] A.F. BOBICK, S.S. INTILLE, J.W. DAVIS, F.BAIRD, C.S. PINHANEZ, L.W. CAMPELL, Y.A. IVANOV, A. SCHUTTE, AND A. WILSON (1999). The KidsRoom: A Perceptually-Based Interactive and Immersive Story Environment. Presence (USA). 8(4), p. 369-393.

[2] M. MUEHLENBROCK, O. BRDICZKA, D. SNOWDON, AND J.-L. MEUNIER (2004). Learning to Detect User Activity and Availability from a Variety of Sensor Data. IEEE International Conference on Pervasive Computing and Communications (PerCom ’04). p. 13-23.

[3] E. HORVITZ, J. BREESE, D. HECKERMAN, D. HOVEL, AND K. ROMMELSE (1998). The Lumiere Project: Bayesian User Modeling for Inferring Goals and Needs of Software Users. Uncertainty in Artificial Intelligence. Proceedings of the Fourteenth Conference. p. 256-265.

[4] J.L. CROWLEY, J. COUTAZ, G. REY, AND P. REIGNIER (2002). Perceptual Components for Context Aware Computing. UbiComp 2002: Ubiquitous Computing. 4th International Conference. Proceedings (Lecture Notes in Computer Science). 2498, p. 117-134.

[5] L. R. RABINER (1990). A Tutorial on Hidden Markov Models and selected Applications in Speech Recognition. Readings in speech recognition. p. 267-296.

[6] J. R. QUINLAN (1990). Learning Logical Definitions from Relations. Machine Learning. 5(3), p. 239-266.

[7] T.M. MITCHELL (1997). Machine Learning. McGraw Hill, New York, USA, international edition.

[8] J.R. QUINLAN (1986). Induction of Decision Trees. Machine Learning. 1(1), p. 81-106.

[9] CH. LE GAL, J. MARTIN, A. LUX, AND J.L. CROWLEY (2001). SmartOffice: Design of an Intelligent Environment. IEEE Intelligent Systems. 16(4), p. 60-66.

[10] A. CAPOROSSI, D. HALL, P. REIGNIER, AND J.L. CROWLEY (2004). Robust Visual Tracking from Dynamic Control of Processing. Sixth IEEE International Workshop on Performance Evaluation of Tracking and Surveillance. Prague, Czech Republic.

[11] JESS (1995). The rule engine for Java. http://herzberg.ca.sandia.gov/jess/.