UNIVERSITY OF CRETE
COMPUTER SCIENCE DEPARTMENT

USING CONSTRAINT OPTIMIZATION FOR CONFLICT RESOLUTION AND DETAIL CONTROL IN ACTIVITY RECOGNITION

Filippaki Chrysi

Master's Thesis

Heraklion, September 2011

UNIVERSITY OF CRETE
COMPUTER SCIENCE DEPARTMENT

Using Constraint Optimization for Conflict Resolution and Detail Control in Activity Recognition

Thesis submitted by

Chrysi Filippaki

in partial fulfilment of the requirements for the
Master of Science degree in Computer Science

Author:

_____
Chrysi Filippaki

Supervisory Committee:

_____
Grigoris Antoniou
Professor, Thesis Supervisor

_____
Ioannis Tsamardinos
Assistant Professor, Thesis Supervisor

_____
Dimitris Pleksousakis
Professor, Member

Departmental Approval:

_____
Angelos Bilas
Associate Professor, Chairman of Graduate Studies

Heraklion, September 2011

Using Constraint Optimization for Conflict Resolution and Detail Control in Activity Recognition

Filippaki Chrysi

Master of Science Thesis

Computer Science Department, University of Crete

Abstract

Activity recognition is an open domain which can be used in a wide spectrum of applications, ranging from video surveillance to monitoring and assisting children, elderly or sick people. Some prior work on activity recognition focuses on identifying all occurrences of a specific type of activity. In more general activity recognition settings however (e.g., Ambient Assisted Living), a system should identify all of user activities and their relations, e.g. their temporal sequence or subsumption (for activities hierarchically organized). Due to noise or other imperfections a naïve recognition system may report activities that are logically inconsistent with each other, such as the user is sleeping on the couch and at the same time is watching TV.

In this work, we develop a rule-based activity recognition system for hierarchically-organized complex activities that returns only logically consistent sets of activities (scenarios). This is achieved by explicitly formulating activity conflicts as a weighted partial maximum satisfiability problem (Weighted Partial MaxSAT) such that any solution to it corresponds to a set of identified activities (scenario) that do not conflict. The system also has the ability to adjust the desired level of detail of the scenarios returned, e.g., (1 hour TV-watching, 1' phone-call, 2 hours TV-watching) vs. (3 hours of TV-watching). This is accomplished by assigning preferences to clauses of the Weighted Partial MaxSAT problem. Each complex activity's weight is calculated by taking into account its confidence, temporal duration and number of used atomic activities. We note that the optimization techniques presented could accommodate other types of preferences and be generalized to other settings. The system is implemented and evaluated in an Ambient Intelligence experimental space.

Χρήση Βελτιστοποίησης Περιορισμών για Επίλυση Συγκρούσεων και Έλεγχο Λεπτομέρειας στην Αναγνώριση Δραστηριοτήτων

Φιλιππάκη Χρυση

Μεταπτυχιακή Εργασία

Τμήμα Επιστήμης Υπολογιστών

Περίληψη

Η αναγνώριση δραστηριοτήτων είναι ένα ευρύ πεδίο, που μπορεί να χρησιμοποιηθεί σε ένα μεγάλο φάσμα εφαρμογών, από τη παρακολούθηση μέσω βίντεο ως την επιτήρηση και τη υποβοήθηση παιδιών, ηλικιωμένων ή αρρώστων. Ορισμένες από τις προηγούμενες εργασίες στην αναγνώριση δραστηριοτήτων επικεντρώνεται στον εντοπισμό όλων των εμφανίσεων ενός συγκεκριμένου τύπου δραστηριότητας. Ωστόσο σε πιο γενικά πλαίσια αναγνώρισης δραστηριοτήτων (π.χ., Περιβάλλοντα Υποβοηθούμενης Διαβίωσης), το σύστημα πρέπει να εντοπίσει όλες τις δραστηριότητες των χρηστών και τις μεταξύ τους σχέσεις, π.χ. χρονική ακολουθία και υπαγωγή τους (για τις δραστηριότητες που οργανώνονται ιεραρχικά). Λόγω θορύβου ή άλλων ατελειών ένα απλό σύστημα αναγνώρισης μπορεί να αναφέρει δραστηριότητες που είναι λογικά ασυμβίβαστες μεταξύ τους, όπως για παράδειγμα ο χρήστης κοιμάται στον καναπέ και την ίδια στιγμή βλέπει τηλεόραση.

Στην εργασία αυτή, έχουμε αναπτύξει ένα σύστημα αναγνώρισης δραστηριοτήτων βασισμένο σε κανόνες για ιεραρχικά οργανωμένες σύνθετες δραστηριότητες, που επιστρέφει μόνο λογικά συνεπή σύνολα δραστηριοτήτων (σενάρια). Αυτό επιτυγχάνεται με την σαφή διατύπωση των συγκρούσεων δραστηριοτήτων ως ένα σταθμισμένο πρόβλημα μερικής ικανοποιησιμότητας (Weighted Partial MaxSAT), έτσι ώστε οποιαδήποτε λύση του, να αντιστοιχεί σε ένα σύνολο αναγνωρισμένων δραστηριοτήτων (σενάριο) οι οποίες δεν έρχονται σε σύγκρουση. Το σύστημα έχει επίσης τη δυνατότητα να ρυθμίζει το επιθυμητό επίπεδο λεπτομέρειας των σεναρίων που επιστρέφονται, π.χ., (1 ώρα παρακολούθηση

τηλεόρασης, 1 λεπτό συνομιλίας στο τηλέφωνο, 2 ώρες παρακολούθηση τηλεόρασης) έναντι (3 ώρες παρακολούθηση τηλεόρασης). Αυτό επιτυγχάνεται με την ανάθεση προτιμήσεων στις προτάσεις του προβλήματος ικανοποιησιμότητας. Το σύστημα υλοποιήθηκε και αξιολογήθηκε σε ένα πειραματικό χώρο Διάχυτης Νοημοσύνης.

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον επόπτη καθηγητή μου, κ. Γρηγόρη Αντωνίου που μου εμπιστεύτηκε την εργασία αυτή, καθώς και για την καθοδήγηση και τις συμβουλές του με τις οποίες συνέβαλλε καθοριστικά στην επιτυχή ολοκλήρωση των μεταπτυχιακών μου σπουδών. Επίσης ευχαριστώ θερμά τον δεύτερο επόπτη της μεταπτυχιακής μου εργασίας τον καθηγητή κ. Ιωάννη Τσαμαρδίνο για το χρόνο που διέθεσε μέσω των συναντήσεων μας, για τις συμβουλές του, τις γνώσεις του και τις εύστοχες παρατηρήσεις του σε όλη την διάρκεια της συνεργασίας μας. Επισης ευχαριστώ και τον καθηγητή κ. Πλεξουσάκη για την συμμετοχή του στην εισηγητική επιτροπή.

Εν συνεχεία, οφείλω να αναφερθώ στον διδάκτορα Θεόδωρο Πάτκο για τις συμβουλές του και την ανεκτίμητη βοήθεια του στην ολοκλήρωση της εργασίας μου κατά την διάρκεια των μεταπτυχιακών μου σπουδών.

Ευχαριστώ όλα τα μέλη του Εργαστηρίου Πληροφοριακών Συστημάτων καθώς και το Ινστιτούτο Πληροφορικής του Ιδρύματος Τεχνολογίας και Έρευνας για την οικονομική και υλικοτεχνική υποστήριξη που μου παρείχε κατά την διάρκεια των μεταπτυχιακών σπουδών μου.

Επίσης θα ήθελα να ευχαριστήσω τα μέλη της ομάδας του Εργαστηρίου Αλληλεπίδρασης Ανθρώπου-Υπολογιστή και που με βοήθησαν να ολοκληρώσω την πρακτική εφαρμογή (demo) της μεταπτυχιακής μου εργασίας στο AmI Sandbox και ιδιαίτερα τον Γιάννη Δρόσση, την Χρυσούλα Μπιρλιράκη τον Νίκο Φτυλιτάκη και τον Νίκο Καζέπη. Επίσης ευχαριστώ την Ειρήνη Γενιτσαρίδη και την Μαίρη Κουτράκη που συμμετείχαν στα demo μου.

Ευχαριστώ την Άννα, την Ειρήνη, την Βαγγελιώ, τον Μανόλη και όλους τους υπόλοιπους φίλους μου και συμφοιτητές μου για την ψυχολογική υποστήριξη, την θετική διάθεση και αισιοδοξία που μου μετέφεραν όλον αυτά τα χρόνια.

Ένα μεγάλο ευχαριστώ στον Χρυσόστομο που ήτανε πάντα δίπλα μου, με στήριζε και μου συμπαραστεκότανε σε όλες τις εύκολες και δύσκολες στιγμές τον σπουδών μου.

Τέλος, ευχαριστήσω θερμά τους γονείς μου Χαράλαμπο και Μαρτίνα, τους αδερφούς μου Νίκο και Φίλιππο και όλη την οικογένειά μου για την στήριξη, την αγάπη και συμπαράσταση που μου παρείχαν και εξακολουθούν να μου παρέχουν σε όλα μου τα βήματα.

Σας ευχαριστώ!

Στους γονείς μου Χαράλαμπο και Μαρτίνα

Contents

List of Figures

List of Tables

1 Introduction

1.1 Introduction

The population of developed countries is aging at an alarming rate due to the increase in life expectancy and decrease in birth rate. The percentage of the European population (EU27) aged 65 years and over is anticipated to rise from 17% in 2010 to 30% in 2060 and those aged 80 and over rising from 5% to 12% over the same period [1]. Moreover nearly three quarters of older adults suffer from one or more chronic diseases, and require some degree of care due to loss of function, according to the Centers for Disease Control (CDC). Therefore it is crucial that elderly individuals can live longer independently with minimal support from their relatives or caregivers.

All these social changes will lead to enormous challenges for the healthcare systems, but at the same time they will offer innovation and business opportunities for technology providers in the field of **Ambient Intelligence (AmI)**. AmI refers to electronic environments that are sensitive and responsive to the presence of people. In AmI environments the emphasis is given on greater user-friendliness, more efficient services support, user-empowerment, and support for human interactions [2]. People are surrounded by technologies that are embedded (non-invasive devices, distributed throughout the environment or directly integrated into appliances or furniture) adaptive (responsive to the user and the user's environment) and personalized (adjusting to the users' needs).

So in an AmI world, devices work together in order to support people in carrying out their everyday life activities, in an easy natural way using information and intelligence that is hidden in the network connecting these devices. As they are getting smaller and more integrated into the environment, the technology disappears into the surroundings until only the user interface remains perceivable by users.

Ambient Assisted Living (AAL) is the utilization of ambient intelligence in the respective social domains of ageing at home. The concept of AAL refers to intelligent

systems of assistance for a better, healthier and safer life in the preferred living environment. The goal of AAL is to improve the life quality (physical, mental and social well-being) for everyone. It focuses on elder persons and can help them to stay healthier and to live longer, thus extending their active and creative participation in the community [3].

Especially in the domain of AAL a central challenge is the recognition of user's activities, so that assistance services can be offered. **Activity recognition** aims to recognize the actions and goals of one or more agents from a series of observations on the agents' actions and the environmental conditions. It can be used not only for helping elderly people but also for video surveillance and for monitoring and consequently assisting children or sick people. So potential applications of an activity recognition system include several AmI environments, such as intelligent homes, workspaces, classrooms etc (figure 1.1).

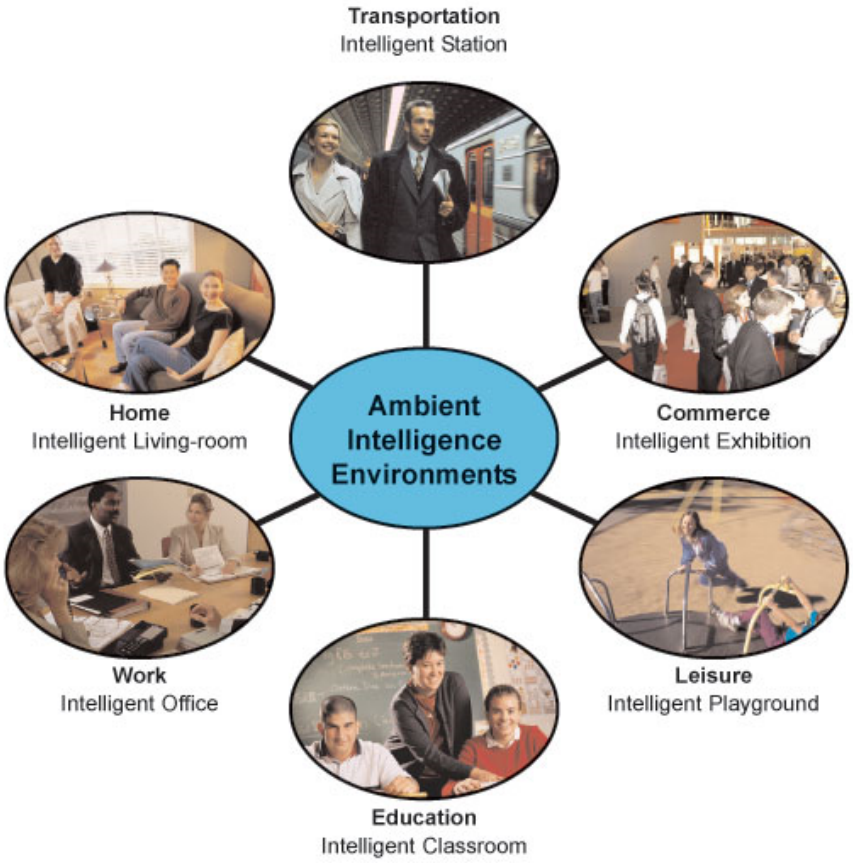

Figure 1.1 Thematic areas of Ambient Intelligence include a) Home: Intelligent living room b) Work: Intelligent Office c) Education: Intelligent classroom d) Transportation: Intelligent transportation e) Commerce: Intelligent exhibition f) Leisure: Intelligent playground

1.2 The Problem

Some prior work on activity recognition focuses on identifying all occurrences of a specific type of activity, e.g., people abandoning their luggage in an airport [4]. In more general activity recognition settings however, a system should identify all of user activities and their relations, e.g. their temporal sequence or subsumption (for activities hierarchically organized). In the later case, it is important for the system to report activities that are logically consistent and avoid reported scenarios, such as the user is sleeping on the couch and at the same time is watching TV.

Another consideration is deciding the level of detail granularity of the reported activities, depending on the context of use. Some applications may require very detailed recognition (1 hour TV-watching, 1 minute phone-call, 1 hour TV-watching), while other coarser (2 hours TV-watching, or 2 hours relaxing at home). For example, a nurse requesting a report from an AAL system on a patient may be interested in all of her daily activities and their exact temporal duration in order to determine whether and when the patient is taking her medication and whether she needs help with some activities or not. A doctor considering the patient lifestyle may only be interested in her sleep patterns, amount of rest, exercise and regularity of eating schedule.

1.3 Contribution

This work proposes a logic and rule-based system that deals with noisy sensors, and uncertainty of primitive event detection, resolves conflicts of the identified activities and reports logically consistent scenarios; finally, the system takes parameters that adjust the preferences for the level of abstraction and the amount of detail of the recognized activities. To the extent of our knowledge, this is the first system that addresses both activity conflicts and preferences about the level of detail.

The system input is a list of recognized atomic, instantaneous events (e.g., turn-on-TV, lie-on-bed, start-slideshow), also called atomic activities, from the sensors and their timing (figure 2.2). Complex Activities are recognized based on predefined hierarchical rules, which express temporal and spatial patterns between lower-level activities. Uncertainty is addressed by allowing the presence of optional activities: a complex activity may be detected even if some optional primitive activity is not detected.

The presence of the optional activities increases the confidence of the recognition, expressed as a confidence factor. The temporal aspects of our rules are explicitly represented; the intervals of the recognized complex activities are computed and saved in our knowledge base for further processing. Both the confidence factors and the temporal intervals of the complex activities are computed from the factors and intervals of their constituent lower-level activities.

The system first creates a list of all potential complex activities and then detects all pairs of conflicting activities. The problem of resolving the conflicts is converted into a weighted partial maximum satisfiability problem (Weighted Partial MaxSAT) such that any solution to it corresponds to a set of identified activities (scenario) that do not conflict. The system also has the ability to adjust the desired level of detail in the scenarios returned. In order to achieve this, preferences (weights) are assigned to clauses of the Weighted Partial MaxSAT problem. Each complex activity's weight is calculated by taking into account its confidence, temporal duration and number of used atomic activities. Then the optimal solution is found by maximizing an objective function. We note that the optimization techniques presented could accommodate other types of preferences and be generalized to other settings.

The system is implemented and evaluated in a real Ambient Intelligence experimental space: a real user interacted with the system for a period of time and her activities are accurately recognized. The level of detail of the recognized scenario is demonstrated to be easily controlled by the use of two preference parameters. The system is shown to be robust to noisy inputs with simulation studies.

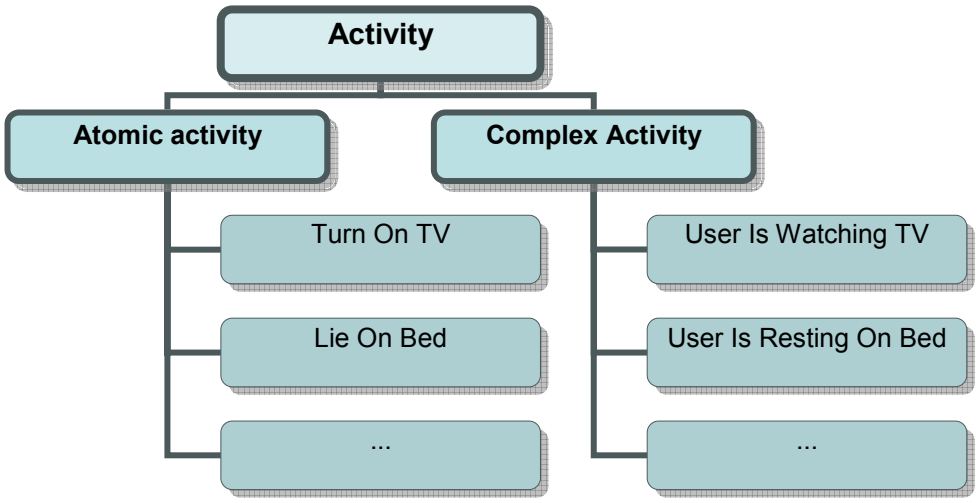

Figure 1.2 Structure and examples of atomic and complex activities.

1.4 Motivating Scenarios

We now present a theoretical motivating scenario and a running example scenario to use for modeling and recognition of user activities. The theoretical motivating scenario is a fictional scenario illustrating our vision of how activity recognition can be integrated into an AAL system. The running example scenario is a more specific application scenario that we use to explain the concepts of our approach in the following sections.

1.4.1 Theoretical Motivating Scenario

Suppose Bob is an elderly person living in an AAL environment. His activities are recognized in order to support him with assistance services. His nurse or caregiver is getting a report from all of his daily activities and their exact temporal duration, in order to determine whether and when Bob is taking his medication, and whether he needs help with some activities or not.

Moreover by recognizing more abstract activities, that describe longer periods of time, a pattern with his daily routines can be extracted. Thereafter whenever a significant differentiation in his habits is detected the system could inform his doctor. The doctor considering the patient's lifestyle is only interested in Bob's sleep patterns, amount of rest, exercise and regularity of eating schedule. For example if suddenly Bob is less active and sleeps more hours than usually it could be a sigh of some health problem or even depression.

1.4.2 Running Example Scenario

Mary is relaxing in her home after a long day at work. She first lies in her bed for a while. Then she watches TV until the telephone rings. She answers the call and a colleague is asking her to watch a slideshow he prepared, for a project they are working on. She goes to the next room where she watches the slideshow and then she returns to her bedroom and watches again TV. All of her activities could be considered a type of a more general activity "User is relaxing at home", except for the activity were she is watching a slideshow about her work.

1.5 Organization of this thesis

The rest of this thesis is organized as follows:

Chapter 2 discusses related work and gives an overview about the existing activity recognition systems. It also makes a comparison between those systems and our work.

Chapter 3 introduces our system, providing its architecture and its components. Moreover it presents the basic concepts involved in our activity recognition system and it continues with the definitions of our operators and optimization function.

Chapter 4 presents the tools and systems used in the thesis namely JESS, Sat4J and Java. It explains the expressiveness of our language and the complexity of our rules based on JESS' algorithm Rete.

Chapter 5 demonstrates the system with the implementation of the running example scenario (presented in section1.4.2) in our lab - an Ambient Intelligence environment. Also experimental results based on simulation studies are presented showing how the system responds to noisy inputs.

Finally **Chapter 6** concludes this thesis and identifies issues that are worth further research.

2 Related Work

There is a variety of applications and approaches which aim to recognize human activities. There are three main approaches in activity recognition frameworks, logic-based, probabilistic-based and combinations of them. We are going to examine each of these categories in the following sections.

2.1 Probabilistic-based approaches

Several probabilistic algorithms have been used to build activity recognition systems. In probabilistic systems, such as [5] and [7] where Hidden Markov Models and Conditional random fields have been used respectively, noise and uncertainty are handled well. Activity recognition is usually performed using maximum likelihood estimation given observation sequences. However they require training data and do not cope with conflicts or abstraction preferences.

2.1.1 Hidden Markov Models

The Hidden Markov Model (HMM) and its variants are among the most popular modeling techniques for activity recognition. They are based on data-driven supervised learning. An HMM has a graph structure associated with it, describing a model where the Markov assumption holds i.e. where the current observations are dependent only on the current state and the current state is only dependent upon the state at the previous "time slice" [10]. As the HMM structure is fixed and repetitious, the probability of long sequence of observations can be defined by specifying a set of parameters. The parameters' number depends on the number of possible states and observation symbols.

An HMM activity model is defined by observation symbols related to the chosen abstraction scheme. The number of states is chosen empirically while the parameters of the HMM model can be specified manually using knowledge of the activity domain or learned from training data. An HMM activity model is trained for each activity un-

der consideration in order to discriminate them. Then test examples are evaluated to determine how possible they are to have been generated by each of the HMM models and the activity with the highest probability is recognized [10].

Patterson et al. [11], collected sensor data from the morning routines of users performing eleven interleaving activities. These activities share a large number of sensor-tagged objects the subjects can interact with. In this setting, HMMs with one state for each activity are employed as prediction algorithm.

Some HMM extensions have been proposed to incorporate the inherent hierarchical composition of video activities into the activity models. In Hierarchical Hidden Markov Models (HHMM) each possible state is represented by a lower-level HMM. In [6] Nguyen et al. have examined the problem of recognizing single person indoor activities (household or office tasks) from movement trajectories extracted from camera data. In [12] the use of a multilayer representation of HMMs, named LHMMs, that reasons in parallel at different levels of temporal detail is described.

However HMMs and their variants do not cope with conflicts or abstraction preferences. Another drawback is that the number of parameters that need to be set in an HMM is huge. For example, for a very simple three-state HMM, there are a total of 15 parameters that need to be evaluated. As a result of the above, the amount of data that is required to train an HMM is very large. Obtaining substantial amounts of labeled data is often a bottle-neck for these approaches.

2.1.2 Conditional Random Fields

As the assumption of independence between the states is usually made in generative models such as HMMs, it is difficult to accommodate multiple overlapping features or long-range dependencies among activities with them. Conditional random fields (CRFs) do not make the independence assumption between activities, and therefore they have the possibility to incorporate both overlapping features and long-range dependencies into the model.

CRFs have been used for activity recognition in [7]. In [8] they used skip-chain conditional random fields (SCCRFs), a variant of CRFs that can be used to model interleaved activities. Factorial Conditional Random Fields (FCRFs), another variant of CRFs, can be used to model concurrent activities [9]. However, like other learning-

based techniques, they require that a predicting instance must have its model presented in the training dataset. This implies that the training dataset has to be large enough to build the complete models for interleaved and concurrent activities. Any partial model will result in a loss of recognition accuracy. However in real life there exists a large variety of ways in which activities can be interleaved and performed concurrently. Therefore it may be not possible to construct a complete model by training. Moreover CRFs and their variants lack the ability to adjust the abstraction level in their results and to detect possible conflicts.

2.2 Logic-based approaches

Logic-based systems on the other hand have predefined rules for recognizing activities based on observed facts. A significant limitation is that they do not deal with missing activities and noise.

2.2.1 Event Caclulus dialects

The Event Calculus (EC) first presented by Kowalski and Sergot in 1986 [16] is a set of first-order predicate calculus, including temporal formalism, for representing and reasoning about events and their effects. It provides a more generic and expressive representation of events, taking advantage of the full power of logic programming on which it is based. EC supports complex temporal as well as atemporal representation and reasoning. It is based upon general axioms concerning the notions of events, relationships, and the periods for which they hold. More specifically EC defines a model of change in which events happen at time-points and initiate and/or terminate time-intervals over which some properties (time-varying fluents) of the world hold. The basic idea is to state that *fluents* are true at particular time-points if they have been initiated by an event at some earlier time-point and not terminated by another event in the meantime. The EC embodies a notion of default persistence, according to which fluents are assumed to persist until an event occurs which terminates them. The clauses they presented about their axioms run reasonably efficiently as a PROLOG program. However, they should be regarded not as a program but as a specification. In [17] and [18] to EC dialects are implemented.

2.2.1.1 LTAR-EC

Artikis et al. developed LTAR-EC (event calculus for long-term activity recognition), an activity recognition system consisting of an Event Calculus dialect implemented in Prolog [17]. The input of the system is a set of time-stamped short-term activities (atomic activities in our context) detected on video frames. The output of the system is a set of recognized long-term activities (complex activities in our context), which are predefined temporal combinations of short-term activities. So they defined a set of long-term activities of interest, such as "fighting", "leaving an object" and "meeting", as temporal combinations of short-term activities e.g. "walking", "running", and "inactive" (standing still).

According to EC short-term activities are represented by events and long-term activities are represented by fluents. An event description in LTAR-EC includes axioms that define, among other things, the event occurrences (with the use of the happensAt and happensFor predicates), the effects of events (with the use of the initiatedAt and terminatedAt predicates), and the values of the fluents (with the use of the initially, holdsAt and holdsFor predicates). Figure 2.1 summarizes the main predicates of LTAR-EC.

Predicate	Meaning
happensAt(E, T)	Event E is occurring at time T
happensFor(E, I)	I is the list of maximal intervals during which event E takes place
initially($F=V$)	The value of fluent F is V at time 0
holdsAt($F=V$, T)	The value of fluent F is V at T
holdsFor($F=V$, I)	I is the list of maximal intervals for which $F=V$ holds continuously
initiatedAt($F=V$, T)	At time T a period of time for which $F=V$ is initiated
terminatedAt($F=V$, T)	At time T a period of time for which $F=V$ is terminated

Figure 2.1 Main Predicates of LTAR-EC

LTAR-EC's approach does not allow for uncertainty values in the rules of activity definitions. Thus it can not address the issues arising from incomplete atomic activity narratives, inconsistent annotation of atomic and complex activities, and a limited dictionary of activity types. We use confidence values in our rules and facts to express

the noise and missing activities in our system. Another limitation is that their system does not store the outcome of query computation i.e. the intervals of the recognized complex activities. Consequently they can not recognize complex activities in a second level, based on the already recognized complex activities, as we do. Also they do not return scenarios but only single activities and therefore do not detect possible conflicts.

2.2.2 Chronicle Recognition System

Dousson et al. presented the Chronicle Recognition System (CRS) [19] which is a purely temporal reasoning system that allows for very efficient and scalable activity recognition. A *chronicle* can be seen as a complex activity, which is expressed in terms of a set of atomic activities (events in their context), linked together by time constraints and possibly, a set of context constraints. A chronicle is represented by a time constraint graph labeled by predicates (figure 2.3-up). The language of CRS relies on a reified temporal logic, where propositional terms are related to time-points or other propositional terms. Time is considered as a linearly ordered discrete set of instants. The language includes predicates for persistence, event absence and event repetition. Figure 2.2 presents the CRS predicates.

Predicate	Meaning
`event(E, T)`	Event `E` takes place at time `T`
`event(F:(?V1,?V2),T)`	An event takes place at time `T` changing the value of property `F` from `V1` to `V2`
`noevent(E, (T1,T2))`	Event `E` does not take place between `[T1,T2)`
`noevent(F:(?V1,?V2), (T1,T2))`	No event takes place between `[T1,T2)` that changes the value of property `F` from `V1` to `V2`
`hold(F:?V, (T1,T2))`	The value of property `F` is `V` between `[T1,T2)`
`occurs(N,M,E,(T1,T2))`	Event `E` takes place at least `N` times and at most `M` times between `[T1,T2)`

Figure 2.2 Predicates of CRS

Figure 2.3 shows the mechanism of the recognition algorithm on a small example. An instance of event is a pair *(e, t)* where *t* is the date of the event and *e* is its type. Figure

2.3 - up shows a chronicle which contains four events: the vent *e* (if instantiated) must occur between *1* and *3* time-units after an instantiation of *f*, the event *g* must occur between *0* and *3* time-units after *e* and between *−1* and *4* time-units of time after *e′*. When CRS receives event *f* at *1*, it creates the new instance *I1* and updates the forthcoming window of the node *e*. Then a new event *f* occurs at *3*, instance *I2* is created (the forthcoming windows of *I1* is updated "using" a clock tick set at *3*). Finally *e* occurs, *I3* is created (from *I2*) and *I1* is destroyed as no more event *e* could from now be integrated into (instance *I2* is not destroyed, waiting for another potential *e* between 5 and 6). As all events of *I3* are instantiated the chronicle *I3* is recognized.

A significant limitation of the CRS language is that it does not allow mathematical operators in the constraints of atemporal variables e.g. they can not compute the distance between two people/objects. Therefore, it cannot be used for activity recognition in applications requiring any form of spatial reasoning, or any other type of atemporal reasoning. Moreover the CRS language does not allow for uncertainty representation and is not suitable for event recognition in noisy environments. On the contrary, our system allows the assignment of confident values for each event, and is also capable of performing both temporal and atemporal computations. Finally they do not handle conflict detection and detail control.

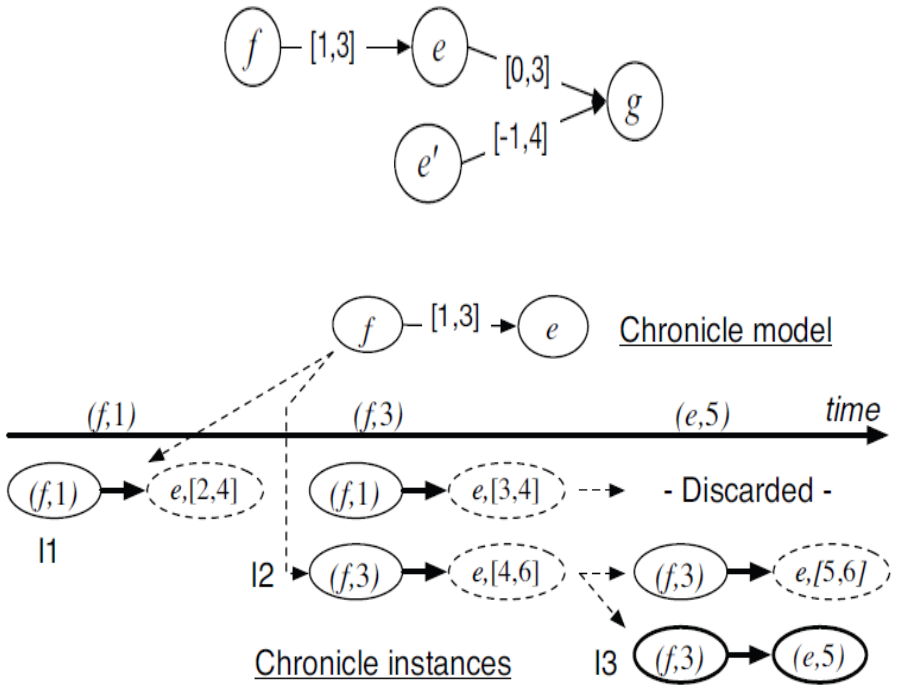

Figure 2.3 A chronicle (up). Created instances of this chronicle by the incoming event stream: *(f, 1) (f, 3) (e, 5)* (down).

2.3 Logic and probabilistic combinations

There are some approaches that combine logic with probabilities (e.g. [21], [22]), trying to maintain the strengths of logical and probabilistic inference. They allow for uncertainty representation and are thus suitable for event recognition in noisy environments. But the main drawback of these works is that they do not handle conflicts as they return single activities instead of scenarios. Unlike our system they can not adjust the level of detail in their results.

2.3.1 Shet et al.

An activity recognition system based on logic programming is the work presented of Shet and his colleagues [20], [21]. A Prolog based reasoning engine uses the atomic activities in conjunction with predefined rules to recognize various complex activities in the input video streams. These researchers have presented activity definitions such as unattended packages, theft and entry violation. Moreover they have incorporated in their logic programming framework a mechanism for reasoning over rules and facts that have an uncertainty value attached. Positive and negative information from different rules, as well as uncertainties from detections are integrated within a bilattice framework. This framework proposes complex activities and also generates proofs or justifications for them.

More specifically uncertainties assigned to the rules that guide reasoning, as well as detection uncertainties reported by the low level detectors, are taken from a set structured as a bilattice. These uncertainty measures are ordered along two axes, one along the source's (applied rule's) degree of information and the other along the agent's degree of belief. Multiple sources of information may be available as there can be many rules deriving the same proposition (positive or negative forms of it). The $\leq_t$ partial order (agent's degree of belief) indicates how true or false a value is, with *f* being the minimal and *t* being the maximal. The $\leq_k$ partial order indicates how much is known about a particular proposition, with $\perp$ (completely unknown) being the minimal element and $\top$ (representing a contradictory state of knowledge where a proposition is both true and false) being the maximal element. In Figure 2.4 the bilattice square is depicted.

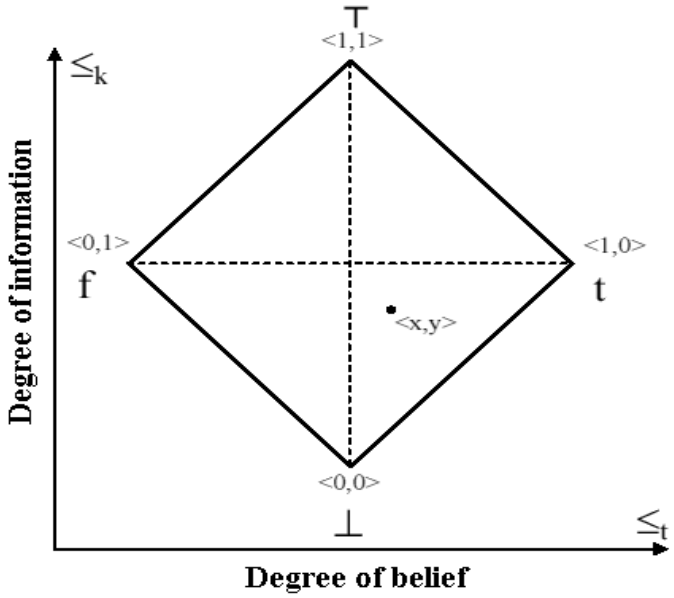

Figure 2.4 The bilattice square (*[0, 1]2, ≤t, ≤k*)

The main drawback of their work is that their temporal representation and reasoning is really poor. They have no rules for computing the intervals in which a complex activity takes place. In contrast in our system the temporal aspects of the definitions are clearly represented. As they do not calculate the interval of recognized complex activities, they can not return activity scenarios explaining longer periods of time.

2.3.2 Markov Logic Networks

Markov Logic Networks (MLNs) [22], unlike EC and CRS, allow for uncertainty representation and are thus suitable for activity recognition in noisy environments. MLNs are useful for handling noisy atomic activity streams, as they combine the strengths of logical and probabilistic inference. A first-order knowledge base (KB) can be seen as a set of hard constraints on the set of possible worlds - if a world violates even one clause, it has zero probability. The basic idea in MLNs is to soften these constraints. So when a world violates one clause in the KB, this world becomes less probable, but not impossible. The fewer clauses a world violates, the more probable it is. Each clause has an associated weight that reflects how strong a constraint it is.

There is already some evidence that Markov logic-type relational models can be used for activity recognition from video [23], [24] or from sensors [25]. Tran and

Davis [23] used MLNs to probabilistically infer activities in a parking lot. However, their MLNs relied on the assumption that an identical atomic activity occurs only once during interactions, limiting itself from being applied to dynamically interacting actors. The system's performance is demonstrated on a number of videos from a parking lot domain that contains complex interactions of people and vehicles.

Their system maintains an undirected network of grounded[1] atoms which correspond to activities that have occurred in the video. At any moment, primitive events (atomic activities in our context) are detected with associated detection probabilities. They are then used to ground logical rules in the Knowledge Base, which generally leads to generating more grounded events. Next, these grounded logical rules are added to the Markov network. The network parameters or structures are revised with these updates. The marginal probability of any complex activities can be determined using probabilistic inference on this network. Figure 2.5 shows an overview of the system.

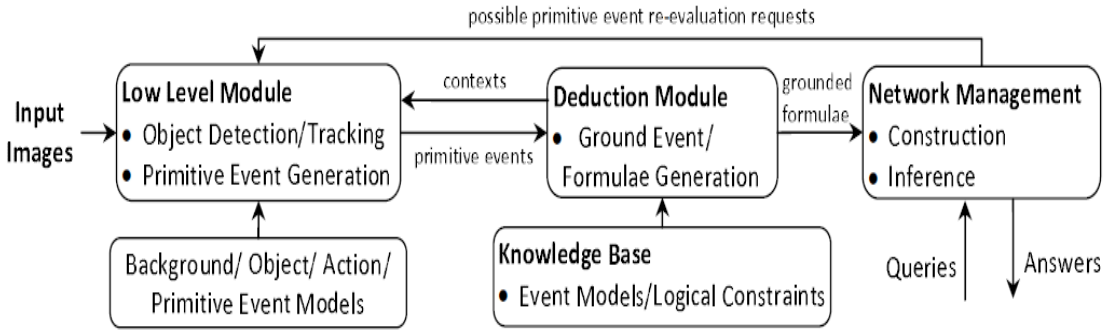

Figure 2.5 Overview of Tran et al. system.

In [24] Biswas et al. introduce a first-order probabilistic model that combines multiple clues to classify human activities from video data. The probabilistic model is implemented as a Dynamic Markov Logic Network (DMLN) that groups fifteen first-order logic propositions. The system is applied to an office setting where only activities that involve an agent and an inanimate object are considered (e.g. talking to the phone, writing with a pen).

In figure 2.6 their DMLN model is presented. Sentence 1 states that activities tend to persist over time (succ(t) means successor to t, i.e. the next frame). Activ-

[1] A *ground term* is a term containing no variables. A *ground atom* or *ground predicate* is an atomic formula all of whose arguments are ground terms.

1	$\forall t \forall a$ Activity$(a,t) \rightarrow$ Activity$(a,$succ$(t))$	8.3
2	$\forall t \forall a_1 \forall a_2$ Activity$(a_1,t) \rightarrow =(a_1,a_2) \vee \neg$ Activity(a_2,t)	∞
3	$\forall t \exists a$ Activity(a,t)	∞
4	$\forall t \forall a$ Activity$(a,t) \rightarrow$ Pose(a,t)	0.7
5	$\forall t$ Activity$(a,t) \wedge$ Useful$(o,a) \rightarrow$ Present(o,t)	0.7
6	Useful(TYPING,KEYBOARD)	∞
7	Useful(MOUSING,MOUSE)	∞
8	Useful(EATING,CANDY)	∞
9	Useful(EATING,APPLE)	∞
10	Useful(DRINKING,SODA)	∞
11	Useful(READING,BOOK)	∞
12	Useful(TALKING,PHONE)	∞
13	Useful(WRITING,PEN)	∞
14	Useful(WRITING,PAPER)	∞
15	$\forall t \forall a$ Activity$(a,t) \rightarrow$ Movement(a,t)	0.2

Figure 2.6 Probabilistic Model as Dynamic Markov Logic Network (sentence weights appear in the right column) used by Biswas et al. system.

ity(*a*,*t*) is a binary variable that means that the person is engaged in activity *a* at time *t*. Sentence 2 establishes a mutual exclusion constraint between simultaneous activities. Sentence 3 states that the user must be doing something. Sentence 4 states that a person's pose will reflect the activity they are currently engaged in. The object recognition component uses the fluents Present(*o*,*t*) (an object of type *o* is present at time *t*) and Useful(*a*,*o*) (objects of type *o* are useful for activity *a*). Sentence 5 states that objects useful to an activity will be present when that activity is engaged in. Sentences 6 through 14 specify which objects are useful for which activities. Sentence 15 states that the activity undertaken influences the location of the movement in the camera image. The fluent *Movement*(*a, t*) states which activity seems likely by analyzing movement alone.

Finally Helaoui et al. describe the use of Markov logic as a declarative framework for recognizing interleaved and concurrent activities incorporating both input from pervasive light-weight sensor technology and common-sense background knowledge [25]. In particular, they assess its ability to learn statistical-temporal models from training data and to combine these models with background knowledge to improve the overall recognition accuracy. They use two Markov logic formulations for inferring the foreground activity as well as each activity's start and end times. Their primary goal is the derivation of concurrent activities. They define concurrent

activities as being in progress simultaneously but not necessarily involving the user's interaction at the same time steps. In other words, for any two or more interleaving activities, they distinguish between foreground and background activities. At each time step, only one activity can be identified as a foreground activity involving direct user interaction. However they have used datasets where every activity starts and ends only once.

An attraction of our system is that we return scenarios of activities instead of single activity occurrences. Moreover all the aforementioned systems do not recognize a second level of complex activities. Our system has the ability to recognize more abstract activities and also can tune the level of detail in the returned scenarios according to our preferences. Finally systems using MLNs have to deal with the bottleneck of obtaining large amounts of training data. The training dataset has to be large enough to build the complete models especially in cases of interleaved and concurrent activities (activities can be interleaved or performed concurrently in many ways and therefore it may be not possible to construct a complete model by training).

2.3.3 Hongeng et al.

Hongeng et al. presented in [15] an approach for both single- and multiple- actor activity recognition. An activity is considered to be composed of action threads, each thread being executed by a single actor. A single thread action is represented by a *stochastic finite automaton* of atomic activity states, which are recognized from the characteristics of the trajectory and shape of moving blob of the actor using *Bayesian methods*. A multi-agent activity is composed of several action threads related by temporal constraints. Multi-agent activities are recognized by propagating the constraints and likelihood of event threads in a temporal logic network.

Figure 2.7 shows schematically their approach to recognize the behavior of moving objects from an image sequence and available context. Context consists of associated information, other than the sensed data, that is useful for activity recognition such as a spatial map and prior activity expectation (task context).

Their system models scenarios using a hierarchical activity representation extended from [14]. Scenarios correspond to complex activities described by the classes of moving objects (e.g., human, car or suitcase) and the activity in which they are in-

volved. Both the class of an object and the activity have a confidence value attached to them, based on statistical analysis. They classify scenario events into a single thread and multiple thread activity. In a single thread activity, relevant actions occur along a linear time scale. Single thread activities are further categorized into a simple or complex activity. Simple activities are defined as a short coherent unit of movement (e.g., "approaching a reference person") and can be verified from a set of sub-activities ("getting closer to the reference person", "heading toward", etc.) and mobile object properties.

One disadvantage of their work is that further development of the temporal and logical constraints is required to represent a more sophisticated scenario model. For example, numerical constraints such as "A occurs before B at least an hour ago" or additional logical constraints such as "no other events should happen between A and B" are not available in the current system. Moreover they have high complexity as all scenarios of all possible combinations of moving objects and reference objects are analyzed instead of processing only scenarios relevant to the observations.

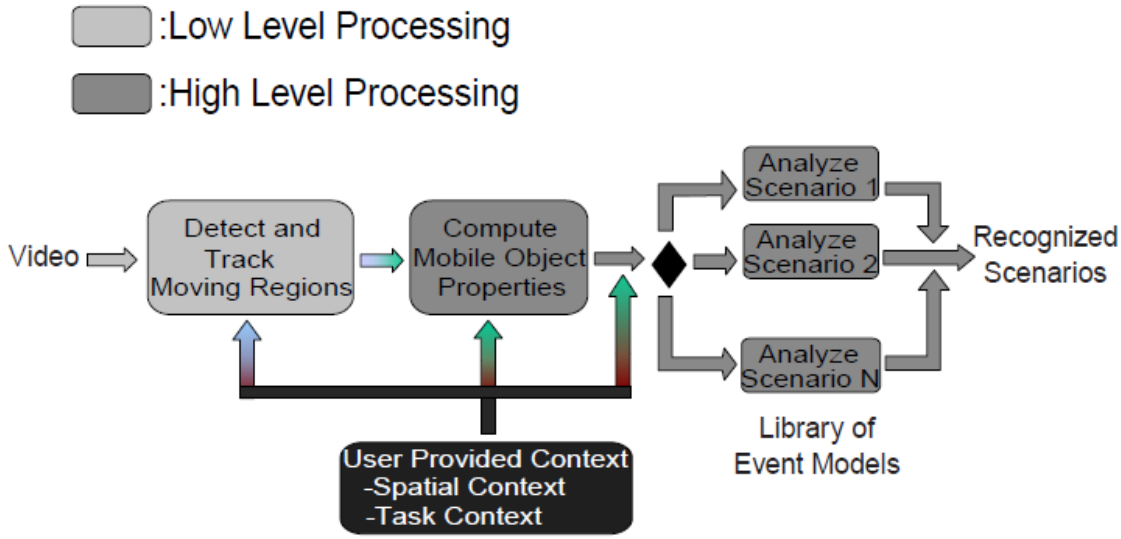

Figure 2.7 Overview of Hongeng's et al. system that is composed of two modules: 1) Motion detection and tracking (shown in blue); 2) Event Analysis (shown in green).

Another drawback of their approach is that they estimate the parameters of their model using supervised training i.e. a Bayesian formulation. The major difficulty with the Bayesian formulation lies in obtaining the *a priori* estimates, especially for unusual activities which are not part of a closed set. Moreover, in their model the states of the FSM's depend on absolute time clocks causing an explosion of possibilities for

state changes. Finally they return scenarios consisting only from one complex activity, while the returned scenarios by our system may contain several activities (related with each other or not).

2.4 Summary and Discussion

To sum up the main approaches in activity recognition frameworks are: logic-based, probabilistic-based and combinations of them. In probabilistic systems, such as [5], [6], [7] where HMMs, HHMMs and CRFs have been used respectively, noise and uncertainty are handled well. However they require training data and do not cope with conflicts or abstraction preferences. Logic-based systems on the other hand, do not deal with missing activities and noise [17], [19]. They also lack the ability to handle conflicts or adjust the desired level of detail in their results.

There are various approaches trying to combine logic with probabilities. Shet and his colleagues [20] proposed a system that uses logic programming (Prolog) for activity recognition. In [21] they extended their system with the bi-lattice framework to detect humans under partial occlusion, based on the output of parts based detectors. A problem of their work is that their temporal representation and reasoning is really poor, as they have no rules for computing the intervals in which a complex activity takes place.

There are some works that use Markov logic-type relational models for activity recognition based on video such as Tran and Davis [23] that used MLNs to probabilistically infer activities in a parking lot. Helaoui et al. [25] use MLNs as a declarative framework for recognizing interleaved and concurrent activities incorporating input from sensors and common-sense background knowledge. Finally in [24] Biswas et al. introduce a first-order probabilistic model that combines multiple clues to classify human activities from video data and is implemented as a Dynamic Markov Logic Network. The system is applied to an office setting where only activities that involve an agent and an inanimate object are considered (e.g. writing with a pen).

Hongeng et al. in their work, use Bayesian networks and probabilistic finite state automata to describe single actor activities, which are recognized from the characteristics of the trajectory and shape of the moving blob of the actor. The interaction of multiple actors is modeled by an activity graph, useful for plan and coordinated activ-

ity recognition. Further development of the temporal and logical constraints in their system is required to represent more sophisticated scenario models.

The main drawback in all the aforementioned approaches [21, 23-25] is that they do not handle conflicts. They return single activities instead of scenarios and therefore they may report activities that are logically inconsistent with each other. Furthermore unlike our system they can not adjust the level of detail in their results according to their preferences. While our system needs knowledge engineering, in these systems training data are needed which are not always easy to obtain.

3 Reasoning Framework

The activity recognition system has three levels. The first is the recognition of every possible complex activity, based on the atomic activity, using a set of predefined rules implemented in the Java Expert System Shell (JESS) [26]. The second is the detection of all the conflicts between the recognized complex activities (also implemented with JESS rules). The final step is conflict resolution and optimization of activity recognition according to preferences for the level of detail.

3.1 Identification of Plausible Occurrences of Activities

As a first step, the system identifies all plausible activities that may have taken place. The final scenario reported is a subset of these that maximizes preferences while also resolving all conflicts.

An *activity instance* (or simply, activity) is denoted by $E[t_1,t_2]_{cf}$, where E is a unique identifier of the activity, t_1 is its start time, t_2 is its end time and cf is the confidence value we have for this activity. Activity instances have the t_1, t_2, and cf values bound. An atomic activity E is defined as an instantaneous activity:

$$E\left[t_1, t_2\right]_{cf} \text{ is atomic activity} \Leftrightarrow t_1 = t_2 \quad (1)$$

The input to the system is a *dataset* of detected atomic activities; the parameters t_1, t_2, and cf are obtained by the sensors. Complex activities are constructed recursively from the atomic and lower-level activities (the atomic and complex activities types from the running example scenario from Section 1.4.2 are depicted in table 3.1). The construction of the complex activities is based on an event algebra over *activity types*: any pattern of activities that matches an algebra rule produces a new activity. The algebra operators define the time interval of the recognized new activity.

Table 3.1 The atomic and complex activity types from the running example scenario (Section 1.4.2)

Atomic Activity Types	**Complex Activity Types (1st level)**	**Complex Activity Types (2nd level)**
Lie On Bed	*User Is Resting On Bed*	*User Is Relaxing At Home*
Stand Up From Bed		
Turn Off Light		
Turn On Light		
Turn On Tv	*User Is Watching TV*	
Change Tv Volume		
Change Tv Channels		
Turn Off Tv		
Pick Up Telephone	*User Is Talking On Telephone*	
Close Telephone		
Light Brightened	*User Is Watching Slideshow*	
Light Dimmed		
Turn On Projector		
Change Slides		
Turn Off Projector		

3.1.1 Operators

The event algebra operators on which the construction of the complex activities is based are formalized as follows:

1. **Negation as failure NAF (not**): used to derive not *E* (i.e. that *E* is assumed not to hold) from failure to derive *E*. Any single pattern can be enclosed in a list with not as the head. In this case, the pattern is considered to match if a fact (or set of facts) which matches the pattern is not found.

2. **Disjunction operator ($\vee$):** at least one of the specified instances has to occur. Disjunction of two activities E_1 and E_2 occurs when E_1 occurs or E_2 occurs.

$$\left(E_1 \vee E_2 \left[t_s, t_e\right] \right)_{cf} \leftarrow E_1 \left[t_s, t_e\right]_{cf} \tag{2}$$

$$\left(E_1 \vee E_2 \left[t_s, t_e\right] \right)_{cf} \leftarrow E_2 \left[t_s, t_e\right]_{cf} \tag{3}$$

$$\left(E_1 \vee E_2 \left[t_s, t_e\right] \right)_{cf} \Leftrightarrow \left(E_2 \vee E_1 \left[t_s, t_e\right] \right)_{cf} \tag{4}$$

$$\mathrm{E}\left[\mathrm{t}_1, \mathrm{t}_4\right]_{min(cf1, cf2)} \Leftrightarrow E\left[\mathrm{t}_1, \mathrm{t}_2\right]_{cf1}, E\left[\mathrm{t}_3, \mathrm{t}_4\right]_{cf2} \quad t_1 \le t_3, t_3 \le t_2 \tag{5}$$

3. **Conjunction operator ($\wedge$):** the specified activity instances must occur at the same interval. Conjunction of two activities E_1 and E_2 occurs when both E_1 and E_2 occur, irrespective of their order of occurrence.

$$\left(E_1 \wedge E_2 \left[max(t_{1s}, t_{2s}), min(t_{1e}, t_{2e})\right] \right)_{min(cf1, cf2)} \leftarrow E_1\left[t_{1s}, t_{1e}\right]_{cf1}, \; E_2\left[t_{2s}, t_{2e}\right]_{cf2}, \; t_{1s} \le t_{2e}, t_{2s} \le t_{1e} \tag{6}$$

4. **Optional-activity operator (optional):** an optional activity still allows the recognition of higher-level activities that may depend on it, with smaller confidence: an activity is still recognized, if flagged as optional, with *0* confidence even if it never occurred.

$$optional\left(E[t_s,t_e]\right)_0 \leftarrow not\left(E[t_s,t_e]_{cf}\right) \tag{7}$$

$$optional\left(E[t_s,t_e]\right)_{cf} \leftarrow E[t_s,t_e]_{cf} \tag{8}$$

Table 3.2 The complex activity type definitions from the running example scenario (Section 1.4.2)

	Complex Activity Types Definitions
1.	$UserIsRestingOnBed \leftarrow set(LieOnBed, optional(TurnOffLight)); StandUpFromBed\ ; optional(TrunOnLight)$
2.	$UserIsWatchingTv \leftarrow TurnOnTv\ ; set\begin{pmatrix} optional(ChangeTvChannels), \\ optional(ChangeTvVolume) \end{pmatrix}; TurnOffTv$
3.	$UserIsTalkingOnTelephone \leftarrow PickUpTelephone\ ;\ CloseTelephone$
4.	$UserIsWatchingSlideshow \leftarrow set\begin{pmatrix} TurnOnProjector, \\ optional(LightDimmed) \end{pmatrix}; optional(ChangeSlides); TurnOffProjector\ ; optional(LightBrightened)$
5.	$UserIsRelaxingAtHome \leftarrow set\begin{pmatrix} optional\ (UserIsRestingOnBed), \\ optional\ (UserIsWatchingTv), \\ optional\ (UserIsTalkingOnTelephone), \end{pmatrix}$

5. **Sequence operator (;):** the activity *(E_1 ; E_2)* is recognized when E_1 and E_2 occur in this order. The activities have to follow each other within at most w time-units from each other. This precludes the situation the set is recognized from activities separated by an arbitrarily long time interval.

$$\left((E_1; E_2)\ [t_{1s}, t_{2e}] \mid w \right)_{cf} \leftarrow E_1[t_{1s}, t_{1e}]_{cf1},\ E_2[t_{2s}, t_{2e}]_{cf2},\ t_{1e} < t_{2s}, (t_{2s} < t_{1e} + w) \tag{9}$$

$$\left((E_1 ; optional(E_2))\ [t_{1s}, t_{1e}] \mid w \right)_{cfa} \leftarrow E_1[t_{1s}, t_{1e}]_{cf1}, optional(E_2[t_{2s}, t_{2e}])_0,\ t_{1e} < t_{2s}, (t_{2s} < t_{1e} + w) \tag{10}$$

$$\left((E_1 ; optional(E_2))\ [t_{1s}, t_{2e}] \mid w \right)_{cfb} \leftarrow E1[t_{1s}, t_{1e}]_{cf1}, optional(E2[t_{2s}, t_{2e}])_{cf2},\ t_{1e} < t_{2s}, (t_{2s} < t_{1e} + w) \tag{11}$$

where *cfa* <*cfb*.

6. **Set operator (set):** the activity *set(E_1 , E_2)* is recognized when both E_1 and E_2 occur in any order. The activities have to follow each other within at most *w* time-units from each other.

$$set\left((E_1, E_2)[min(t_{1s}, t_{2s}), max(t_{1e}, t_{2e})] \mid w \right)_{cf} \leftarrow E_1[t_{1s}, t_{1e}]_{cf1},\ E_2[t_{2s}, t_{2e}]_{cf2},\ max(t_{1s}, t_{2s}) < min(t_{1e}, t_{2e}) + w \tag{12}$$

$$set\left(E_1, optional(E_2)[t_{1s}, t_{1e}] \mid w \right)_{cfa} \leftarrow E_1[t_{1s}, t_{1e}]_{cf1},\ optional(E_2[t_{2s}, t_{2e}])_0,\ max(t_{1s}, t_{2s}) < min(t_{1e}, t_{2e}) + w \tag{13}$$

$$set(\ (E_1, optional(E_2))[min(t_{1s}, t_{2s}), max(t_{1e}, t_{2e})] \mid w)_{\ cfb} \leftarrow E_1[t_{1s}, t_{1e}]_{\ cf1}\ ,\ optional(E_2[t_{2s}, t_{2e}])_{\ cf2}\ ,\ max(t_{1s}, t_{2s}) < min(t_{1e}, t_{2e}) + w \quad (14)$$

The confidence values *cf* of the complex activities are computed as follows. For a first level activity, the *cf* is sum of the confidences of its atomic activities normalized to [0, 100], where 100 is achieved when all activities, including optional ones, are recognized with the highest confidence. For a second and higher level activity, the *cf* is the sum of the *cf* of the component lower-level activities. Given this definition, a second level activity (E_1 ; E_2) is equally preferable (same *cf*) to recognizing E_1 and E_2 as distinct activities, everything else being equal.

The definitions of the complex activity types from our running example scenario (from section 1.4.2), based on the previous operators can be seen in Table 3.2. For simplicity we have omitted the representation of activities time intervals and confidence values. Note that the complex activity "User is relaxing at home" is a second level complex activity as it is recognized based on other complex activities.

3.2 Conflict Detection

Obviously there are some pairs of activities that a user cannot perform at the same time e.g. "User is relaxing at home" and "User is watching slideshow". As a result an activity recognition system has to detect these conflicts, and select to report a sequence of recognized activities that is logically consistent. One way to detect conflicts is to define conflicting pairs of activity types, e.g., relaxing vs. working. However, this approach complicates knowledge engineering: whenever a new type is defined, all conflicting predefined types should be declared.

Instead, we follow an approach based on the concept of *activity resources* that facilitates knowledge representation. For each activity type a list of activity resources is specified. Resources can be split into tangible and intangible. Tangible resources are those which have actual physical existence e.g. a chair, whereas intangible resources are those that are present but cannot be grasped or contained e.g. user's attention.

We consider that two complex activities are in conflict, if their time-intervals overlap and they use common resources. The formal definition for the conflict detection between two recognized complex activities $E_1[t_{1s}, t_{1e}]_{cf1}$ and $E_2[t_{2s}, t_{2e}]_{cf2}$ is:

$$conflict(E_1, E_2) \Leftrightarrow t_{1s} \le t_{2e},\ t_{2s} \le t_{1e},\ \begin{pmatrix} (\exists r : r \in resources(E_1), r \in resources(E_2)) \vee \\ \begin{pmatrix} \exists E_n, E_k : E_n \in usedActivities(E_1),\ E_k \in usedActivities(E_2), \\ conflict(E_n, E_k) \end{pmatrix} \end{pmatrix} \quad (15)$$

Where:

- *resources(E_i):* the set of resources complex activity $E_i\,[t_{is}, t_{ie}]_{cfi}$ uses.
- *usedActivities(E_i):* the set of activities we used to recognize complex activity $E_i[t_{is}, t_{ie}]_{cfi}$.
- *r*: a resource that both complex activities use.

3.3 Conflict Resolution and Preference Optimization

Let us denote with B_i a propositional (binary) variable denoting whether a recognized activity E_i is selected in the final output. When two activities E_i and E_j are conflicting, only one of them should be selected for the returned scenario. This statement corresponds to the constraint $(\neg B_i \vee \neg B_j)$ on the propositional variables. Thus, resolving all conflicts is equivalent to solving a satisfiability problem (SAT) of the form:

$$(\neg B_k \vee \neg B_m) \wedge \ldots \wedge (\neg B_i \vee \neg B_j) \quad (16)$$

where a clause is included in the formula for each identified conflict. Unfortunately, this problem has the trivial solution of setting all B_i to false, thus not returning any activities and avoiding all conflicts. Instead, one would like to give preference to solutions recognizing as many activities as possible, or even better, high-confidence activities that "explain" a large percentage of user's time and atomic activities.

To obtain optimal solutions we convert instead to a Weighted Partial MaxSAT problem, which is a generalization of the SAT problem. In a Weighted Partial MaxSAT problem some clauses are specified as hard constraints (must be satisfied), while other ones are soft constraints (desirable to be satisfied). In the soft constraints,

weights are assigned. Every weight represents the penalty to falsify the clause. Optimal solution for a Weighted Partial MaxSAT instance is an assignment such that satisfies all the hard clauses, and the sum of the weights of the falsified soft clauses is minimal. We used Sat4j, an open source library of SAT-solvers [27]. Specifically, for each plausible activity E_i (Section 3.1) we define the following:

- B_i *:* a binary variable denoting the selection of E_i in the output
- $D(E_i)$: the temporal duration of E_i
- $C(E_i)$: the confidence of E_i
- $A(E_i)$: the number of atomic activities we used to recognize (explained-by) E_i

For each conflict between E_i and E_j we create the clause $(\neg B_i \vee \neg B_j)$ as a hard constraint. For each activity E_i, we create the singleton clause B_i as a soft constraint. The weight given to B_i is

$$w_i = a \cdot D(E_i) + b \cdot C(E_i) + c \cdot A(E_i) \tag{17}$$

where $a, b, c > 0$ are preference parameters. Thus, the preference given to selecting E_i increases with its confidence factor, the number of atomic activities it encompasses, and its temporal duration. The relative weight of these factors depends on the preference parameters. Thus, the Weighted Partial MaxSAT solves the following optimization problem:

$$\max_{B_1 \cdots B_n} \sum_{i=1}^{n} w_i \cdot B_i \tag{18}$$
$$\text{s.t., all conflicts are resolved}$$

So with the above optimization problem we want to get a set of recognized complex activities that are:

- ✓ as many as possible
- ✓ with high-confidence
- ✓ "explain" a large percentage of the user's time
- ✓ "explain" a large percentage of detected atomic activities.

3.4 System's Architecture

The system is composed by three modules. The architecture of the system can be seen in figure 3.1. In the core of the system is the *Main Module* that ensures the communication and the information flow between the two other modules. It is a Java program that runs the JESS rules and then runs the weighted partial MaxSAT solver. The input of this module are the atomic activities and their relevant information (time of occurrence, actor etc.).

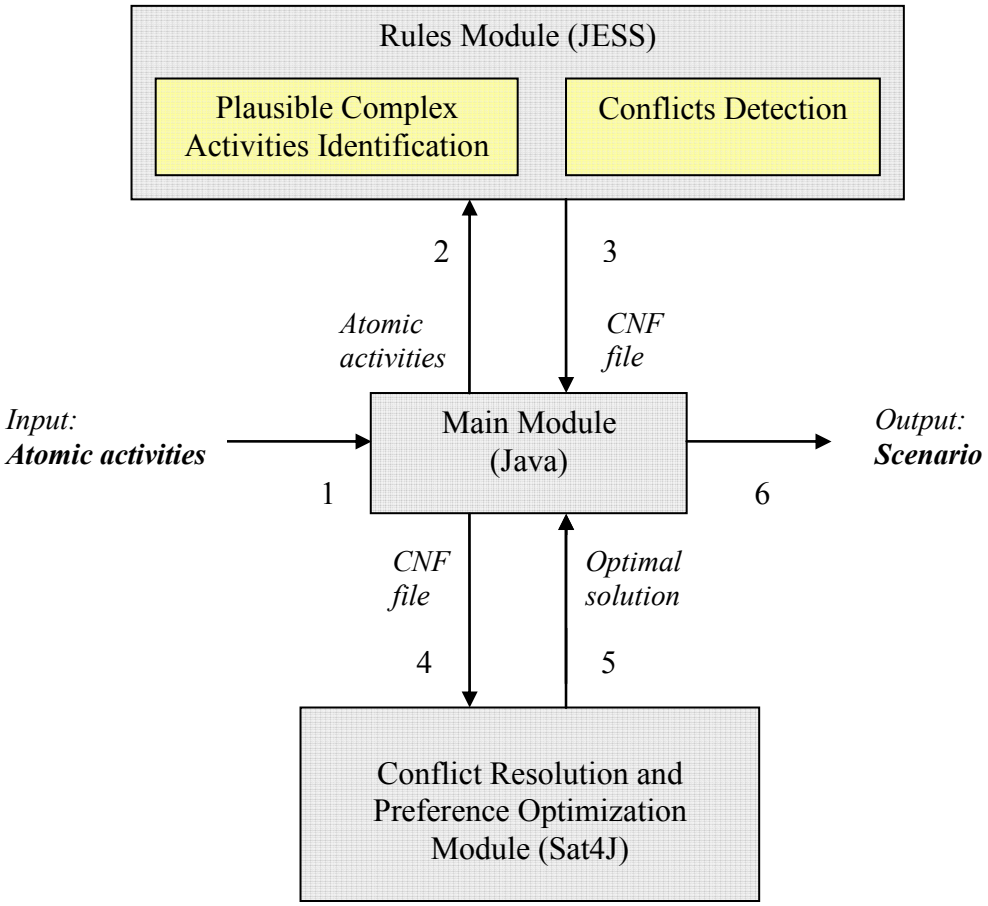

Figure 3.1 The architecture of our system.

The second component of the system is the *Rules Module* which is a JESS program. Here the atomic activities are asserted in the working memory of JESS. Then the rules that identify every plausible complex activity, and detect all the conflicts between them are activated (sections 3.1 and 3.2). The identified plausible complex activities and their conflicts are transformed into conjunctive normal form (CNF) [42].

The third component is the *Conflict Resolution and Preference Optimization Module*. Here the weighted partial MaxSAT solver is trying to solve the problem given in the created CNF file from the *Rules Module*. The result of the satisfiability process is returned in the *Main Module* where it is transformed into a user-friendlier format and presented to the user. As we explained in section 3.3 the output of the system is a set of complex activities (scenario) adjusted to the preferences the user has given to the system.

4 Background Information

4.1 Java Expert System Shell (JESS)

We express the core of our system in rules written in the Java Expert System Shell (JESS). Both the identification of every plausible complex activity, based on the detected atomic activities and the detection of all the conflicts between those complex activities are implemented with a set of predefined rules in JESS [26].

JESS is a rule engine and scripting language developed by Ernest Friedman-Hill in the late 1990s. It is written in Java, so it is an ideal tool for adding rules technology to Java-based software systems. The original inspiration for JESS was CLIPS expert system shell, an open-source rule engine written in C. However JESS and CLIPS implementations are very different. JESS is dynamic and Java-centric, so it gives access to all of Java's APIs for networking, graphics, database access, etc. On the other hand CLIPS has none of these features [28].

JESS stores facts in its working memory and then decisions are made by running rules against the facts currently in it. Users can define their *deftemplates*, which specify the structure of facts, and *deffacts* which assert a series of *facts* (instances of the deftemplates) to working memory. A deftemplate describes a kind of fact, in the same way as a Java class describes a kind of object. A deftemplate lists the *slots* i.e. the attributes that this particular kind of fact can have. JESS also allows users to define functions (*deffunction*s) either directly in JESS or as Java classes written externally to JESS, which can be called using the standard JESS syntax.

Finally *defrules* are the rules defined by the user. They may contain function calls that manipulate the fact base and/or other Java code. A defrule consists of a left-hand side, the symbol "=>", and a right-hand side, in this specific order. The left-hand side is made up of zero or more *conditional elements*, while the right-hand side consists of zero or more function calls. A conditional element is either a *pattern*, or a grouping construct like "and", "or", or "not." The conditional elements are matched against

Jess's working memory. When they all match, and if the engine is running, the code on the rule's right-hand side will be executed [26]. The syntax of a defrule is the following:

$$\begin{aligned} &(defrule\ rule-name \\ &\quad [" Documentation\ comment"] \\ &\quad conditiona\ l\ element^{*} \\ &\quad \Rightarrow \\ &\quad funtion\ \ call^{*}\ ) \end{aligned} \tag{19}$$

We have defined with deftemplates atomic and complex activities. In the specified slots all the information needed about the activities is kept. The deftemplates of the atomic and complex activities can be seen in Appendix A1. The initial facts of the system are the atomic activities we have detected. Then with our activity recognition rules, complex activities and the conflicts between them are found and asserted as facts into the working memory of JESS.

4.1.1 Declarative Programming

Rule-based programming is not procedural but declarative. In *procedural programming*, the programmer tells the computer what to do, how to do it, and in what order. Procedural programming is well suited for problems in which the inputs are well specified and for which a known set of steps can be executed in order to solve the problem.

On the other hand in purely *declarative programming*, the programmer describes what the computer should accomplish, rather than describing how to go about accomplishing it. Declarative programs have to be executed by a runtime system that understands how to fill in the blanks and use the declarative information to solve problems. Because declarative programs include only the important points of a solution, they can be easier to understand than procedural programs. Moreover, a declarative program can flexibly manage incomplete input data, as the control flow is chosen by the runtime system. Declarative programming is useful when dealing with problems involving control, diagnosis, prediction, classification or pattern recognition [28].

In a rule-based program, only the individual rules have to be written. Then the rule engine determines which rules apply at any given time and executes them as appropriate. Therefore, a rule-based version of a complex program can be shorter and easier to understand than its procedural solution. It is simpler for the programmer to write the necessary rules for one situation at a time, instead of trying to find the whole solution at once. Finally in many cases the modification of a rule-based program is easier than in an equivalent procedural program.

Rules are similar to the *if-then* statements of programming languages. The *if* part of a rule is called its *left-hand side* (*LHS*) of the rule, or *premises*. The *then* part is called the *right hand side* (*RHS*) of the rule, or *conclusions*. Thus a *rule-based system* is a system that uses *rules* to derive *conclusions* from *premises*.

A typical rule engine such as JESS contains the following components (shown schematically in figure 4.1):

1. **A rule base:** contains all the rules of the system. They may be stored as strings of text, but most often a rule compiler processes them into some form that is more efficiently managed by the inference engine.

2. **A working memory:** also called the *fact base*, contains all the pieces of information the rule-based system is working with. The working memory can hold both the premises and the conclusions of the rules.

3. **An inference engine:** controls the whole process of applying the rules to the working memory to obtain the results of the system. It consists of :
    a. **A pattern matcher:** decides which rules apply, based on the current contents of the working memory.
    b. **An agenda:** decides based on the conflict strategy of the system which of the rules, out of all those that apply, have the highest priority and should be fired first.
    c. **An execution engine:** is the component of a rule engine that fires the rules.

The pattern-matcher applies the rules of the rule-base to the facts in working memory to construct the agenda. Then the inference engine fires the rules from the agenda. This changes the contents of working memory and restarts the process.

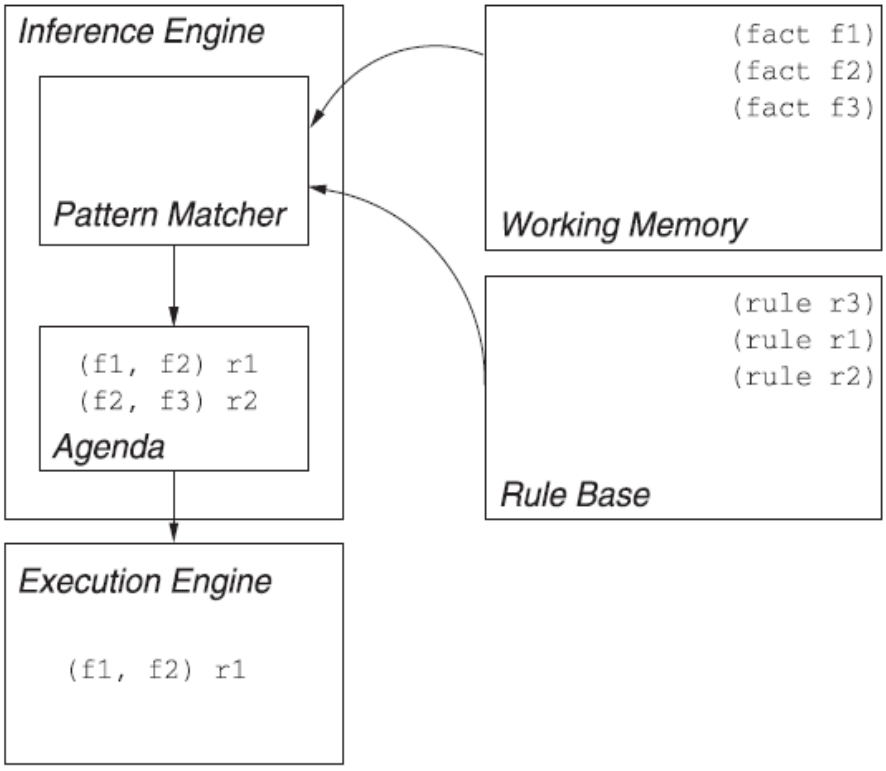

Figure 4.1 The architecture of a typical rule-based system such as JESS.

4.1.2 Rete Algorithm

The *Rete* algorithm which is fast and efficient is being used for pattern matching by JESS. The main advantage of Rete is that it uses a set of memories to retain information about the success or failure of pattern matches during previous cycles. This is achieved through the creation of a network of pattern-matching nodes.

Usually rule-based systems have a fixed set of rules, and a working memory that changes continuously. The naive implementation of pattern matching works as follows: A list of the rules is kept by the system and each one's LHS is checked in turn against the working memory, forming a set of activation records with the matching rules. After choosing one rule and executing it, the set of activation records could be discarded and the process would be repeated [30]. But this is not the optimal solution

as it does not scale up well. After every rule firing, the system must recheck every fact against every rule. For example doubling the number of facts or the number of rules roughly halves the performance of the system [28].

However, in most rule-based systems, a large part of the working memory is almost fixed over time. New facts are inserted in the working memory and old ones are removed as the system runs, but the percentage of facts that change per unit time is generally quite small. The naive algorithm is inefficient, because most of the tests made on each iteration are unnecessary as they will have the same results as on the previous cycle. An algorithm that could remember previous pattern-matching results between cycles, only updating matches for facts that actually changed, could get the same results with far less work [28].

This knowledge is used by the Rete algorithm to eliminate the inefficiency of the naive pattern matcher. It remembers previous test results across iterations on the rules and only new working memory elements are tested against the rules at each iteration. Furthermore, Rete organizes the pattern matcher so that these few facts are only tested against the subset of the rules that may actually match.

More precisely a Rete-based system builds a network of nodes, where every node (except of the root) corresponds to a pattern occurring in the LHS of a rule. The path from the root node to a leaf node defines a complete rule's LHS. Each node has a memory of facts which satisfy that pattern. As new facts are asserted or modified, they propagate along the network, causing nodes to be annotated when that fact matches that pattern. Whenever one or more facts satisfy all of the patterns for a rule, a leaf node is reached and the corresponding rule is triggered [30].

4.1.3 JESS Performance

As explained in the previous section JESS uses an improved form of an algorithm called Rete to match rules against the working memory. Rete is an algorithm that explicitly *trades space for speed*, so JESS' memory usage[2] is not inconsiderable. In general, JESS will benefit if the heap size and the object nursery size are each set larger than the default.

[2] JESS does contain some commands which will allow you to sacrifice some performance to decrease memory usage.

The naïve algorithm described in the previous section has performance of the order of:

$$O\left(R \cdot F^{P}\right) \tag{20}$$

Where:

- R is the number of rules
- F is the number of facts on the working memory
- P is the average number of patterns per rule LHS

This escalates dramatically as the number of patterns per rule increases.

Rete's performance is harder to analyze precisely than the naive algorithm. Generally, the performance on the first cycle is basically the same for the two algorithms. Rete has to do pattern matching for every fact in working memory, because there are no previous results to consult. In the worst case, where every fact changes on every cycle and there is no network sharing between rules, the performance for later cycles is the same as well.

So JESS won't be very efficient if the working memory is populated, the pattern matcher is ran for just one cycle, and then the working memory is reset and created again. But in the typical case of a slowly changing working memory, moderate sharing in the network, and effective indexing, Rete will outperform the naive algorithm for all cycles after the first.

In the Rete algorithm, only new facts are tested against any rule LHSs. Additionally new facts are tested against only the rule LHSs to which they are most likely to be relevant. As a result, the *computational complexity per iteration* drops to something more like:

$$O\left(R \cdot F \cdot P\right) \tag{21}$$

or linear in the size of working memory [31].

According to Dr. Friedman-Hill [28] the *runtime* will be proportional to something like:

$$O\left(R' \cdot F'^{P'}\right) \tag{22}$$

Where:

- R' is a number less than R, the number of rules
- F' is the number of facts that change on each iteration
- P' is a number greater than one, but less than the average number of patterns per rule

Of course the analysis depends also on the rules the programmer writes and the data he uses. The "worst case" complexity happens only when the worst possible rules are written and the worst possible data and the worst possible implementation of Rete are used. At least the first of these factors is completely under the programmer's control, so it can be avoided.

Gilman et al. implemented an application with three reasoning engines, based on CLIPS [35], JESS and Win-Prolog [36]. Their application is a pervasive system that utilizes RFID technology to store and pick digital content with mobile phones and play it with smart space's displays and speakers. For their application all three engines have demonstrated close performance results. Jess performed slightly better. However, in pure reasoning Win-Prolog showed the best results. Jess exploits much more memory than other engines but less CPU time.

4.1.4 JESS Expressiveness

In his article Wolfgang Laun [29] explains how the objects of JESS relate to more theoretical concepts that are encountered in topics as Propositional Logic or First Order Predicate Calculus.

First of all a *proposition* is a statement of some alleged fact which is either true or false. The relation between propositions (as they are used in Propositional Logic) and JESS facts, is simple: *any proposition can be represented as a JESS fact*.

In the process of reasoning about propositions, more expressive power is provided by predicates. A *predicate* states a specific property of an object, or lets us conveniently express relations between two or more objects. More interesting constructs are obtained by quantification, using the symbols $\forall$ ("for all") and $\exists$ ("exists"). This creates quantified propositions, with the first one stating that the proposition P holds for all objects x, and the second that proposition P holds for at least one object x.

Quantification and other properties of first-order logic are expressed through the condition elements (CEs) of JESS. CEs are a way to express more complex relationships between facts, and to qualify the matches for entire facts. Moreover CEs are pattern modifiers. They can group patterns into logical structures, and they can say something about the meaning of a match. The following are the most interesting CE of JESS:

1. "***EXISTS***": Matches if at least one fact matches. An *exists* CE is true if there exist any facts that match the pattern, and false otherwise. *exists* is useful when a rule should fire only once, although there may be many facts that could potentially activate it:

```
(defrule exists-demo
        (exists (honest ?))
        =>
        (printout t "There is at least one honest man!" crlf) )
```

2. "***FORALL***": Matches if, for every match of the first pattern inside it, all the subsequent patterns match:

```
(defrule every-employee-has-a-stapler-and-holepunch
        ( forall (employee (name ?n)) )
        (stapler (owner ?n))
        (holepunch (owner ?n))
        =>
        (printout t "Every employee has a stapler and a holepunch." crlf) )
```

3. ***"AND":*** Matches multiple facts. Any number of patterns can be enclosed in a list with *and* as the head. The resulting pattern is matched if and only if all of the enclosed patterns are matched. By themselves, *and* groups are not very interesting, but combined with *or* and *not* CEs, they can be used to construct complex logical conditions. The entire LHS of every rule and query is implicitly enclosed in an *and* conditional element. The following rule matches only if *(flaps-up)* and *(en-*

gine-on) both match (of course the rule would behave precisely the same way if the *and* CE was omitted):

```
(defrule ready-to-fly
        (and (flaps-up)
             (engine-on))
        =>
        (printout t "Ready to fly!" crlf) )
```

4. "***OR***": Matches alternative facts. Any number of patterns can be enclosed in a list with *or* as the head. The *or* CE matches if one or more of the patterns inside it matches. If more than one of the patterns inside the or matches, the entire *or* is matched more than once:

```
(defrule prepare-sandwitch
        (or (mustard)
            (mayo))
        =>
        (printout t "Preparing sandwitch" crlf))
```

5. "***NOT***": Matches if no facts match. Any single pattern can be enclosed in a list with *not* as the head. In this case, the pattern is considered to match if a fact (or set of facts) which matches the pattern is not found. For example:

```
(defrule not-married
        (person ?x)
        (not(married ?x))
        =>
        (printout t ?x "is not married." crlf) )
```

4.1.5 JESS and Uncertainty

One of the first Expert Systems incorporating uncertainty was MYCIN [33]. Both the properties of the modelled objects and the rules are associated with uncertainty factors. The definition of Certainty Factors (CF) in MYCIN are the following:

- Range: $-1 \leq CF \leq +1$
- CF level definitions:
  - $CF = +1$ the fact or rule is certainly true
  - $CF = 0$ we know nothing about whether the fact or rule is true or not
  - $CF = -1$ the fact or rule is certainly not true

When the user inputs values he is asked to specify his confidence about them. MYCIN has a procedure for calculating the CF for a conclusion of a rule on the basis of the strength of belief in the rule itself and in the various data inputs:

$$CF_{conclusion} = CF_{rule} \cdot CF_{data} \tag{23}$$

$$CF_{data} = min\left(CF_{d1}, CF_{d2}, \ldots, CF_{dn}\right) \tag{24}$$

Where CF_{di} are the CFs for the several data inputs.

MYCIN has also a mechanism to calculate the cumulative CF for a conclusion reached by several rules. Suppose a rule reports a conclusion with certainty CF_p and another rule reaches the same conclusion with certainty CF_n. How the combined CF (CF_{comb}) is calculated depends on the signs of CF_p and CF_n:

$$\begin{aligned} &If\ \left(CF_p > 0 \wedge CF_n > 0\right)\ then\ CF_{comb} = CF_p + CF_n - CF_p * CF_n \\ &If\ \left(CF_p < 0 \wedge CF_n < 0\right)\ then\ CF_{comb} = CF_p + CF_n + CF_p * CF_n \\ &Else\ CF_{comb} = \frac{CF_p + CF_n}{1 - min\left(\left|CF_p\right|, \left|CF_n\right|\right)} \end{aligned} \tag{25}$$

So the overall algorithm for calculating the CF for a conclusion is:

1. Follow up all conclusions regardless of their Certainty Factors
2. The first chain that provides support cannot be simply chosen, as the next could give more (or less) confidence in the result.
3. The stopping condition is when the complete space has been searched *OR* they find a confidence level of *1.0 OR -1.0*.

Corsar et al. [32] have developed *Uncertainty Jess*, a system that provides Jess with the same uncertainty handling as MYCIN. Uncertainty Jess allows the user to assign certainty factors / scores to both the properties of their data and to the rules, which it then makes use of to determine the certainty of rule conclusions for single and multiple identical conclusions.

Using Uncertainty Jess requires four changes:

(a) The deftemplates for the facts need to include a slot in which the certainty factor can be saved.

(b) The rules need to be modified to generate a certainty factor for the classification.

(c) Another rule is required which detects multiple identical conclusions and calls a function to calculate the certainty factor of multiple identical conclusions.

(d) A function would be useful for calculating the combined certainty factor of multiple conclusions.

An alternative method for dealing with uncertain knowledge in Jess is to use the *FuzzyJess Extension* [34]. Fuzziness occurs when the boundary of a piece of information is not clear-cut. For example, words such as good, and high are fuzzy. FuzzyJess is part of the FuzzyJ toolkit and is built upon the concept of fuzzy logic and fuzzy set theory whereby knowledge is described in terms of its degree of membership of a fuzzy set. The membership of an element in a set can be partial—that is, an element belongs to a set with a certain grade (or degree, or possibility) of membership. This grade of membership in the fuzzy set is usually represented by values from 0 to 1 (with 0 meaning definitely not a member of the set and 1 meaning definitely is a member of the set).

FuzzyJess deals with three main concepts: *fuzzy variables*, *fuzzy sets* and *fuzzy values*. A fuzzy variable defines the basic components used to describe a fuzzy concept, providing the name of the variable, the units of the variable (if required), and the variables' universe of discourse. A fuzzy set defines a mapping of a real number onto a membership value in the range zero to one. A fuzzy value is an association of a fuzzy variable, a fuzzy set and a linguistic expression which describes the fuzzy variable. If a fuzzy value falls within the boundaries of the fuzzy set it is said to be a member of that set [28].

FuzzyJess requires the user to define fuzzy variable for every uncertain concept, along with a series of fuzzy values which describe the various fuzzy concepts related to a fuzzy variable, by defining a numeric range and natural language term for each fuzzy value. It also requires that the rules specify which fuzzy values the uncertainty associated with a piece of data (a fact) must match (as determined by the fuzzy-match function), in order to provide some conclusion.

Essentially, the Uncertainty Jess approach involves the addition of one extra slot in the deftemplates of atomic and complex activities, to represent the data CF, and the addition of a CF to each rule. Though using the Uncertainty Jess system is considerably less complex for the domain expert. Additionally the procedures needed to calculate combined uncertainties are considerably simpler and more succinct than the various counterparts in FuzzyJess [32].

Our system handles noise and uncertainty, with the use of optional activities and confidence factors in its facts (atomic and complex activities) as described in section 3.

4.2 Java with JESS

Arguably, one of the most powerful features of JESS is that it can be easily integrated with Java. From Java code, you can access all parts of the JESS library, so that it's very easy to embed JESS in any Java application, applet, or other system. Likewise, from the JESS language, the full power of Java is directly available. This capability is shown schematically in figure 4.2.

JESS can create Java objects, call their methods, and access their data. JESS can also work with Java primitives by converting between Java and Java types. With a few exceptions (particularly working with large or multidimensional arrays), most Java code can be directly translated to JESS. Even these exceptions can easily be overcome by extending JESS with functions written in Java [28].

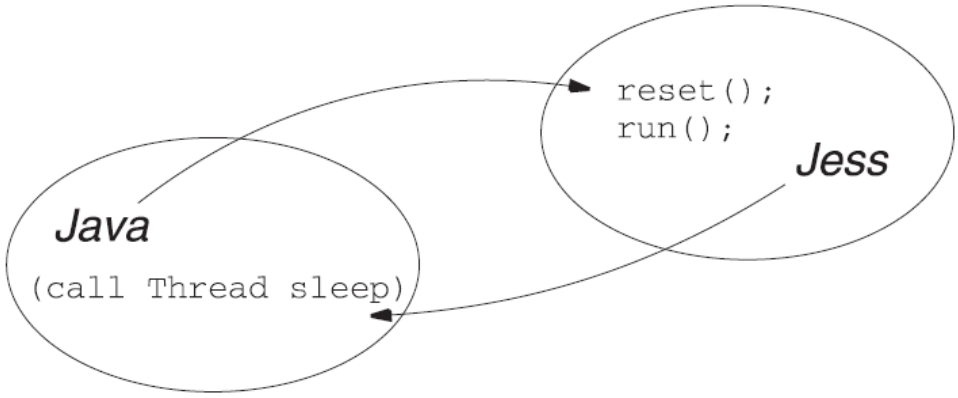

Figure 4.2 You can use JESS from Java, and call Java methods from JESS.

In our system we run JESS through a Java program that provides the JESS program with the necessary input (atomic activities). Then the output produced of the rules activation is given to the next component of the system (the weighted partial MaxSAT solver). So this Java module ensures the communication and the information flow between the different modules of the system.

4.3 Sat4J

Conflict resolution and optimization of activity recognition according to preferences for the level of detail are implemented with Sat4J. As explained in section 3.3 activity conflicts are formulated as a Weighted Partial MaxSAT problem, such that any solution to it corresponds to a set of identified activities (scenario) that do not conflict. Moreover through the Weighted Partial MaxSAT we have the ability to adjust the desired level of detail of the scenarios returned. This is accomplished by assigning preferences to clauses of the Weighted Partial MaxSAT problem.

Sat4j [27] is an open source library of SAT-based solvers in Java. It provides a modular SAT solver architecture designed to work with generic constraints. Such architecture is used to provide SAT, MaxSat and pseudo-boolean and solvers for light-weight constraint programming. Those solvers have been evaluated regularly in the corresponding international competitive events.

Sat4j is developed using both Java and open source standards. The project is supported by the OW2 consortium infrastructure and is released under both the EPL and the GNU LGPL licenses. SAT4J includes an implementation in Java of Niklas Een and Niklas Sorenson's MiniSAT specification [40]. Although the overall algorithmic of the solver is respected, the design has been adapted to Java practices.

A Weighted Partial MaxSAT problem is a generalization of the satisfiability problem. The basic idea is that often not all restrictions of a problem can be satisfied, so the instance is divided in the restrictions (clauses) that must be satisfied (hard), and the ones that may or may not be satisfied (soft). In the soft clauses, different weights are assigned that express the penalty to falsify the clause. The addition of weights to clauses makes the instance weighted, and the separation into hard and soft clauses makes the instance partial. Optimal solution for a weighted partial MaxSAT instance is an assignment such that satisfies all the hard clauses, and the sum of the weights of the falsified clauses is minimal.

The weighted partial MaxSAT problem can be used in many domains, e.g. combinatorial auctions, scheduling and timetabling problems. However, state-of-the-art solvers have not yet experienced the same success as SAT solvers for the Satisfiability problem in the industrial field. The MaxSAT problem is NP-hard, since its solution easily leads to the solution of the boolean satisfiability problem, which is NP-complete. In particular $O\left(NP^{log}\right)$ for unweighted, and $O\left(NP^{NP}\right)$ for weighted [41]. The aim of SAT4J is to produce efficient solvers to handle real and industrial problems that although generally big in size, are not as hard as the worst case instances [27].

The idea behind SAT4J is to add a new variable per weighted soft clause that represents that such clause has been violated, and to translate the maximization problem on those weighted soft clauses into a minimization problem on a linear function over those variables. Formally, suppose $T = \left\{ C_1^{w1}, C_2^{w2}, \ldots, C_n^{wn} \right\}$ is the original set of weighted clauses of the Weighted Partial MaxSAT problem. They translate that

problem into $T' = \{ s_1 \vee C_1, s_2 \vee C_2, \ldots, s_n \vee C_n \}$ plus the objective function $min : \sum_{i=1}^{n} w_i \cdot s_i$ [27]. The new selector variables that are needed are less than the clauses in the original problem as in the following cases no new variable is needed:

- ✓ **Hard clauses:** since they always must be satisfied. They can be treated "as is" by the SAT solver.

- ✓ **Unit soft clauses:** those constraints are violated when its literal is falsified. Therefore that literal is considered only in the optimization function, so no new selector variable is needed. In that case, the optimization function should minimize $\sum_{i=1}^{n} w_i \cdot \bar{l}_i$ where l_i is the literal in the unit clause C_i.

Since the underlying SAT solver is tailored to solve application benchmarks, the SAT4J approach performed poorly on randomly generated partial weighted MaxSAT problems, but provided good results on industrial classes of application benchmarks during the MaxSAT evaluation 2009 [27].

5 Experiments

We have implemented the recognition system and integrated it within a real ambient intelligence (AmI) environment (ICS-FORTH AmI Facility). Moreover we ran simulation studies to test how robust is the system when processing noisy inputs.

5.1 ICS-FORTH AmI Facility

Over the last years a long-term AmI Research and Technical Development (RTD) Programme [37] has been initiated by the Institute of Computer Science of the Foundation for Research and Technology – Hellas (ICS-FORTH). The goal is to develop innovative human-centric AmI technologies and Smart Environments.

Based on innovative technologies and their related expertise, the various laboratories of ICS-FORTH, are currently working towards the development of several interoperating AmI components [37], such as:

- ✓ AmI software and hardware architectures
- ✓ Middleware
- ✓ Computer vision subsystem for multiple user localization and gesture recognition
- ✓ Speech recognition and speaker localization
- ✓ Environment sensing technologies and sensor fusion
- ✓ Dynamic surround sound playing system
- ✓ Environmental control
- ✓ Context management and reasoning
- ✓ Access control, information and communications security
- ✓ Seamless and intuitive user-environment interaction
- ✓ Activity recognition

As described in [38] three related distinct spaces have been set up in the premises of ICS-FORTH. Also a large-scale AmI facility is currently under construction:

1. **AmI Sandbox**: is a laboratory space of about 100m^2 comprising six rooms. Researchers have in this setting the opportunity to share their knowledge and resources in order to obtain practical experience. Also they can experiment in a highly flexible setting. The AmI Sandbox is depicted in figure 5.1. In this space, several AmI technologies and applications are installed, integrated and demonstrated, and multiple ideas and solutions are cooperatively developed, studied and tested. More specifically the following technologies have already been installed in the AmI Sandbox [37]:
    - Computer vision system, comprising 8 cameras
    - Surround speaker system with 8 speakers
    - Various computer-operated lights (neon, spot lights, floor and desk lamps) using both the DMX and X10 protocols
    - Computer-operated air-condition
    - Various screens and high definition TVs, including touch screens
    - One large front projection screen created by 2 ceiling-mounted short-throw projectors
    - One back projection screen
    - Several sensors (distance, temperature, etc.) and actuators
    - Desktop and mobile RFID readers
    - An interactive table
    - Access control systems
    - Positioning system through wireless access points
    - Various robotic systems

Figure 5.1 AmI Sandbox computer room (top); AmI Sandbox main room (bottom).

2. **Smart office prototype:** is a prototype environment exhibiting the concept of Ambient Intelligence in a typical office environment (figure 5.2). The way our everyday life and activities are going to change is demonstrated through this prototype. Also the way people are going to perceive and interact with information and communication technologies in the future is tested [39].

Figure 5.2 Smart office prototype environment.

3. **"Portable" exhibition space** (under development): is a two-room exhibition space of about 50m^2 that can be taken apart, shipped and reassembled in exhibitions. It comprises interactive exhibits integrating AmI technologies developed in the context of the AmI Programme [38]. A 3D rendering of the exhibition space can be seen in figure 5.3.

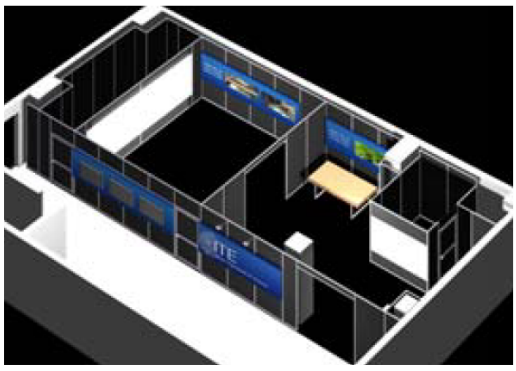

Figure 5.3 3D rendering of ICS-FORTH' exhibition space.

4. **AmI Facility** (under construction): is an Ambient Intelligence facility that will occupy a two-floor building of 3.000m^2, comprising simulated AmI environments and their support spaces, laboratory spaces, staff offices and public spaces.

The simulated environments are full-scale reproductions of the corresponding real-life spaces. Representative users will be invited to use these spaces for limited time periods. The variety of the simulated environments in the Facility will offer the researchers a unique opportunity to explore specific application domains and investigate the interaction between heterogeneous smart environments. A digital maquette and some blueprints of ICS-FORTH' AmI Facility can be seen in figure 5.4.

One of the key application domains that will be addressed in this new facility is Ambient Assisted Living. The northern part of the building will simulate a home environment that will expand on two floors which will be linked through both a staircase and an elevator and will include an open-space living room, a kitchen, a house office, two bedrooms (one for adults and one for children) and a bathroom. The AmI home will be fully accessible by the elderly and people with disabilities [38].

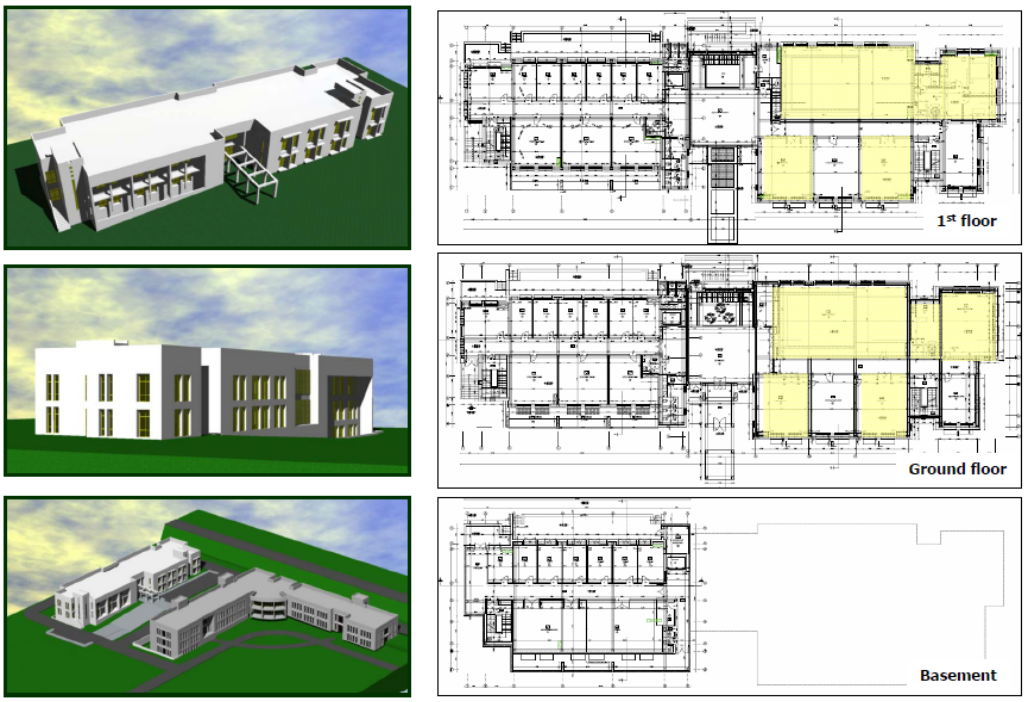

Figure 5.4 Digital maquette and Blueprints of ICS-FORTH' AmI Facility (currently under construction).

5.2 Experimental Results

5.2.1 Demo Implementation in AmI Environment

We have implemented the recognition system and integrated it within AmI Sandbox, a real ambient intelligence (AmI) environment described in the previous section. The facility includes a complex of six rooms equipped with state of the art AmI hardware and software, made available through a middleware infrastructure.

A colleague of ours (not otherwise participating in this work) was given the instructions to enter the facility and perform the actions stated in the running example scenario of Section 1.4.2. The user was not given any other instructions or guidance. A video of the demo is freely available[3]. Screenshots are shown in figure 5.5 at representative moments.

We ran the system with the atomic activities detected from the facility and it correctly identified all user activities. The machine we used for this demonstration and all the following experiments is equipped with an Intel Pentium M 750, clocked at 1.86 GHz, 1 GB of RAM, Microsoft Windows XP, and Java version 1.6. In addition it had typical programs running in the background e.g. antivirus program. These background programs were not disabled during the experiment. The initial heap size was 256 MB and maximum heap size of 512M MB.

The atomic activities created from our demo were *22*, the plausible complex activities *122* and the conflicts between them *6693*. The total computation time was *5.0567* minutes. *97.6%* of the computation time was needed for the jess rules and more specifically for the conflicts detection process. *0.5%* of the systems computation time was the plausible complex activities detection time and *0.3%* for the conflict resolution and preference optimization process. The results are analytically presented in table 5.1.

As shown from the demo results the main computational cost of the system is the detection of the conflicting activities as their number (6693 pairs of conflicting activi-

[3] https://rapidshare.com/files/1954778061/AmI_Demo.rar . Some machines have to be operated through software for some atomic activities to be registered.

ties) is match larger than the rest of the facts. If we have n activities that are all in conflict with each other the number of conflicting pairs is: $\frac{(n-1)*n}{2}$. So the number of conflicts is a significant factor for our system's computation time.

Table 5.1 Computation time of the systems modules when running the atomic activities detected in our demo (running example scenario implementation).

Module		Computation Time (msec)	Percentage over Total Computation Time	Computation Time (min)
Jess rules	**Plausible Complex Activities Detection**	*937*	*0.309 %*	*5.029*
	Conflicts Detection	*295266*	*97.317 %*	
Sat4J	**Conflict Resolution and Preference Optimization Process**	*1563*	*0.515 %*	*0.026*
Activity Recognition System		*303406*	*100 %*	*5.0567*

Moreover in order to demonstrate the system's ability to report activities at different levels of detail we ran the recognition algorithm with various settings of the preference parameters:

- When setting the preference parameters in formula (17) from section 3.3 to (*a=0.1, b=0.1, c=0.8*) higher preference is given to scenarios that explain more atomic activities, i.e., detailed scenarios. In this case, the system returns the scenario: "Resting on bed", "Watching TV", "Talking on Telephone", "Watching Slideshow", "Watching TV". Mary's activities are described in detail, as we asked the system to explain the maximum possible number of atomic activities. The second line of figure 5.6 illustrates the returned scenario with the given preference parameters.

- When setting the preference parameters to (*a=0.85, b=0.1, c=0.05*) higher preference is given to scenarios with activities of longer temporal duration, even if some atomic activities are not explained (do not participate in any recognized complex activity). In this case, the system returns a single activity "User is relaxing at home". This is explained as follows: a plausible activity "User is relaxing at home" is identified encompassing the activities ("Resting on bed", "Watching TV", "Talking on Telephone", "Watching TV") *with duration the whole demo length*. Notice that, the "relaxing" activity is plausible even though the "Watching Slideshow" activity is interleaved, is also recognized and obviously conflicts with it. The third line of figure 5.6 illustrates the returned scenario with the given preference parameters.

Though adjusting our optimization function, results in different abstraction levels in our results. The recognized activities in our demo and their confidence, time duration and number of used activities can be seen in table 5.2. Note that the complex activity "User is resting on bed" has been recognized with confidence 75% because the user did not perform in the demo the optional event "Turn on light". Although an event from our definition was missing the system still recognized the complex activity, just with lower confidence. Similarly with the last complex activity "User is watching TV".

Figure 5.5 Screenshots from demonstration of the running example of Section 1.4.2. From left to right: a) Resting on bed b) Watching TV c) Talking on telephone d) Watching slideshow in a different room e) Watching TV (not shown)

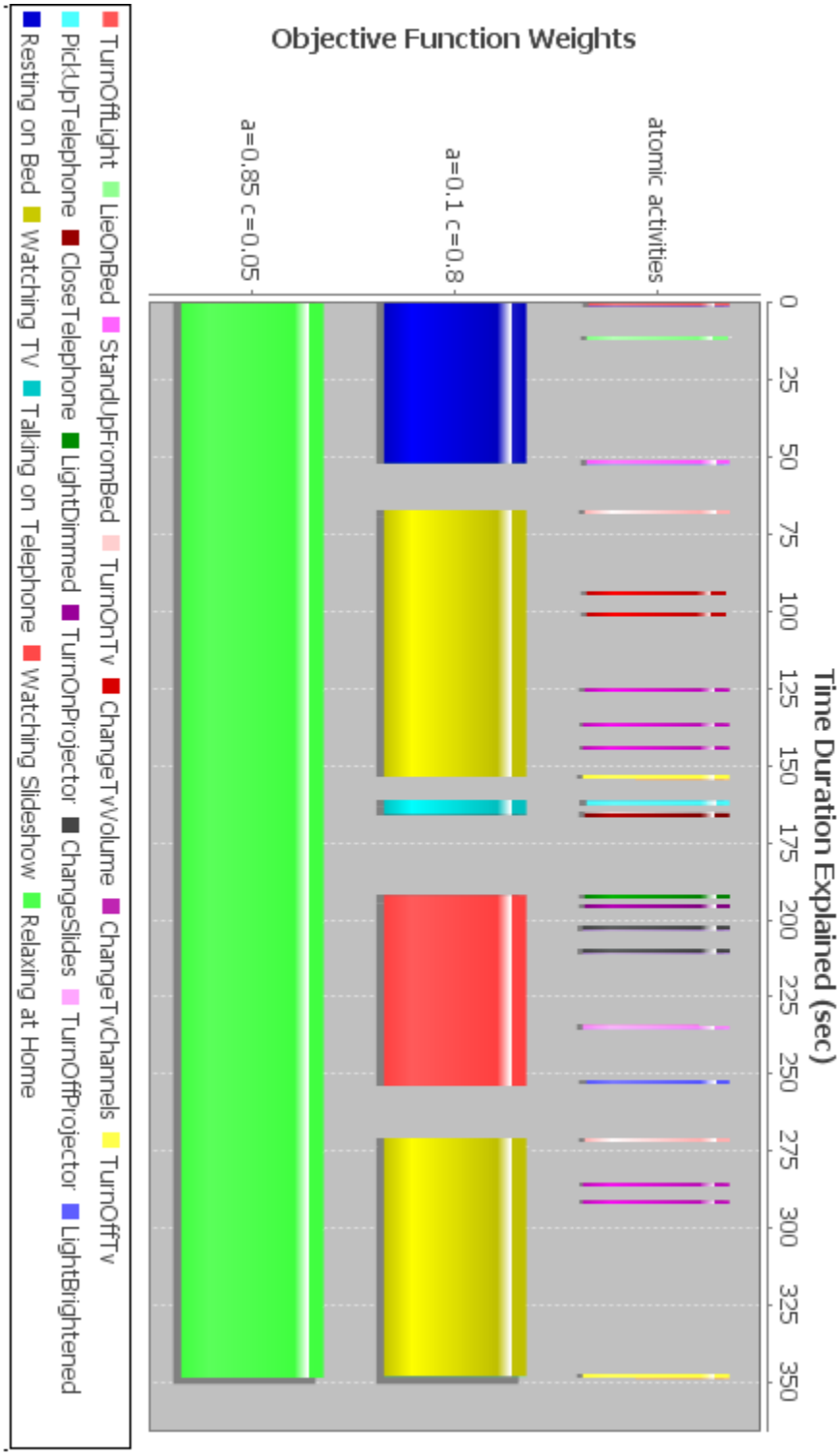

Figure 5.6 Controlling the level of detail of the recognized activities with the use of preference parameters *a*, *b*, and *c* (Eq. 9). From top to bottom the times lines correspond to the atomic activities, recognized scenario when higher preference (*c = 0.8, a = 0.1, b=0.1*) is given to the atomic activities "explained" by the scenario, recognized scenario when higher preference is given to recognizing longer complex activities (*a =0.85, c = 0.05, b=0.1*).

Table 5.2 Recognized activities in our demo and their confidence, duration and number of used activities.

Objective function weights	Recognized complex activity	Confidence	Duration	#Used activities
a=0.85 b=0.1 c=0.05	Relaxing at home	350	100%	54,5%
a=0.1 b=0.1 c=0.8	Resting on bed	75	15%	13,6%
	Watching TV	100	25,2%	18,2%
	Talking on telephone	100	1,6%	9,1%
	Watching slideshow	100	17,7%	22,7%
	Watching TV	75	22,2%	13,6%

5.2.2 Simulation Studies

We conducted some simulation studies in order to see how the system's average computational time is affected by the number of atomic activities it takes as input. 20 random datasets have been created. We started with a set of 4 datasets with 5 atomic activities in each. Then we created another 4 sets of datasets where every time we increased the number of atomic activities by 5. Figure 5.7 illustrates the system's average total running time, Jess running time (i.e. detection of plausible complex activities and conflicts) and Sat4J running time (i.e. conflict resolution and preference optimization process) as a function of the input dataset size. The average running time of the system increases exponentially because of the rapid increase of conflicts. More specifically the conflict detection time increases exponentially causing the system to delay.

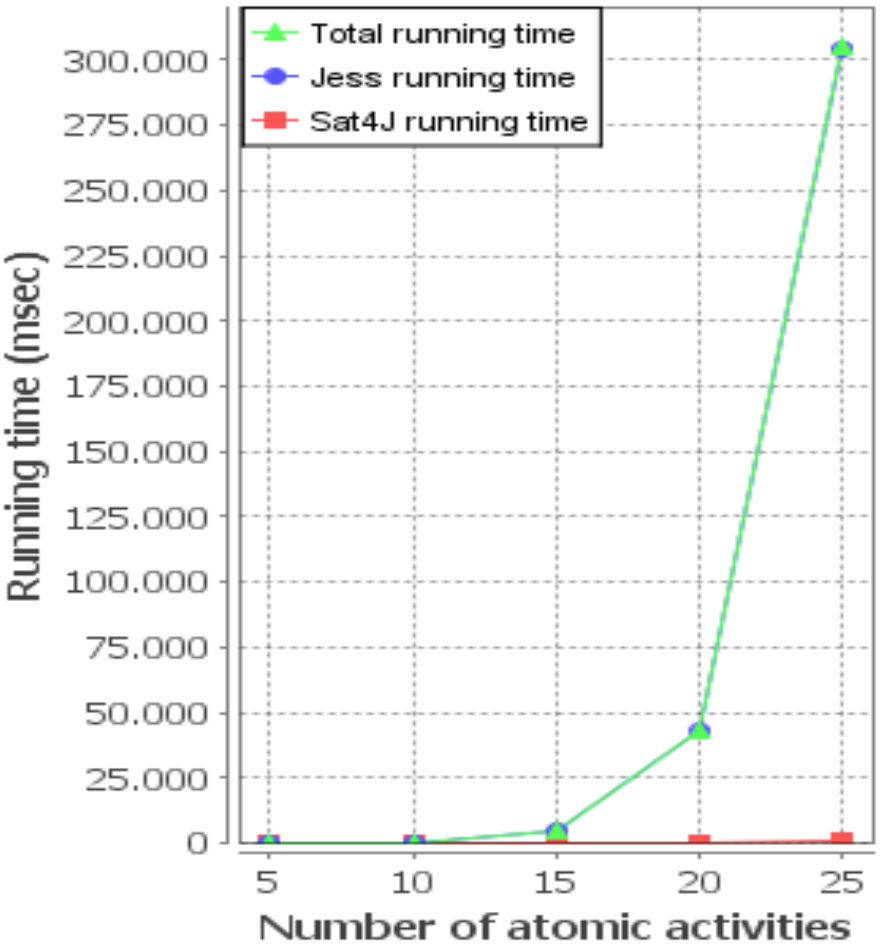

Figure 5.7 System's average total running time, Jess running time (i.e. detection of plausible complex activities and conflicts) and Sat4J running time (i.e. conflict resolution and preference optimization process) as a function of the number of atomic activities given as input.

Moreover to evaluate the robustness of the system, we conducted a number of simulation studies by running 40 datasets containing *1592 atomic activities*. The activity types in the datasets are the same as in the running example scenario of Section 1.4.2. We generated the datasets and then added randomly different percentages of noise. Noise can be split into random activities that are inserted into our datasets and lost activities. For a given level *l* of noise, *l×10%* of total atomic activities in the datasets is deleted (lost activities) and *l×90%* random atomic activities are inserted in the dataset.

Generally the relations between the ground truth and the outcomes of a classification test (illustrated in table 5.3) are the following:

- True Positive (TP): correctly identified
- False Positive (FP): incorrectly identified
- True Negative (TN): correctly rejected
- False Negative (FN): incorrectly rejected

In order to evaluate the results of simulation studies, evaluation measures, TP, Accuracy, Precision and Recall have been used. Their definitions are:

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

$$precision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{TP + FN}$$

The evaluation results are shown in Fig. 5.7. As expected the accuracy, precision and recall values decrease as the percentage of noise in the datasets increases (Fig. 5.7-up). But even with high level of noise (e.g. 50%) activities are accurately recognized (accuracy equal to 0.83). So the system has been proven robust to noise.

As illustrated in Fig. 5.7-down from the total of activities correctly recognized (True Positives) the percentage with temporal errors (incorrect recognized start time or end time) is also affected by the noise levels in the datasets.

Table 5.3 Relations between of the ground truth and outcomes of the test.

		Ground Truth	
		True	**False**
Test Outcome	**Positive**	*TP*	*FP*
	Negative	*FN*	*TN*

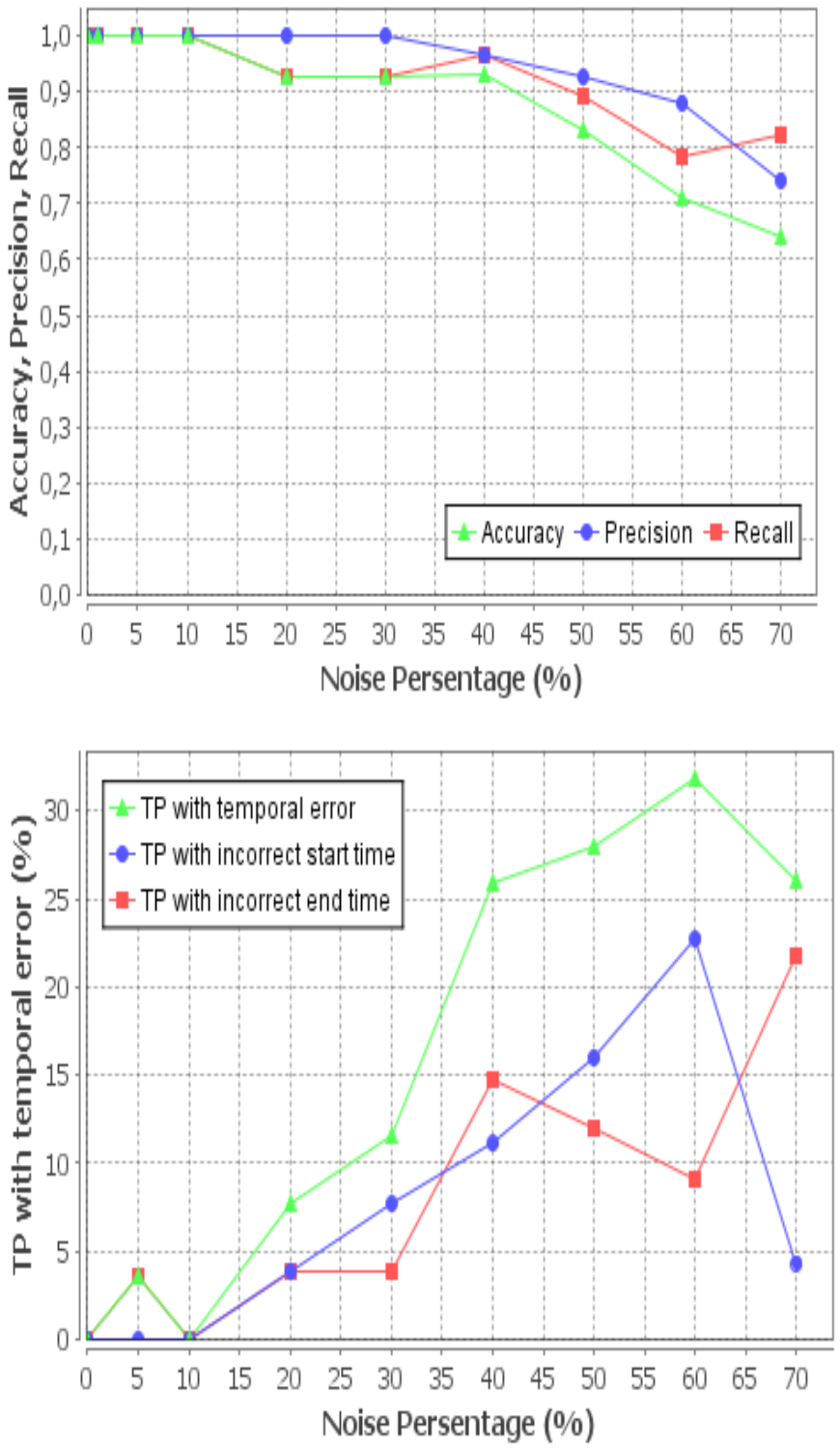

Figure 5.8 Robustness of activity recognition in the presence of noise. (Up) Accuracy, precision and recall of the system as a function of noise level *l*. (Down) True Positives (TP) with temporal errors (incorrect start or end time) as a function of noise level *l*. For a given level *l*, *l*×*10%* of total atomic activities in the datasets is deleted (lost activities) and *l*×*90%* random atomic activities are inserted in the dataset.

6 Conclusion

To conclude this thesis, we summarize and discuss its main contributions, and propose possible directions for future research.

6.1 Synopsis

In general activity recognition settings such as Ambient Assisted Living environments, a system should identify all of user activities and their relations, e.g. their temporal sequence or subsumption (for activities hierarchically organized). However due to noise or other imperfections a naïve recognition system may report activities that are logically inconsistent with each other (conflicting activities).

We have presented a rule-based activity recognition system for hierarchically-organized complex activities that returns only logically consistent sets of activities (scenarios). This has been achieved by detecting conflicting activities and explicitly formulating them as a Weighted Partial MaxSAT problem such that any solution to it corresponds to a set of identified activities (scenario) that do not conflict.

The system also has the ability to adjust the desired level of detail of the scenarios returned. This is accomplished by assigning preferences to clauses of the Weighted Partial MaxSAT problem. Each complex activity's weight is calculated by taking into account its confidence, temporal duration and number of used atomic activities. A fully implemented scenario in an AmI environment demonstrated that the system is efficiently working and that the level of detail can be easily adjusted according to our preferences.

The system handles noise and uncertainty, with the use of optional activities and confidence factors in its facts. Experimental results have shown that complex activities are being recognized, even when some of their optional conditions have not been detected. So the system has been proven to be robust to noise.

In most related systems you have to query one by one all the possible activities. Thus the key difference between our system and the existing approaches is that while

they are answering the question "*Was the complex activity E occurring at time t?*" our system answers the question "*Which complex activities have occurred in the given time interval?*".

6.2 Future Directions

In the future we plan to improve the system's performance, by decreasing its computational time. One optimization we could do is to avoid the identification of some unnecessary plausible activity occurrences. As we take all the possible combinations of our atomic and complex activities for the recognition of other complex activities the number of facts and conflicts in our working memory increases exponentially. Avoiding some of them will lead to a significant decrease of the system's computational time.

Another consideration is the idle time between the activities. When the system is returning more abstract scenarios, there is the possibility that we explain large time periods of idle user time with short time durational activities. For example if the user is watching TV for 5 minutes, then he does no other activity for 2 hours and then he watches TV for another 10 minutes, it would be wrong to return a scenario that the user is watching TV for 3 hours and 15 minutes. With some specific values in the preference parameters of our system the returned scenario would be exactly the previous one. One possible way to deal with such scenarios is to recognize an activity "Idle time", whenever the user is doing no other activity. This way the system could not ignore large periods of time when the user is doing nothing. Of course this approach would increase the number of plausible complex activities and the computational time of our system.

The optimization techniques presented in this work could accommodate other types of preferences and be generalized to other settings. Though we want in the future to include more preference factors in our system, for controlling the abstraction level in the scenarios returned.

One more future direction is to brake down the use of resources in our activities. Instead of assuming that a resource is used the whole time interval of a complex activity, we could keep a set of time intervals in which the resource is actually being used. In this manner we could more flexibly address interleaved complex activities. For

example suppose that a user is cooking. The resource “stove” is now used from the beginning to the end of the complex activity “UserIsCooking”. But the resource is actually being used only in a subset of the activity’s interval e.g. between the occurrence of the atomic activities “TurnOnHotPlate” and “TurnOffHotPlate”.

Finally more extensive experiments could be conducted in order to work out the relative merits and weaknesses compared to other approaches.

Bibliography

[1] Eurostat Press Office (2011) EU27 population Projection 2010-2060 10/6/2011 « http://epp.eurostat.ec.europa.eu/cache/ITY_PUBLIC/3-08062011-BP/EN/3-08062011-BP-EN.PDF »

[2] Ducatel, K., Bogdanowicz, M., Scapolo, F., Leijten, J., Burgelma J.C.: Scenarios for ambient intelligence in 2010. In: ISTAG 2001 Final Report, IPTS. Seville (2000)

[3] Van Den Broek, G., Cavallo, F., Wehrmann, C.: AALIANCE Ambient Assisted Living Roadmap In: Ambient Intelligence and Smart Environments, vol. 6, pp. 136 (2010)

[4] Liao, H.H., Chang, J.Y., Chen, L.G.: A localized Approach to abandoned luggage detection with Foreground –Mask sampling. In: Proceedings of 5th IEEE International Conference on Advanced Video and Signal based Surveillance, pp. 132-139. Santa Fe (2008)

[5] Yamato, J., Ohya, J., Ishii, K.: Recognizing Human Action in Time-Sequential Images Using Hidden Markov Model. In: Proc. Computer Vision and Pattern Recognition, pp. 379-385 (1992)

[6] Nguyen, N.T., Phung D.Q., Venkatesh, S., Bui, H.H.: Learning and detecting activities from movement trajectories using the hierarchical hidden Markov model. In: Proceedings of IEEE International Conference on Computer Vision and Pattern Recognition (CVPR), pp. 955–960. San Diego (2005)

[7] Vail, D.L., Veloso, M.M, Lafferty, J.D.: Conditional random fields for activity recognition. In: International Conference on Autonomous Agents and Multi-agent Systems (AAMAS). (2007)

[8] Liao, L., Fox, D., Kautz, H.: Hierarchical Conditional Random Fields for GPS based Activity Recognition. In: Robotics Research the Twelfth International Symposium (ISRR-05). Springer-Verlag (2006)

[9] Wu, T., Lian, C., Hsu, J.Y.: Joint recognition of multiple concurrent activities using factorial conditional random fields. In: Proceedings of AAAI Workshop on Plan, Activity, and Intent Recognition. California (2007)

[10] Lavee, G., Rivlin, E., Rudzsky, M.: Understanding Video Events: A Survey of Methods for Automatic Interpretation of Semantic Occurrences in Video. In: IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews, vol. 6:5. (2009)

[11] Patterson, D.J., Fox, D., Kautz, H., Philipose, M.: Finegrained activity recognition by aggregating abstract object usage. In ISWC ’05: Proceedings of the Ninth IEEE International Symposium on Wearable Computers.: IEEE Computer Society. Washington, DC, USA (2005)

[12] Oliver, N., Horvitz, E., Garg, A.: Layered representations for human activity recognition. In: Computer Vision and Image Understanding Journal, vol. 96:2, pp.163–180 (2004)

[13] Hu, D.H., Yang, Q.: CIGAR: concurrent and interleaving goal and activity recognition. In: Proceedings of the 23rd AAAI Conference on Artificial Intelligence (AAAI 2008). Chicago, Illinois, USA, (2008)

[14] Medioni, G., Cohen, I., Brémond, F., Hongeng, S., Nevatia, R.: Event detection and analysis from video streams. In: IEEE Trans. Pattern Anal. Mach. Intell., vol. 23:8, pp. 873–889. (2001)

[15] Hongeng, S., Nevatia, R., Brémond, F.: Video-based event recognition: Activity representation and probabilistic recognition methods. In: Comput. Vis. Image Understand., vol. 96:2, pp. 129–162. (2004)

[16] Kowalski, R., Sergot, M.: A logic-based calculus of events. New Generation Computing, vol. 4:1, pp.67-96. (1986)

[17] Artikis, A., Sergot, M., Paliouras, G.: A Logic Programming Approach to Activity Recognition. In: Proc. of ACM International Workshop on Events in Multimedia (2010)

[18] Paschke, A., Bichler, M.: Knowledge representation concepts for automated SLA management. Decision Support Systems, vol. 46:1, pp. 187-205. (2008)

[19] Dousson, C., Maigat, P.L.: Chronicle recognition improvement using temporal focusing and hierarchisation. In: Proceedings of International Joint Conference on Artificial Intelligence (IJCAI), pp. 324-329. (2007)

[20] Shet, V., Harwood, D., Davis, L.: VidMAP: video monitoring of activity with Prolog. In: Advanced Video and Signal Based Surveillance IEEE. (2005)

[21] Shet, V., Neumann, J., Ramesh, V., Davis, L.: Bilattice-based logical reasoning for human detection. In: Proc. Of IEEE Computer Vision and Pattern Recognition (CVPR) (2007)

[22] Richardson, M., Domingos, P.: Markov logic networks. Machine Learning, 62(1-2), 107-136 (2006)

[23] Tran, S.D., Davis, L.S.: Event modeling and recognition using Markov logic networks. In: Forsyth, D., Torr, P., Zisserman, A. (eds.) Computer Vision. LNCS, vol. 5303, pp. 610-623. Springer, Heidelberg (2008)

[24] Biswas, R., Thrun, S., Fujimura, K.: Recognizing activities with multiple cues. In: Elgammal, A., Rosenhahn, B., Klette, R. (eds.) Human Motion. LNCS, vol. 4814, pp. 255-270. Springer, Heidelberg (2007)

[25] Helaoui, R., Niepert, M., Stuckenschmidt, H.: Recognizing Interleaved and Concurrent Activities: A Statistical-Relational Approach. In: Proceedings of the 9th

Annual IEEE International Conference on Pervasive Computing and Communications (2011)

[26] JESS , The Rule Engine for the Java Platform, "http://www.jessrules.com/"

[27] Le Berre, D., Parrain, A.: The Sat4j library, release 2.2. Journal on Satisfiability, Boolean Modeling and Computation (JSAT). 7, 59-64 (2010)

[28] Friedman-Hill, E.: JESS in Action: Rule-Based Systems in Java. Manning Publications Co., (2003)

[29] Laun, W.: Predicate Calculus and JESS (2009)

[30] Rete algorithm 22/6/11 "http://en.wikipedia.org/wiki/Rete_algorithm"

[31] JESS , The Rule Engine for the Java Platform - The Rete Algorithm 22/6/11 "http://www.JESS rules.com/doc/70/rete.html"

[32] David Corsar, Derek H. Sleeman, Anne McKenzie: Extending Jess to Handle Uncertainty. SGAI Conf. 2007: 81-93

[33] Buchanan, B.G., Shortliffe, E.H.: Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project. Addison-Wesley, Reading, MA., (1984)

[34] Orchard, B., The FuzzyJ Toolkit. "http://www.iit.nrc.ca/IR_public/fuzzy/fuzzyJToolkit2.html"

[35] CLIPS: A Tool for Building Expert Systems 25/6/2011 "http://clipsrules.sourceforge.net/"

[36] LPA WIN-PROLOG 4.9. 25/6/2011 "http://www.lpa.co.uk/win.htm"

[37] Stephanidis, C., Argyros, A., Grammenos, D., Zabulis, X.: Pervasive Computing@ ICS-FORTH. In: Proceedings of Pervasive 2008 Workshop, pp. 119-124. Sydney, Australia (2008)

[38] Grammenos, D., Zabulis, X., Argyros, A., Stephanidis, C.:FORTH-ICS internal RTD Programme 'Ambient Intelligence and Smart Environments'. In: 3rd European Conference on Ambient Intelligence. Salzburg, Austria (2009)

[39] Grammenos, D., Georgalis, Y., Partarakis, N., Zabulis, X., Sarmis, T., Kartakis, S., Tourlakis, P., Argyros, A., Stephanidis, C.: Rapid Prototyping of an AmI augmented Office Environment Demonstrator. In: Proceedings of HCI International 2009, Lecture Notes in Computer Science Series of Springer, pp. 397-406. Berlin Heidelberg (2009)

[40] Een, N., Sorensson, N.: MiniSAT specification: An extensible SAT solver. In: Proceedings of the Sixth International Conference on Theory and Applications of Satisfiability Testing, LNCS 2919, pp 502-518, (2003)

[41] Maximum satisfiability problem 27/6/2011
"http://en.wikipedia.org/wiki/Maximum_satisfiability_problem"

[42] Conjunctive normal form 27/6/2011
"http://en.wikipedia.org/wiki/Conjunctive_normal_form"

Appendix

Appendix A.1 – Source Code Example

Here the source code for recognizing the complex activity “User is watching TV” is presented:

```
(deftemplate AtomicEvent
 "An atomic event."
  (slot occurrenceTime (type LONG)) ;expressed as milliseconds since 1 Jan 1970
  (slot date)
  (slot videoTime)
  (slot location)
  (slot actor)
  (slot actorStatus)
  (slot weight (default 1))
)

(deftemplate TurnOnTv extends AtomicEvent
  "The user turns on the TV."
  (slot location (default livingroom))
)

(deftemplate TurnOffTv extends AtomicEvent
  "The user turns off the TV."
  (slot location (default livingroom))
)

(deftemplate ChangeTvChannels extends AtomicEvent
  "The user is changing channels in the TV."
```

```
  (slot location (default livingroom))
)

(deftemplate ChangeTvVolume extends AtomicEvent
  "The user is changing the volume of the TV."
  (slot location (default livingroom))
)

(deftemplate ComplexEvent
  "A complex event."
  (declare (slot-specific TRUE))
  (slot startTime (type LONG)) ;expressed as milliseconds since 1 Jan 1970
  (slot endTime (type LONG) ) ;expressed as milliseconds since 1 Jan 1970
  (slot confidence)
  (slot location)
  (slot totalEvents)
  (slot percentageIncrease)
  (slot timeWindow)
  (slot actor)
  (slot actorStatus)
  (slot numberOfUsedEvents)
  (multislot resources)
  (multislot foundEvents)
  (multislot foundEventTypes)
)

(deftemplate UserIsWatchingTv extends ComplexEvent
  "The user is watching TV."
  (declare (slot-specific TRUE))
  (multislot resources (default PhilippsTV usersAttention))
  (slot location (default bedroom))
)
```

```
(defrule initial-user-watching-tv

   ?fact1 <- (TurnOnTv (occurrenceTime ?t1) (location bedroom) (actor ?a) (actorStatus ?st))

   ?fact2 <- (TurnOffTv (occurrenceTime ?t2 &:(< ?t1 ?t2) &:(< (- ?t2 ?t1) ?*timeWindow*))

                              (location bedroom)  (actor ?a) )

   =>

   (bind ?totalEvents 4)

   (bind ?increaseOfEachEvent (/ 100 ?totalEvents))

   (bind ?newFactId

      (assert (UserIsWatchingTv (confidence (* 2 ?increaseOfEachEvent))
                              (timeWindow ?*timeWindow*) (startTime ?t1) (endTime ?t2)
                              (percentageIncrease ?increaseOfEachEvent)
                              (totalEvents ?totalEvents) (location bedroom)
                              (foundEvents ?fact1 ?fact2) (actor ?a) (actorStatus ?st)
                              (foundEventTypes TurnOnTv TurnOffTv)
                              (numberOfUsedEvents 2)
            )
         )

    )

   (if (eq ?*firstComplexEventId* 0) then (bind ?*firstComplexEventId* (call ?newFactId getFactId)))

)


(defrule user-watching-tv-changing-channels

   ?fact <- (ChangeTvChannels (occurrenceTime ?t1) (location ?l) (actor ?a))

   ?tvFact <- (UserIsWatchingTv (confidence ?oldConf&:(< ?oldConf 100))
                  (timeWindow ?tw)(location ?l) (actor ?a) (actorStatus ?st)
                  (startTime ?st1 &:(< ?st1 ?t1)) (endTime ?et1&:(< ?t1 ?et1))
                  (totalEvents ?totalEvents) (percentageIncrease ?increaseOfEachEvent)
                  (foundEventTypes $?listEventTypes &:(not (member$ ChangeTvChannels
                                                                  $?listEventTypes)))
                  (foundEvents $?listEvents) (numberOfUsedEvents ?numberOfUsedEvents)

         )

  (not (exists(ChangeTvChannels (occurrenceTime ?t&:(< ?t ?t1) &:(< ?st1 ?t) &:(< ?t ?et1))
                                 (location ?l) (actor ?a) )) )

  (not (UserIsWatchingTv (startTime ?st2 &:(<= ?st2 ?et1)) (endTime ?et2 &:(<= ?st2 ?et1))
                          (location ?l) (actor ?a)
                          (foundEvents $?list2 &:(checkForEventWithSameUsedEvents ?listEvents
                                                                  ?list2 ?fact))

       )

   )

   =>
```

```
  (assert  (UserIsWatchingTv (confidence (+ ?oldConf ?increaseOfEachEvent))
                    (timeWindow ?tw) (startTime ?st1) (endTime ?et1)
                    (percentageIncrease ?increaseOfEachEvent)
                    (totalEvents ?totalEvents) (location ?l) (actor ?a) (actorStatus ?st)
                    (foundEventTypes (insert$ $?listEventTypes 1 ChangeTvChannels))
                    (foundEvents (insert$ $?listEvents 1 ?fact))
                    (numberOfUsedEvents (+ 1 ?numberOfUsedEvents))

          )

  )

)



(defrule user-watching-tv-changing-volume

   ?fact <- (ChangeTvVolume (occurrenceTime ?t1) (location ?l) (actor ?a))

   ?tvFact <- (UserIsWatchingTv (confidence ?oldConf&:(< ?oldConf 100))
                    (timeWindow ?tw) (location ?l) (actor ?a) (actorStatus ?st)
                    (startTime ?st1 &:(< ?st1 ?t1)) (endTime ?et1 &:(< ?t1 ?et1) &:(< (- ?et1 ?t1) ?tw) )
                    (totalEvents ?totalEvents) (percentageIncrease ?increaseOfEachEvent)
                    (foundEvents $?listEvents) (foundEventTypes $?listEventTypes &:(not (member$
                                                        ChangeTvVolume $?listEventTypes)))
                    (numberOfUsedEvents ?numberOfUsedEvents)

               )

   (not (exists(ChangeTvVolume (occurrenceTime ?t&:(< ?t ?t1) &:(< ?st1 ?t) &:(< ?t ?et1))
                                    (location ?l) (actor ?a) )) )

   /* check that there is not an other overlapping complex event with the same list of used facts (avoid
multiple assertions of same complex event) */

   (not (UserIsWatchingTv          (startTime ?st2 &:(<= ?st2 ?et1)) (endTime ?et2 &:(<= ?st2 ?et1))
                           (location ?l) (actor ?a)
                           (foundEvents $?list2 &:(checkForEventWithSameUsedEvents ?listEvents
                                                                                    ?list2 ?fact))

          )

   )

   =>

   (assert (UserIsWatchingTv  (confidence (+ ?oldConf ?increaseOfEachEvent))
                           (timeWindow ?tw) (startTime ?st1) (endTime ?et1)
                           (percentageIncrease ?increaseOfEachEvent) (totalEvents ?totalEvents)
                           (location ?l) (actor ?a) (actorStatus ?st)
                           (foundEventTypes (insert$ $?listEventTypes 1 ChangeTvVolume))
                           (foundEvents (insert$ $?listEvents 1 ?fact))
                           (numberOfUsedEvents (+ 1 ?numberOfUsedEvents))

         )

   )

)
```