SAPIENZA
UNIVERSITÀ DI ROMA

Habit Mining: Applying Process Mining to Activities of Daily Life

Dipartimento di Ingegneria Informatica Automatica e Gestionale ANTONIO RUBERTI,
SAPIENZA – Università di Roma

Dottorato di Ricerca in Ingegneria Informatica – XXVI Ciclo

Candidate

Francesco Leotta
ID number 799650

Thesis Advisor

Ing. Massimo Mecella

A report submitted in partial fulfillment of the requirements
for the admission to the third year of the Ph.D course in
Computing Science and Engineering

October 2012

Francesco Leotta. *Habit Mining: Applying Process Mining to Activities of Daily Life.*
Ph.D. report. Sapienza – University of Rome

VERSION: 2012, October the 2th
WEBSITE: http://www.dis.uniroma1.it/~leotta
EMAIL: leotta@dis.uniroma1.it

Contents

Chapter 1

Fact Sheet

1.1 Introduction

This preliminary chapter briefly exposes the results that I obtained up to now. Section 1.2 describes the amount of credits I collected during the year. Section 1.3 reports the activities achieved and the publications that I contributed to.

1.2 Exams

Here follow table 1.1, summing up the exams that I already passed. Credits in reds has to be considered odd.

1.3 Main activities and publications

1.3.1 Main Publications

- F. Leotta, M. Mecella and F. Aloise. Pericles: a performance evaluation platform for indoor localization systems. *Proceedings of the 3rd ACM SIGSPATIAL International Workshop on Indoor Spatial Awareness*, Chicago (IL), USA, 2011.

- M. Caruso, F. Cincotti, F. Leotta, M. Mecella. My-World-in-My-Tablet: an Architecture for People with Physical Impairment. *Technical Report of Dipartimento di Ingegneria Informatica, Automatica e Gestionale "A. Ruberti"*, Rome, Italy, 2012.

- F. Leotta and M. Mecella. Introducing PLaTHEA: A Markerless People Localization and Tracking System for Home Automation. *Submitted to international journal*, 2012.

1.3.2 Research project activities

The TOBI project

I have been actively involved in Work Packages 1, 5 and 8 of the TOBI[1] European Research Project (`http://www.tobi-project.org/`) whose aim is to develop practical technology for brain-computer interaction (BCI) that will improve the quality of life of disabled people and the effectiveness of rehabilitation.

A Brain Computer Interface (BCI) allows to use the electroencephalographic (EEG) signal of a subject as control signal for computer applications extracting some features. My work in the TOBI project has been mainly focused on the following aspects:

[1] Tools for Brain Computer Interfaces

Table 1.1. This table summarizes the exams that were passed up to now.

	Name	Professor	Cr.	Date
A	Artificial Intelligence II	D. Nardi	6	2011-09-13
	Machine Learning	L. Iocchi	6	2011-10-17
B	Winter School: Hot Topics in Secure and Dependable Computing for Critical Infrastructures	R. Baldoni and A. Bondavalli	2.5	2012-01-19
	Readings in Business Processes	G. De Giacomo	2.5	2012-05-10
	Least Squares Estimation and SLAM	G. Grisetti	2.5	2012-06-04
	Introduction to STRIPS Planning and Applications in Video-games	S. Vassos	2.5	2012-10-13
C	Rescue Robotics (Sensing and State Estimation)	A. Kleiner	0.5	2011-04-01
	Seminar on Nonlinear system identification and sensor fusion	T. Schoen	0.5	2011-05-19
	Learn to Behave	V. A. Ziparo	0.5	2011-06-06
	The "Roman Model" for Automatic Synthesis in Practice: the SM4All Experience	C. Di Ciccio	0.5	2011-07-26
	DISC Workshops and Tutorials	Various	4	2011-09-[20-22]
	Supervisory Control of Discrete-Event Systems	W.M. Wonham	2	2011-10-[03-07]
	Physarum Can Compute Shortest Paths	V. Bonifaci	0.5	2011-11-14
	First-order Modal Languages for the Specification of Multi-Agent Systems	F. Belardinelli	0.5	2011-11-24
	Mobile Robots for Exploration of Extreme and Uncertain Environments in Terrestrial and Planetary Applications	K. Yoshida	0.5	2012-03-30
	Approximation of Conjunctive Queries	L. Libkin	0.5	2012-05-30

- **Standardization**. Though BCI is an active research field since '90s, there not exists any standardized way of making modular BCI systems due to the lack of standard interfaces for the transmission of raw biosignals, signal features, classification results, markers and events. My work in TOBI WP8 has been directed at the definition of such a kind of interfaces. Additionally, as a proof of concept, I have used the defined interface to design and develop a portable BCI system.

- **Hybrid BCI**. Traditional BCIs extract control features using a sole EEG phenomenon. TOBI WP5 aims at defining an Hybrid BCI able to exploit different EEG phenomenon at the same time and to optionally combine them with other kind of bio-signals (such as electromyogram - EMG and electrooculogram - EOG). My work on this work package has been focused on the implementation of the interfaces defined in WP8.

- **Combination of BCI with established Assistive Technology (AT) platforms**. My work on this work package has been focused on the design of a BCI P300 stimulation to be overlaid over existing AT softwares.

I contributed to the writing of different papers, related to the TOBI project, which are:

- F. Aloise, F. Schettini, P. Aricò, F. Leotta, S. Salinari, D. Mattia, F. Babiloni and F. Cincotti. P300-based BCI for Environmental Control: an Asynchronous Approach. *Journal of Neural Engineering*, 2011.

- A. Riccio, F. Leotta, L. Bianchi, F. Aloise, C. Zickler, EJ. Hoogerwerf, A. Kubler, D. Mattia and F. Cincotti. Workload Measurement in a Communication Application Operated through a P300-based BCI. *Journal of Neural Engineering*, 2011.
- C. Zickler, A. Riccio, F. Leotta, S. Hillian-Tress, S. Halder, E. Holz, P. Staiger-SÃďlzer, E.-J. Hoogerwerf, L. Desideri, D. Mattia, A. Kubler. A brain-computer interface as input channel for a standard assistive technology software. *Clinical EEG and neuroscience: official journal of the EEG and Clinical Neuroscience Society (ENCS)*, 2011.
- F. Aloise, F. Schettini, P. Aricò, F. Leotta, S. Salinari, F. Babiloni, D. Mattia and F. Cincotti. Towards Domotic Appliances Control through a Self-Paced P300-based BCI. *International Conference on Bio-Inspired Systems and Signal Processing (BIOSIGNAL)*, 2011.
- F. Schettini, F. Aloise, P. Aricò, F. Leotta, S. Salinari, F. Babiloni, D. Mattia and F. Cincotti. Improving Asynchronous Control for P300-based BCI: towards a completely autoadaptative system. *In Proceedings of 2nd Workshop of the TOBI Project*, 2010.
- GR. Muller-Putz, C. Breitwieser, M. Tangermann, M. Schreuder, M. Tavella, R. Leeb, F. Cincotti, F. Leotta and C. Neuper. TOBI Hybrid BCI: Principle of a New Assistive Method. *In Proceedings of 2nd Workshop of the TOBI Project*, 2010.
- F. Pichiorri, P. Aricò, F. Leotta, F. Aloise, F. Cincotti, M. Secci, M. Petti, D. Mattia. Neurorehabilitation-driven design of hybrid BCI-controlled FES for motor recovery after stroke. *In Proceedings of 3rd Workshop of the TOBI Project*, 2012.
- SC. Kleih, T. Kaufmann, C. Zickler, S. Halder, F. Leotta, F. Cincotti, F. Aloise, A. Riccio, C. Herbert, D. Mattia and A. Kubler. Out of the frying pan into the fire - the P300 based BCI faces real world challenges. Book chapter. *Progress in Brain Research*, Vol. 194, 2011.

The Brindisys Project

Thanks to the experience maturated during my involvement in the TOBI project, I have also participated to the Brindisys projecy (http://www.brindisys.it/) funded by AriSLA (Italian Agency for Research on Amyotrophic Lateral Sclerosis). In particular I have been involved in the design of the architecture and the prototype of an assistive system, which allows people with physical impairments to express themselves and partially preserve their independence in controlling electrical devices at home. In particular the system allows the user to drive a wide range of standard and assistive software using a common interface driven by different input methods ranging from classical assistive technology devices to Brain Computer Interface. The result of this research effort has been described into a technical report.

1.3.3 Teaching Activity

I have been teaching assistant for the academic year 2011/2012 of the Database course of the bachelor degree in Ingegneria Gestionale held by prof. Tiziana Catarci and prof. Monica Scannapieco.

On 14 of May 2012 I held, in collaboration with Mario Caruso, for the Human Computer Interaction course held by prof. Tiziana Catarci, a seminar entitled “Brain Computer Interfaces (BCI) and the case study of Brindisys project”.

Chapter 2

Research

2.1 Introduction

Pervasive computing embodies a vision of computers seamlessly integrating into everyday life, responding to information provided by sensors in the environment, with little or no direct instruction from users. It assumes a number of invisible sensing/computational entities that interact both with users and with the environment in which they operate. In this sense pervasive computing can be considered as a first step towards the *Internet of Things* (IoT) [UHM11].

Smart spaces refer to built environments such as apartments, offices, hospitals, schools, malls, and outdoor areas where pervasive computing is applied to deliver customized services to users in a context-aware manner when they are interacting and exchanging information with the environment. *Smart homes* in particular represent controlled environments where pervasive computing could take advantage of techniques from artificial intelligence (AI) more easily than in other spaces where AI problems soon become intractable [AN06].

Figure 2.1 depicts a typical architecture for a pervasive system deployed in a smart home ([HI06] offers a detailed description of key elements). Such a kind of system aims at providing a certain set of *services* exploiting the information provided by *sensors* potentially deployed anywhere and on any objects or human bodies.

Sensors in pervasive computing can capture a broad range of information about different aspects such as environment, devices, users and interactions [CD07]. They do not only include those traditionally employed for space automation (e.g. presence detectors, smoke detectors, contact switches for doors and windows, network and close circuit cameras) but also modern units which

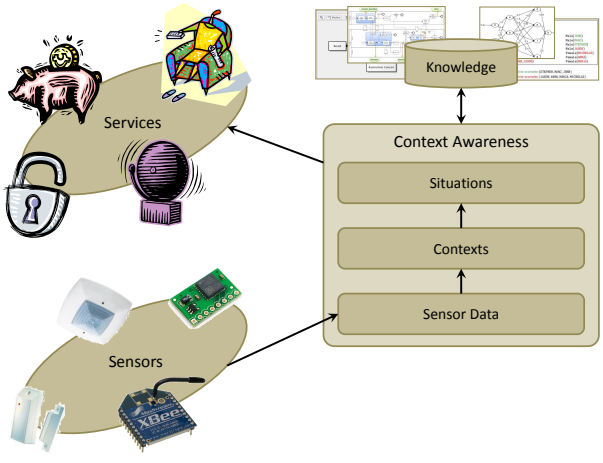

Figure 2.1. The architecture of a smart home pervasive system.

are nowadays growingly available as off-the-shelf products (e.g. IMUs - Inertial Measurements Units such as accelerometer and gyroscopes, WSN-enabled motes). Some kind of sensors lift concerns about privacy and comfort; for example it is reasonable to think users (except those, such as elderly, who need specific monitoring services) will find uncomfortable to wear IMUs or to be continuously followed by cameras. Sensing technologies have made significant progress on designing sensors with smaller size, lighter weight, lower cost, and longer battery life. Sensors can thus be embedded in an environment and integrated into everyday objects and onto human bodies. However, sensor data exhibits high complexity (different modalities, huge volumes, and inter-dependency relationships between sources), dynamism (real-time update and critical ageing), accuracy, precision and timeliness.

The term *sensor data* encompasses raw (or minimally-processed) data retrieved from both physical sensors and "virtual" sensors observing digital information such as user calendars and network traffic. We can imagine a smart house producing a *sensor log* containing raw measurements from deployed sensors. In dealing with the real world, these sensors typically produce imperfect data. Noisy sensor data may result in misunderstanding of a user or environmental state, which potentially lead to incorrect behaviours of final services. These sensors also have their own technical limitations, are prone to breakdown, or may be disconnected from the sensor network or be vulnerable to environmental interference. This leads to the uncertainty issue of sensor data, which can be out of date, incomplete, imprecise, and contradictory with each other.

Services belong to different areas including energy saving, security, easy-living, safety monitoring (including health monitoring), and are provided in an automated or semi-automated manner. A service provider should therefore not concern itself with the individual pieces of sensor data (which room the user is in, what his heart rate or blood pressure is): rather, this information should be interpreted into a higher, domain-relevant concept, such as whether the user is suffering a heart attack or exercising. Thus between sensors and services we put an additional layer called *context awareness*. In order to define the concept of context awareness as a layered comprehension of the environment (see [CNM+09]) we briefly introduce the concepts of *context* and *situation* as defined in [DCGA+06].

Sensor data is aggregated to form *context* - the environment in which the system operates, understood symbolically - which may be further sub-divided into context derived directly from sensors (primary context) and that inferred and/or derived from several data streams (secondary context). An important form of secondary context is represented by *derived events* (see Section 2.4.2) representing small, higher-level inferences about contextual information, such as the action of "chopping" derived by observing motion over time. As well as primary context inherits sensor uncertainty, secondary context may introduce an additional uncertainty level given by the processing itself. Take as an example a video-based gesture recognition system, that is a processing unit that, using videos acquired from cameras, recognize hand gestures of users detected into the environment. The quality of a service depending on this secondary context will strongly depends on the performance of this latter. If we also consider the fact that some service may be critical for human safety we understand how performance evaluation of both sensors and secondary context is important [LMA11].

Finally, a situation (also known as situational context) is an abstraction of the events occurring in the real world derived from context and hypotheses about how observed context relates to factors of interest to designers and applications. Situations typically fuse several sources of context, as well as *domain knowledge*, and *spatial and temporal models* of the expected behaviour of the phenomena being observed. In the following we will refer to the latter as to *habit models*. A list common habits (also known as Activities of Daily Life (ADLs)) can be found in [LHP+07]. The execution of an habit is basically a sequence of actions performed by a set of different persons together with other derived events. A habit model should represents the huge number of different execution sequences which correspond to the specific habit.

The aim of this report is to introduce *Habit Mining* (see Section 2.4). Techniques from AI, including machine learning (ML), have been traditionally employed in situation identification (see

Section 2.2). Our intuition is that, habit models show many similarities with business process models but these latter lack of some specific aspects taken into account by situation identification techniques, namely: *uncertainty*, *space* and *time*. While process mining [vdA11], usually employs mined models to perform *conformance checking* of executions and static *verification*, habit mining is more focused on the *runtime probabilistic classification of situations* with the classifier comparing the event log produced by the house with habit models and domain knowledge and return confidence levels. This *probabilistic conformance checking* is employed to enhance habits models.

Specifically, peculiar features of habit models make them suitable to be described using *declarative* formalisms (see Section 2.3) rather than *procedural* ones. A distinctive feature of declarative models is that there they are funded on the principle that "all that is not explicitly forbidden is allowed". In this sense tasks belonging to a single habit are more likely loosely coupled by precedence relations rather than by strong sequential assumptions (as the one assumed in workflow-like formalisms) with the habit performed in different ways by different users (in this sense an habit could be considered an *artful* process [HYJK06]) which decide to perform tasks within habit in a concurrent way or following different strategies execution by execution. In conclusion, habit mining is conceived as an hybrid solution between process mining and AI-based situation identification techniques.

2.2 Related Works: Situation Identification

Situation identification techniques can be roughly divided into *specification based* and *learning based* [YDM12]. Situation identification research started when a few sensors were available and the relationships between sensor data and situation was easy to establish. Specification based approaches (see Section 2.2.1) represent *expert* knowledge in logic rules and apply reasoning engines to infer situations from sensor data. These techniques evolved in the last years in order to take into account uncertainty. The growing availability of different kind of sensors made hand-made models difficult to define due to the big amount of produced sensor data. In order to solve this problem, learning based methods (see Section 2.2.2) employ techniques from *machine learning*, *data and process mining*.

2.2.1 Specification based methods

First approaches to the development of context-aware systems able to recognize situations where based on *predicate logic*. Loke [Lok04] introduced a PROLOG extension called LogicCAP; here the "in-situation" operator captures a common form of reasoning in context-aware applications, which is to ask if an entity E is in a given situation S (denoted as $S^*>E$). In particular a situation is defined as a set of constraints imposed on output or readings that can be returned by sensors, i.e. if S is the current situation, we expect the sensors to return values satisfying some constraints associated with S. LogicCAP rules use backward chaining like PROLOG but then utilizes forward chaining in determining situations, i.e. a mix of backward and forward chaining is used in evaluating LogicCAP programs. The work introduce different reasoning techniques with situations including selecting the best action to perform in a certain situation, understand what situation a certain entity is in (or the most likely) and defining relationships between situations. The same author [Lok10] later defined a formal methodology to compose different context-aware pervasive systems. The following script represents a situation program which follows this approach:

```
if in_meeting_now(E) then
   with_someone_now(E) ,
   has_entry_for_meeting_in_diary(E) .
if with_someone_now(E) then
   location*(E, L) , people_in_room*(L, N) , N > 1.
if has_entry_for_meeting_in_diary(E) then
   current_time*(T1) ,
```

```
diary*(E, 'meeting', entry(StartTime, Duration)) ,
within_interval(T1, StartTime, Duration) .
```

In the above situation program, a situation `in_meeting_now` of a user entity E is defined on two situations `with_someone_now` and `has_entry_for_meeting_in_diary`. Each of these situations has its own program that is defined on sensor predicates; for example, `with_someone_now` refers to two sensor predicates: `location*(E, L)` that returns the location of the entity, and `people_in_room*(L, N)` that returns the number of people in the location. In this way, situation programs are made amenable to formal analysis, and the inference procedure of reasoning about situations is decoupled from the acquisition procedure of sensor readings.

Henriksen [HI06] proposes a full-fledged architecture for a context-aware pervasive system. Here a set of models are associated to the context. These models are hand-made using a graphic approach called Context Modelling Language (CML). A so defined context model contains static facts as well as dynamic facts derived by sensors. All these models work as an ontology to be used by situation models defined using a customized predicate logic. Noticeably, this work introduce a techniques to deal with user preferences.

Other logic approaches were derived from *situation calculus*. Multi-level situation theory (MLST) has been exploited by Kalyan in [KGS05]. Major notions of situation theory are *infons* and situations. Infons are fundamental units, which embody discrete units of information. Situations make certain infons factual and thus support facts. S supports a, (symbolically represented as $S \vdash a$), means that a is an infon that is true for situation S. In the proposed MLST approach, authors introduce one more level in between infons and situations termed as *Micro Situations*. Micro situations and situations are defined to be hierarchically related and hence, situations can be thought of as a hierarchical aggregation of situations and micro situations.

Situation identification techniques relies on an explicit or implicit reference to where and when a certain situation occurs. Thus may approaches for situation identification directly exploits *spatial and temporal* logic. Authors of [GGH06] presents a brief survey of techniques for *temporal calculus* (namely Allen's Temporal Logic and Point Algebra) and *spatial calculus*. Activity patterns are also used as a mean to combine time and space in order to recognize unusual situations. Augusto [AN04] introduces the concept of Active DataBase (ADB) composed by Event-Condition-Action (ECA) rules which take into account time. This work has been extended in [ALM+08] with more complex temporal operators (`ANDlater` and `ANDsim`) and adding uncertainty management.

Ontologies have increasingly gained attention as a generic, formal and explicit way to 'capture and specify the domain knowledge with its intrinsic semantics through consensual terminology and formal axioms and constraints' [YCDN07]. They provide a formal way to represent sensor data, context, and situations into well-structured terminology, which makes them understandable, shareable, and reusable by both humans and machines. A considerable amount of knowledge engineering effort is expected in constructing the knowledge base, while the inference is well supported by mature algorithms and rule engines. Up to 2007 [YCDN07], most relevant systems based on ontologies were CoBrA [CFJ03] (based on OWL), Gaia [RMCM03] (based on DAML + OIL) and SOCAM [GPZ05] (also called CONON, based on OWL). The following example shows how SOCAM define the situation of 'sleeping':

```
(?user rdf:type socam:Person),
(?user, socam:locatedIn, socam:Bedroom),
(?user, socam:hasPosture, 'LIEDOWN'),
(socam:Bedroom, socam:lightLevel, 'LOW'),
(socam:Bedroom, socam:doorStatus, 'CLOSED')
-> (?user socam:status 'SLEEPING')
```

Ontologies have been also used in the analysis of mainly visual scenes [NM08]. Fundamental features of ontologies to be used in a pervasive systems are analysed in [SKDN09].

Another example of using ontologies in situation identification is given by [RB09]. Instead of using ontologies to infer activities, they use the ontologies to validate the result inferred from statistical techniques as those introduced in Section 2.2.2. This use of ontologies as a validation tool could be very interesting also for habit mining in order to represent static knowledge.

While ontology-based methods are mainly specification based, a lot of effort has been put in obtaining them through mining (see Section 2.2.2) using *crawlers* or *mining tools*.

Managing uncertainty and evidences

Sensors deployed into smart areas are affected by errors. The theory of *fuzzy sets* [Zad65] is widely used to deal with uncertainty of vagueness, which is represented by a membership function which denote to what degree an element belongs to a fuzzy set. A probability value reflects the reliability of data, which can be provided from the domain expert, or obtained from the user experience. It allows imprecise knowledge to be modelled, such that an approximation of a numerical value, or a vague information is expressed. For example in the domain of temperature, fuzzy functions on temperature terms like 'cold', 'lukewarm', and 'warm' can be mapped to a (possibly overlapping) range of temperature degrees.

Fuzzy Logic theory has been employed for example in [ANH07]. Here authors apply fuzzy inference on a conceptual hierarchy of situational context ontologies. The inference process is to find a situation that is most similar to the current unknown situation by evaluating the similarity of the specifications of situations in the knowledge base and the current context input. The aim is to calculate the *situational involvement*, that is the degree of belief that a user is involved in a recognized/estimated situation, applying a fuzzy function.

Fuzzy rules are also employed, by [CLMC12]. In this framework, authors propose a robust and general rule-based approach to manage situation awareness. Semantic Web reasoning, fuzzy logic modelling, and genetic algorithms are used to handle, respectively, situational/contextual inference, uncertain input processing, and adaptation to the user's behaviour.

Dempster-Shafer theory (DST) [Sha76] [SFL02] is a mathematical theory of evidence, which propagates uncertainty values and consequently provides an indication of the certainty of inferences. The core concepts of DST are the mass functions, the mass functions, the frame of discernment and combination rules. Mass functions distribute belief from sensors across choices or hypotheses in the frame of discernment. The combination rule is used to fuse belief from multiple sources.

As a generalised probability approach, DST has distinct features: it quantifies and preserves ignorance due to the lack of information and it aggregates beliefs when new evidence is accumulated. One of its important aspects is the combination of evidence obtained from multiple sources and the modelling of conflict between them. As for the model itself, the significant innovation of DST is that it allows for the allocation of a probability mass to sets or intervals. The process of using DST is described as follows. First of all, developers need to apply expert knowledge to construct an evidential network that describes how sensors lead to activities. Second, developers need to determine the evidence space and degree of belief for each evidence. Then the degrees of belief on sensor observations will be translated into the degrees of belief on the associated object context node by using the multi-valued mapping. The context can be lifted to higher-level context, while the mass function on the original context will be propagated to the higher-level context through an evidential mapping.

Figure 2.2 depicts an example DST evidential network taken from [HNM+09]. Here we have two sensors `sfri` and `scup`, which are directly connected to respectively `fridge` and `cup` objects. When the fridge sensor is triggered, the state of the fridge context is changed which indicates the inhabitant interacts with the fridge (opening the fridge and getting food out of the fridge). However, it is not possible to infer what food is removed from the fridge by simply considering the current state of the fridge door. If the inhabitant wants to make a cold drink, it is more likely that the juice is removed. Then we have a composite object, `(cap,juice)`, which has two compulsory context objects namely `cup` and `juice`. This composite object is in turn compulsory for `makingColdDrink`

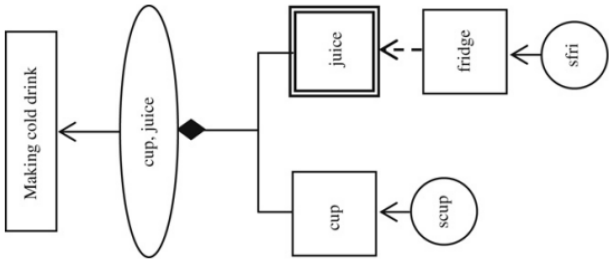

Figure 2.2. En example DST evidential network.

situation. Using DST theory (see [HNM+09] for details) it is possible to estimate the belief of a certain situation depending on the context state.

An additional approach is represented by *probabilistic description logic*. For example [HRN+12] exploits Log-linear DLs to model both probabilistic and deterministic dependencies between DL axioms through extending uncertain axioms with weights. Regarding the expressiveness of the language, Log-linear DL supports the same operators of the OWL 2 language.

2.2.2 Learning based methods

Specifying and identifying situations can have a large variability depending on factors such as time, location, individual users, and working environments. This makes specification-based approaches relying on models of a priori knowledge impractical to use. Machine learning techniques [Bel06] have been widely applied to learning complex associations between situations and sensor data. While in the vast majority of cases supervised learning is employed, manually labelling the huge datasets needed to obtain effective models, can be impossible. Thus, recently, a great interest arises for techniques employing weakly supervised learning (e.g. [SS09]) and unsupervised learning (e.g [GCTL10]).

Bayesian Derivatives

Bayesian classification techniques are based on the well known Bayes' theorem $P(H|X) = \frac{P(X|H)P(H)}{P(X)}$. In the case of situation identification, H denotes a situation, X represents the set of evidences (values of context attributes) we have obtained from the sensors. Calculating $P(X|H)$ can be very expensive so a set of different assumptions can be made. *Naïve Bayes* for example, is a simple classification model which suppose the n single evidences composing X independent given the situational hypothesis, that is $P(X|H) = \prod_{k=1}^{n} P(x_k|H)$. The inference process within the naïve Bayes is straightforward; that is, to choose an activity with the maximum posteriori probability. The learning methods in this class of techniques are mainly *supervised*.

When there exist dependencies between attributes, Bayesian (belief) networks can be applied to substitute naïve Bayes, whose performance degrade in this condition. A Bayesian network is a directed acyclic graph in which each node represents a variable that can be discrete or continuous, and each arc is the causal relationship between nodes. If there is an arc from a node A to another node B, then A is called a parent of B, implying that the variable B is regarded as depending directly on A. In this sense, naïve Bayes models can be viewed as Bayesian networks in which each input attribute has an output classification as the sole parent and the classification has no parents.

Bayesian network has been often employed to add uncertainty management to ontology based situation reasoner. This approach has been for example employed in [GPZ+04] to the already cited SOCAM system. Here each context predicate present in the ontology is mapped onto a node of the BN and an arc is drawn between two nodes if a dependency relation exists (so the dependency between two context predicate is hand-made and represented using an ontology property). After

the BN is created, it is trained (supervised learning) on real data to get probability distributions for the various nodes.

As presented, Bayesian networks provide a clear and well understood method for incorporating how the likelihood of a possibility of an event is conditioned on another event. They are best suited to applications where there is no need to represent ignorance, where conditioning is easy to extract through probabilistic representation and prior odds are available. Similar to naïve Bayes, they may lack credibility due to the unavailability of estimates.

Hidden Markov Models (HMMs) are a statistical model where a system being modelled is assumed to be a Markov chain that is a sequence of events. An HMM is composed of a finite set of hidden states (e.g., s_{t-1}, s_t, and s_{t+1}) and observations (e.g., o_{t-1}, o_t, and o_{t+1}) that are generated from states. HMM is built on three assumptions: *(i)* each state depends only on its immediate predecessor, *(ii)* each observation variable only depends on the current state, and *(iii)* observations are independent from each other. In an HMM, there are three types of probability distributions: *(i)* prior probabilities over initial state $p(s_0)$; *(ii)* state transition probabilities $p(s_t|s_{t-1})$; and *(iii)* observation emission probabilities $p(o_t|s_t)$. The training of an HMM is performed using the Baum-Welch algorithm while situation identification is performed using the Viterbi algorithm. Usually [VKNEK08] the observable variables are vector of measurements.

A problem with the use of a standard HMM for activity detection is that due to the first order Markov assumption (a state depending only on the previous one) the probability of an event state being observed for a certain interval of time declines exponentially with the length of the interval. Also the probability that there is a change in the hidden state does not depend on the amount of time that has elapsed since entry into the current state, which could be an important parameter in modelling human activities. To model the prior duration of an event state, HMM can be extended to *semi-HMM*, where the hidden process is semi-Markovian. This semi-HMM performs better at approximating visual primary and composite activities.

Another drawback of using a standard HMM is its lack of hierarchical modelling for representing human activities. To deal with this issue, several other HMM alternatives have been proposed, such as *hierarchical* and *abstract* HMMs. In a hierarchical HMM each of the hidden states can be considered as an autonomous probabilistic model on its own; that is, each hidden state is also a hierarchical HMM. In [NPVB05] for example, the performance of this technique is compared to that of a flat HMM. In this work every HMM contains a special 'end' state which returns the control to the parent HMM.

In an abstract HMMs the bottom part of this model consists of hidden states and observations as in a typical HMM, but the states are linked to abstract policy (or activity) variables which are placed in a hierarchy. Flag variables are used to indicate whether the current policy or activity continues or terminates in the next time step [BVW02].

HMMs generally assume that all observations are independent, which could possibly miss long-term trends and complex relationships. *Conditional Random Fields* - CRFs, on the other hand, eliminate the independence assumptions by modelling the conditional probability of a particular sequence of hypothesis, Y , given a sequence of observations, X; succinctly, CRFs model $P(Y|X)$. Modelling the conditional probability of the label sequence rather than the joint probability of both the labels and observations $P(X, Y)$, as done by HMMs, allows CRFs to incorporate complex features of the observation sequence X without violating the independence assumptions of the model. The graphical model representations of an HMM (a directed graph) and a CRF (an undirected graph) makes this difference explicit. For example in [VKNEK08] a comparison between HMM and CRF is shown, where CRF outperforms HMM in terms of timeslice accuracy, while HMM outperforms CRF in terms of class accuracy.

Decision Trees, Neural Networks and Support Vector Machines

A decision tree (DT) is a predictive model where each leaf represents a classification and each branch represents a conjunction of features that lead to the target classifications. A decision tree is built on

information entropy in that its construction works by choosing a variable at each step that is the next best variable to use in splitting the set of items. Compared to Bayesian models, a decision tree does not take into account the dependence assumption or potential temporal sequence between classifications. In [BI04] a set of features is extracted from the data acquired from a set of wearable sensors (accelerometers) in fixed size time slots. Different ML techniques are compared to DTs which however results in the best performance.

Support Vector Machines (SVM) allow to classify both linear and nonlinear data. An SVM uses a nonlinear mapping to transform the original training data into a higher dimension. Within this new dimension, it searches for the linear optimal separating hyperplane that separates the training data of one class from another. With an appropriate nonlinear mapping to a sufficiently high dimension, data from two classes can always be separated. SVMs are good at handling large feature spaces since they employ overfitting protection, which does not necessarily depend on the number of features. An interesting approach is presented in [BCR09] where a mixed supervised/unsupervised learning method is employed. First SVM (supervised) are employed to learn roles, then an unsupervised technique (Jeffrey divergence) is employed to extract situation models, and last SVM are employed again to improve discovered models. [SLES11] makes use of Multi-Instance SVM (a weak supervised learning techniques based on SVM) as it is robust to labelling noise and can achieve competitive classification using only few labels (only a little part of the dataset has been annotated).

Artificial neural networks (ANNs) are a sub-symbolic technique, originally inspired by biological neuron networks. They can automatically learn complex mappings and extract a non-linear combination of features. A neural network is composed of many artificial neurons that are linked together according to a specific network architecture. A neural classifier consists of an input layer, a hidden layer, and an output layer. Mappings between input and output features are represented in the composition of activation functions f at a hidden layer, which can be learned through a training process performed using gradient descent optimisation method or resilient backprogagation algorithms. Additionally methods for *unsupervised* learning can be used with ANNs.

Pattern Mining

Pattern mining is a data mining method that involves finding existing patterns in data. In this context patterns often means *association rules*. Being the temporal dimension a major component of situations, temporal pattern mining, in particular, shows a great potential for developing context-aware systems. Temporal pattern mining is strictly connected with the already discussed spatio-temporal representation methods, but here models are automatically obtained using mining instead of being hand-made. An introduction to temporal pattern mining (or temporal data mining) can be found in [GPR06]. Here input data is divided into *sequences* (which correspond to a sequence of events) and *time series* (which correspond to time-continuous functions) and a set of method is introduced for both supervised and unsupervised mining.

A strict correlation does exist between temporal patterns and business process, so business process mining and pattern mining show many similarities [FLB09]. For example in [AIB+10], a temporal pattern mining is obtained from a workflow mining algorithm.

Another interesting approach to pattern mining is given by *suffix trees*. A suffix tree enumerates all unique subsequences of events in a certain activity, where any subsequence of events from the root node to a non-root node is called a event motif. [HMJ+09] presents an *unsupervised* framework to discover temporal patterns of finer-grained events in everyday human activities. Here an activity is defined as a finite sequence of events, while an event is defined as a particular interaction among a subset of key objects over a finite duration of time. A key object is an object present in an environment that provides functionalities that may be required for the execution of an activity of interest. The idea is to extract events from interactions between human users and key objects, organise events into a Suffix-Tree, and mine the patterns of events in each activity.

Other

Obtaining training data represents a major impediment to applying machine learning techniques (see Section 2.4.4). An interesting approach to cope with this issue is based on *web mining* techniques (e.g. [PPG+10])which extract key terms for each targeted activity from its collected online 'howto' documents, and determine each term relevance weight for each object. The assumption underlying this approach is that in most activities of daily living, the lists of relevant objects for a particular activity are similar and do not vary significantly even when the activity is performed in different ways.

In case it is possible to obtain a critical mass of data to apply machine learning techniques, it remains difficult to label all this information to perform supervised learning. We have already introduced some methods which employ weak or non supervision. Other techniques include *emerging patterns* (e.g. [GCTL10]) and the already mentioned *Jeffrey divergence*

2.3 Related Works: Declarative Process Models

As discussed [AIB+10], learning a habit is very similar to mine a process. Both domains are indeed very similar except processes are based on actions of the user instead of events. Still, the objectives are very closely related in both domains. Even so, due to the nature of smart spaces, we have considered that process mining algorithms should be modified in various aspects (see Section 2.4).

Most of classical process management environments for the specification of processes are based on imperative languages and procedural models (such as that used in [AIB+10]), such as YAWL nets, WS-BPEL and BPMN. Procedural process models explicitly define the control-flow of activities, i.e., the schedule and ordering of tasks to be executed, and process executions are driven by the flow of control prescribed by the corresponding models. As a result, many possible execution options and potentially allowed paths not explicitly included in the control-flow are not admitted, as they would violate the strict constraints imposed by the models. On the one hand, an imperative and procedural modelling approach is well suited for processes where tasks need to be repeatedly executed in a predefined and controllable manner, and where all possible interactions among involved entities (people, devices, services, etc.) can be captured and specified in detail by the modelling capabilities and semantics of procedural languages. On the other side, procedural models are not able to provide the degree of flexibility required in many other settings and they may unnecessarily limit possible execution behaviours. When trying to define a process using a procedural model, the designer is forced to either capture all possible allowed executions in an attempt to enhance flexibility, or to make some a priori assumptions and choose (and thus impose at design-time) a restricted subset of allowed executions. As a consequence, produced models may become either over-specified and thus complex to understand and manage, or over-constrained, as "all that is not explicitly modelled is forbidden".

Recent and ongoing work shows instead that *declarative* languages and models can be effectively used to increase the degree of flexibility for process specifications. The definition of these languages was inspired by the observation that, in a process, possible execution alternatives can be classified as: *(i)* forbidden, i.e., executions that are not allowed and should never occur; *(ii)* optional, i.e., executions that, although should be avoided, are still possible and admitted; *(iii)* allowed, i.e., the intended normal executions. Instead of strictly and rigidly defining the control-flow of process tasks using a procedural language, such languages aims at supporting the identification and specification of a (minimal) set of policies and business rules (i.e., constraints) to be satisfied and followed in order to successfully perform a process and achieve the intended goals. This approach results in a shift to "all that is not explicitly forbidden is allowed". This feature makes declarative models more suitable than procedural ones to model habits.

Throughout this section we will introduce LTL and the main declarative specification language derived from it, namely DECLARE, and then we will shortly discuss declarative process mining. For an extended comprehension of the topic we suggest the work by Montali [Mon10].

2.3.1 From LTL to DECLARE

Temporal logics are a special class of modal logics where modalities are interpreted as temporal operators, used to describe and reason about how the truth values of assertions vary over time. In this class, LTL (Linear Temporal Logic) can be considered as being: *(i)* propositional, as formulae are defined from atomic propositions, whose truth values change over time; *(ii)* linear, as temporal operators predicate on the truth of propositions along a single timeline and each moment has a single next future moment; *(iii)* qualitative, as temporal operators are used to express qualitative time relations between propositions; *(iv)* point-based, as temporal operators and propositions are evaluated over points in time; *(v)* discrete, as the present moment corresponds to the current state of the system and the next moment to the immediate successor state induced by the occurrence of an event (i.e., time is discrete); *(vi)* future-tense, as temporal operators predicate on the occurrence of events in the future. Basically, LTL formulae are defined using atomic propositions (with true and false constants), propositional connectives ($\neg$, $\wedge$, $\vee$, $\Rightarrow$), and temporal operators ($\bigcirc$ next time, $\Box$ globally, $\Diamond$ eventually, $\mathcal{U}$ until).

The semantics of LTL is defined with respect to an LTL model in a specific state and temporal formulae are interpreted in a discrete, linear model of time. Given the set $\mathcal{P}$ of atomic propositional formulae, an LTL model is represented by $\mathcal{M} = \langle \mathbb{N}, I \rangle$, where $I : \mathbb{N} \mapsto 2^{\mathcal{P}}$ maps each moment in time (represented by a natural number) to a set of propositions that represents all the propositions $p \in P$ that hold in that moment in time. From an other perspective, an LTL model or interpretation of an LTL formula can be considered as an *infinite* trace $\pi = \pi_0, \pi_1, ...$ having $\mathbb{N}$ as time structure and P as the set of atomic propositions; each element $\pi_i \in 2^{\mathcal{P}}$ of the trace is defined by I and thus refers to the set of propositions that hold at the i-th moment in time (i.e., $\pi_i = I(i)$). At some time point $i \in \mathbb{N}$ a proposition p is true iff $p \in \pi_i$. $\langle \pi, i \rangle \models \varphi$ means that a formula φ is true at time i in π (i.e., trace π satisfies φ), and $\models$ denotes the logical entailment (or satisfaction relation).

LTL can be used to provide an underlying semantic to the DECLARE graphical formalism [PSvdA07][1]. Instead of rigidly defining the flow of interaction, DECLARE focuses on the (minimal) set of rules which must be satisfied in order to correctly carry out a business process. A DECLARE model is defined as a triple $\mathcal{CM} = \langle T, \mathcal{C}_m, \mathcal{C}_o \rangle$, where *(i)* T is a set of tasks, represented as boxes containing their name; *(ii)* $\mathcal{C}_m$ is a set of mandatory constraints; *(iii)* $\mathcal{C}_o$ is a set of optional constraints.

DECLARE constraints, represented using arrows which connect task boxes and annotations, are grouped into four families: *(i)* existence constraints are unary cardinality constraints expressing how many times an activity can/should be executed; *(ii)* choice constraints are n-ary constraints expressing the necessity to execute some activities between a set of possible choices, independently from the rest of the model; *(iii)* relation constraints are binary constraints which impose the presence of a certain activity when some other activity is performed, possibly imposing also temporal constraints on such a presence; *(iv)* negation constraints are the negative version of relation constraints, and are employed to explicitly forbid the execution of a certain activity when some other activity is performed.

Given the set T of tasks defined in a DECLARE model $\mathcal{CM}$, an execution trace $\sigma \in T^*$ is a finite sequence of tasks $\langle \sigma_0, \sigma_1, ..., \sigma_{n-1} \rangle$, where T^* is the set of all possible traces composed of zero or more elements (tasks) of T. $|\sigma| = n$ is the length of the trace, and σ^i is used to denote the suffix of σ starting at i, i.e., $\sigma^i = \langle \sigma_i, \sigma_{i+1}, ..., \sigma_{n-1} \rangle$.

All the DECLARE constraints can be formalized in LTL and in addition, it can be extended as long as added constraints can be represented using LTL. However two main differences may be found between an LTL model $\mathcal{M}$ and an execution trace of a DECLARE model $\mathcal{CM}$. First, LTL considers infinite traces, whereas the execution of a process model generates a finite execution trace as a sequence of executed tasks: the temporal dimension in process executions is bounded, and

[1] There exists different specializations of DECLARE language, including the basic ConDec, and DecSerFlow, which models service flows, compositions and choreographies. Both languages/applications share the same concepts and tools.

therefore their execution traces are always finite. Second, each element of an LTL trace can refer to a set of propositions, whereas each element in a finite execution trace of a DECLARE model refers to exactly one event corresponding to the execution of a task t i , i.e., only one proposition holds at one moment. The first issue (finite execution trace) can be solved either using *LTL for finite traces* or adding a *termination property*.

Noticeably every LTL formula, can be translated into a Büchi automaton [RB66], a non-deterministic finite-state automaton which takes as input infinite traces (words), using the algorithm proposed in [GH01]. As a consequence it is possible to verify and enact a DECLARE model using techniques devised for Büchi automata.

2.3.2 Static Verification and Conformance Checking

A tool is available for DECLARE language providing support for the specification, enactment, verification, monitoring and analysis of constraint models.

Static *verification* techniques aim at providing design-time support for guaranteeing that a constraint model will meet, when executed, one or more desired properties, enabling the early identification of errors and undesired behaviours. This requires a sound and complete formal verification procedure able to prove, or disprove (and provide a counterexample), the correctness of a given model with respect to some property. Properties to be checked against a model can be formally specified to capture different requirements that must be satisfied by the modelled process. Specifically, properties over a DECLARE model can be specified using the Declare language itself, i.e., as LTL formulae.

Two or more constraint models can be composed to build a single process model which combines their constraints. A composite model can thus be obtained by combining the activities and constraints of the single models to be composed. For a composite model, static verification aims at verifying whether the composition can be effectively performed, i.e., verifying whether the single models are compatible. The verification of compatibility of constraint models can be reduced to the verification of conflict-freedom of the resulting composite model.

The possibility of *(i)* generating an automaton for each constraint in a model, and *(ii)* generating an automaton for the conjunction of all constraints in the model, enables the *enactment and monitoring* (conformance checking) of constraint models and their instances [GH01]. The main goal of run-time verification is checking if the actual behaviour of an executing instance effectively complies with all the modelled constraints, in order to identify unwanted execution behaviours. Run-time verification is performed on evolving instances that generate events (corresponding to task executions) collected in an execution trace. As a consequence, the observed execution trace is only a portion of the whole instance (i.e., a partial trace), and therefore a definitive answer on the compliance with modelled constraints can not be always provided. Given an instance trace, the state of a constraint in $\mathcal{CM}$ and of the model instance can be checked by verifying if the trace satisfies the constraint and the model, i.e., if the corresponding automata accept the trace. For an active instance, each constraint can be in one of three states: satisfied, violated, or temporarily violated (i.e., the constraint is currently violated but could still be satisfied by executing other tasks in the model).

While enactment can be used for dataset generation (see Section 2.4.4), we will discuss that, for habit mining, we are not interested in verifying if the current execution is conform to a certain habit model. Instead we would like to understand how much the current event log resembles each of the habit models available in the system repository; in other words for each habit model we would like to have a similarity score between the event log and the model. Following this approach an habit model can be considered as a fuzzy set and event log can show different degree of similarity with them instead of being divided into conform and non-conform instances.

2.3.3 Declarative process mining

Process mining techniques often reveal that real-life processes are more variable than anticipated. Although declarative process models are more suitable for less structured processes, most discovery techniques generate conventional procedural models. Recently an approach has been proposed in [MBvdA12]. Here a technique based on both *vacuity detection* and a form of *Apriori* algorithm. Vacuity detection can be used to prune set of constraints, but this can only be done after generating a set of candidate constraints. As discussed in the introduction, even for smaller processes, there can be millions of potential constraints. Therefore, the basic idea is the well-known Apriori algorithm for discovering association rules. Apriori-like approaches can efficiently discover frequent sets of correlated activities in an event log.

Recent years witnessed a growing interest in the so called *artful* processes, that is informal processes typically carried out by those people whose work is mental rather than physical (managers, professors, researchers, engineers, etc.), the so called "knowledge workers". Artful processes can be modelled using declarative approaches. A miner for these kind of processes has been proposed in [DCM12a].

As we will see in Section 2.4, habit mining is mainly focused on model enhancement which is a particular type of process mining. Here, the idea is to extend or improve an existing habit model using information about the actual situation recorded in some event log. Thus habit mining aims at changing or extending a-priori habit models in an incremental way.

2.4 Habit Mining: Research Agenda

As discussed in Section 2.1, a situation is a high-level interpretation of the context, this latter intended as a well-structured digest of sensor data. Situations we are interested in are those derived from the execution of human habits, thus our goal is inferring situations basically comparing the last events recognized to a set of habit models. These events are not those generated by sensors (called unit events); rather we take into account *derived events* which integrate many unit events. A good habit model should take into account peculiar features of human behaviour, namely *temporal* and *spatial* dimension and *uncertainty*.

As pointed out in [CAJ09] very little can be done without an explicit or implicit reference to where and when the meaningful events occurred. For a system to make sensible decisions it has to be aware of where the users are and have been during some period of time. These insights, together with other information, will provide important clues on the type of activities the user is engaged in and the most adequate response. Both dimensions, space and time, are useful to understand key elements of a situation under execution. Taking into account space can be done only supposing a symbolic knowledge of the space, where beside a physical position we have devices and persons which are nearby.

Uncertainty influences habits at different levels. At the lowest level imprecision of sensors can negatively influence habit detection, with the error propagating to the derived events (secondary context). At an higher level, due to the fact that different user performs habits in different ways and even the same user may use different strategies performing an habit, it is impossible to define firm constraints, so affiliation of a certain context to a situation model cannot be sure, but has to be specified using a degree of confidence (e.g. this context is 60% similar to a certain situation). Additionally recent sequences of derived events could be conform to different situations; this is a not negligible possibility because of the fact that there exists different possible relations between situations [YDM12]. For example a situation could be *(i)* regarded as *more general* than another, *(ii) decomposable* into smaller situations, *(iii) mutually exclusive* with other situations or *(iv)* in temporal sequence with other situations (occurring after, before or interleaved with).

Additionally, human behaviour as well as the environment evolve over time. New habits and scenarios emerge in a smart environment, and others change or disappear. New sensors must be integrated into the environment, whereas obsolete ones should be deleted. Experts can construct

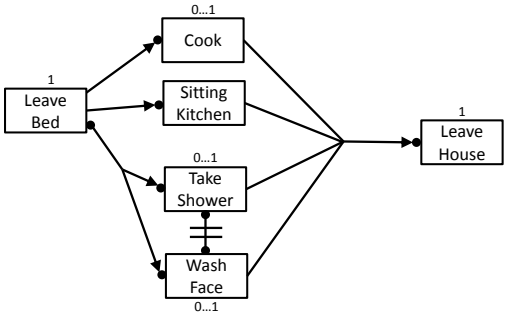

Figure 2.3. The case study habit DECLARE model.

and adapt habit models according to the changing needs of users and application. However, experts are expensive and not always available. Moreover, the environment's intelligence lies in its ability to adapt its operation to accommodate the users. The research challenge is thus to develop machine learning methods for this process, making it possible to automatically acquire and evolve habit models reflecting user behaviour and needs in a smart environment.

The final goal of our research is a framework able to perform *habit mining* intended as continuous conformance checking and model enhancement. The systems starts from a set of hand-made (by experts) declarative habit models (expressed using DECLARE language) (see Section 2.4.1), continuously compare them with the recent performed actions, and choose the most conform one enhancing it with the information extracted from the sensors (see Section 2.4.3). Necessary condition to pursue this goal, is extracting derived events through log segmentation and action recognition (see Section 2.4.2).

Validating a similar framework need a massive dataset which is impossible to obtain from a real smart space because *(i)* gathering it would require years and *(ii)* it would be impossible to incrementally add new sensors in order to test algorithms performance over increasingly information rich datasets. Frequently this problem is coped developing a simulator (e.g. NS2 for computer networks) so a realistic dataset generator (see Section 2.4.4) is currently under development.

2.4.1 Representing Habits: the `MorningRoutine` Habit

The `MorningRoutine` habit is intended as the set of actions which are executed after leaving the bed and before leaving the house by a single person (the house may be at any time hosting more than a single person). The morning habit starts whenever a person leave the bed and ends up when the house has been left. In the meanwhile the person can cook something (at most once), sitting at the kitchen table (how many times he wants). The user either take a shower or wash his face before leaving the home.

Declarative approaches have been employed for situation modelling because of their readability and of the reasoning features [Lok06]. Additionally declarative models are very user-friendly for expert to represent knowledge. The similarity between business processes and habits have been already discussed, thus *declarative process models* represents an interesting alternative to represent and reason about situations and habits. However, business processes are based on event coarser grained than those produced by raw sensors. This level is constituted by *derived events* which are obtained processing raw sensor events.

`MorningRoutine` habit may be represented by the DECLARE model depicted in Figure 2.3. Here a branching `succession` constraint impose that after leaving bed the user has to either wash his face or take a shower (not both as imposed by the `not_coexistence` constraint). Two `precedence` constraints impose that the user could decide to cook something and/or to sit at the table a little bit. An additional branching `precedence` constraint imposes that at least one of the cited actions has to be performed before leaving the house. While initial and final actions have to be performed

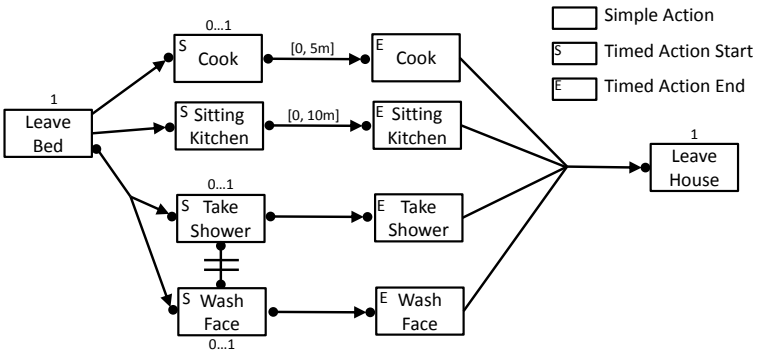

Figure 2.4. The habit DECLARE model enriched with a quantitative notion of time.

exactly once, all the other actions are optional; the `SittingKitchen` action can be performed an arbitrary number of times.

An user executing a habit produce a sequence of derived events. Considering derived events on a per-user basis as described in Section 2.4.2 additionally allows to think in terms of *multi-user* habits. While DECLARE provide a way to express habits flexibly, it lacks of the features we have previously identified as necessary to model habits.

As a first issue, LTL (and thus DECLARE) features a qualitative notion of time whereas some times we need a more precise and *quantitative* description of the duration of the phases which compose the execution of a habit. For example in some cases we need to express a typical duration for derived events while in other cases we need to express the fact that two derived events are typically separated by a certain amount of time. The possibility of expressing separate events for event start and event conclusion is a best practice in business process modelling. However a quantitative notion of time is not included in LTL.

LTL has been extended with time-bounded operators in [HMP94]. As a consequence it is to extend DECLARE in such a way to support the same kind of reasoning. For example, a DECLARE extension called *Timed DECLARE*, recently introduced in [WM12], is based on Metric Temporal Logic (MTL) and additionally allows for runtime verification. In particular, a Timed DECLARE model allows to give user alerts about the process going on.

Figure 2.4 represents the Timed DECLARE version of the habit model depicted in Figure 2.3. Here derived events has been divided into *instantaneous* and *durable*. Durable derived events have been split into a start event and an end event. It is worth to note how derived event recognition technique has to support this feature. In Figure 2.4 a typical duration is specified for `Cook` and `SittingKitchen` actions.

2.4.2 Unit Events, Derived Events, Knowledge and Log Segmentation

Suppose we have a house provided with a set of sensors which includes an indoor localization system `LS`, a temperature sensor `TS` installed in the bathroom, three switch sensors `OS`, `DS`, and `SS`, respectively installed into the oven, at the main entrance door of the house, and to the sink.

A single sensor measurement (called *unit event*) has the form $\langle src, t, val \rangle$ where *src* denotes the source sensor, *ts* is the timestamp (referred to a common global clock) and *val* is the value of the measurement. The value of a unit event can be either a simple data type or a complex type (such as a XML string or a tuple). For example *val* may be of the form $\langle usr, posX, posY \rangle$ for a localization system (such as `LS`), $s \in \{0, 1\}$ for a switch sensor (such as `OS`, `DS` and `SS`)and $t \in \mathbb{R}$ for a temperature sensor (such as `TS`). The *sensor log* produced by the house is a sequence of such unit events.

The DECLARE model of Figure 2.3 contains a set of coarse grain events which has to be inferred from *unit events* present in the *sensor log*. In Section 2.1 we defined these events as part of the so

called *secondary context* because they are obtained processing unit events. In the following we will call these events *derived events* and they can represent actions (as in Figure 2.3) as well as other non trivial events (a fire has started). Derived events represent an intermediate level between raw sensor log and habits; this approach has many advantages; among them, techniques to infer them are usually *tolerant to errors* and imprecisions in sensor data. Uncertainty of sensors employed in indoor area has some peculiar features; in particular while a home represents a very controlled environment with respect, for example, to outdoor, sensors employed are usually very imprecise and impossible to model using some precise mathematical model. While techniques for derived events recognition are out of the scope of the research agenda, it is important to note how *sensor log* is composed by a set of unit events which generally are related to different derived events which can happen simultaneously due to the fact that, for example, a house is populated by many persons which can potentially execute different actions at the same time.

A derived event may be inferred using different techniques. Take as an example the `LeaveBed` and `SittingKitchen` actions; they could be recognized using the `LS` sensor simply verifying respectively that the position of a specific user is not in correspondence of the bed anymore or is staying for a long period near the kitchen table. In this case it is important to note how the knowledge of the environment plays a fundamental role; it might be necessary, in order to integrate unit events from a localization system, to know the correspondence between physical positions and *symbolic* positions. Another way of inferring these two actions could be exploiting vision based methods[2]. Other actions included in the figure can be inferred by a simple analysis of the correspondent sensor related events; for example the `TakeShower` action can be inferred after observing through `TS` an increased temperature into the bathroom. Approaches usually followed for action (and more generally for derived events) recognition, are similar to those employed for situation identification; a survey can be found in [AR11].

An intelligent house may employ different sub-systems at the same time to recognize derived events. The vast majority of the techniques however is intended to analyse only *relevant* user events. This means that the sensor log has to be preprocessed and segmented in order for this sub-systems to work. A derived-event recognition technique usually deals with a limited set of sensors, so the first kind of segmentation that can be done is *sensor-based*. Segmentation can be however performed with respect to other aspects. The segmentation problem is strictly related to that of *data association* and *multi-sensor data fusion*; this is a very well studied field of research due to the possible application to radars.

Actions, for example, are considered as *single user* derived events, thus an interesting way to segment the log is on a *per-user* basis. Sensors can be roughly divided into two classes, namely *user-related* and *environment-related*. User-related sensors are those whose measurements are directly connected with one specific user. This first category includes, for example, IMUs (Inertial Measurement Units), localization systems. This kind of sensors allows in some cases to recognize fine grain actions such as gestures or voice commands. Unit events generated by these sensors usually contains as part of the value an identifier of the source user. Environment-related sensors, instead, produce unit events which are related to the environment or to specific areas or devices contained inside it. This category includes, for example, switch sensors, temperature sensors and smoke sensors. A set of users has to be assigned also to unit events produced by environment-related sensors. In order to do this we could consider a smaller to bigger *topological scope* can be assigned to each of these sensors. For example, the topological scope of a switch sensor is the area from which it can be operated. In the same way the topological scope of a temperature sensor can be an entire room (for example the topological scope of `TS` is the bathroom). Unit events produced by environment-related sensors are assigned to all the users which are in the topological scope of the user. In order to apply this notion it is important that a localization system is available in the house.

Another useful way to segment the log is on a *temporal* basis. While some derived event

[2]The reader interested in vision-based approaches to action recognition can refer to [Pop10]

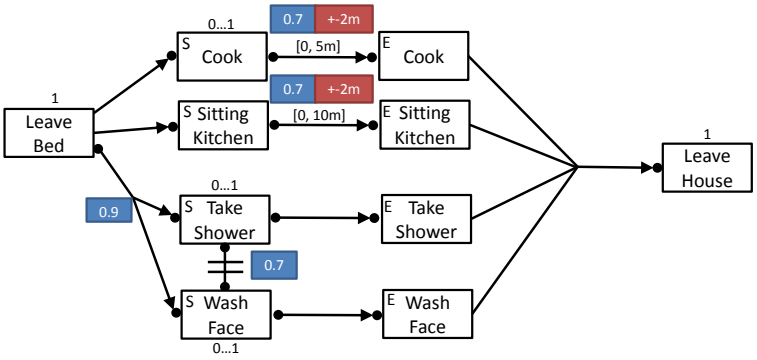

Figure 2.5. The habit Timed DECLARE model enriched with probability assertions.

recognition methods (such as those based on HMM) are able to deal with sequence of actions, some other methods need to analyse data processed in a certain time interval, which represents the typical duration of the action to recognize. So the sensor log can be segmented into overlapping chunks of fixed length.

2.4.3 Uncertainty, Conformance Checking and Model Enhancement

Once we have introduced the kind of events we are considering as important for habit recognition and we have discussed possible extensions to declarative models, we can inspect the idea underneath habit mining more into detail. We start from the assumption that we cannot rely on labelled data in order to mine our habit models and, as a consequence, they cannot be learned following a supervised strategy. On the other hand, it is not possible to apply a completely unsupervised approach because we could not be able to extract pattern from scratch due to the variety of information produced by sensors.

Our idea is to start from expert-made declarative models of interest habits and then incrementally enhance them using the data produced by the sensors at runtime. So we can describe habit mining as a technique which continuously performs *conformance checking* and *model enhancement*. In this sense, we can compare a context-aware system based on habit mining to a *recommender system*. The latter starts from a very little detailed model of the user (in some kind a model with errors) and then incrementally changes the model of the user using information extracted from visited pages.

Our basic idea is that a habit can be considered as a *fuzzy set*. The ratio behind this is that at any time we have for each habit $h \in H$ a model m_h which represents h with an unknown representation bias. While declarative models allow for many compliant traces, they are in any case based on rigid constraints. For example DECLARE constraints do not allow for exceptions and Timed DECLARE time intervals do not support any tolerance margin. However, habits resemble artful processes very closely, thus we cannot consider declarative constraints that strictly. Indeed every subject performs a habit in a way slightly different from other persons and may occasionally perform it with some little difference with respect to the routine.

Our approach associate to each constraint (from both DECLARE and Timed DECLARE languages) in the declarative model a probability value representing our confidence in it. The habit model depicted in Figure 2.5, for example, introduce some probabilities (for sake of simplicity omitted probability values can be considered equal to 1). It is worth to note how introducing probability could introduce defects into modelling. Take as an example the `succession` constraint if the figure; it includes, by definition, the `response` and `precedence` constraints, but if we add probability and we want to specify that one of them is obligatory, we have to explicitly add it to the model. Equivalently for each specified time interval we can specify tolerance margins are specified; in this case we can use a specific probabilistic distribution to express uncertainty about these margins.

At a given time we detect a sequence of recent derived events e. So, for each habit h we can calculate the probability that this sequence is coherent to it as $p(m_h|e) = \alpha p(e|m_h)p(m_h)$. In this equation $p(m_h)$ represents the a-priori probability that the habit h is executed; interestingly this probability may change during the day. For example it is more like `MorningRoutine` habit to be executed early in the morning. This information is part of the habit model itself. Instead $p(e|m_h)$ is the probability of seeing sequence e given a certain model of habit h. If we consider all constraints in m_h as independent one each other it is possible to obtain this probability simply following the step by step the constraints which are respected and those that are violated. As we suppose to associate to each constraint a certain probability p, respected constraints will contribute with a factor p to $p(e|m_h)$ while violated ones will contribute with a factor $(1-p)$. In doing this classic techniques for enactment and monitoring can be used to detect the set of violated constraints; particularly interesting in this sense is the capacity of Timed DECLARE to distinguish between different kind of violations (namely yellow, orange and red).

As derived events are detected our belief in different habits change. Now the question is when a certain sequence e can be definitely associated to a certain habit h. While a threshold can be used at this aim, this approach has to be balanced in order to avoid of taking a wrong decision only because we classified too early.

An interesting research direction could be to inspect the correspondence between such a model with other probabilistic model such as Dynamic Bayesian Networks. In this way conformance checking could be performed mixing up techniques for probability reasoning and for process checking.

Suppose now, we classified a certain sequence e as belonging to a specific habit h. We want to enhance m_h using information taken from e. The first data we can extract is about respected and violated constraints. If e has been confirmed as expression of m_h constraints respected by e will become more probable while those violated will become less probable. This variation of probability can be done on a statistical basis.

However in order to cope with changes, we need a way to incorporate new constraints and delete those which are not significant anymore. In particular for each sequence e we can annotate the model with potential constraints (a method to extract them can be found in [DCM12b]). Once an annotated constraint reaches a certain probability it can be integrated into the model. Updating the probability is done in a way similar to that used for constraints in the "official" model. It is worth to note here that integrating or deleting a constraint from a declarative model may be cause of inconsistencies which has to be checked (with methods from static verification); in this case an emergency strategy has to be devised.

2.4.4 Current Work: Dataset Generation

Although datasets generated by real houses are available from different projects (CASAS [3], MIT House_n [4]) they might not provide the critical mass of data needed for an extensive experimentation of algorithms for habit recognition. A solution to this problem may be found in realistic *dataset generation*. Furthermore dataset generation could provide a way to simulate new sensors without needing a real installation, allowing to compare algorithm performances with respect to a varying set of available indoor sensors which people interact with.

Let's imagine an indoor environment where multiple characters are moving around performing habits composed by very high-level operations, that is actions which changes the state of the environment, of the devices contained in it and of the character itself (assuming for example a particular pose). The environment contains, beside devices, a set of sensors which follow the execution of these habits but, in the vast majority of cases, cannot directly recognize basic actions. A basic example could be represented by a light switch device; a character is able to switch the light on and off, but an installed light sensor can only sense the illumination degree of the room.

[3] http://ailab.wsu.edu/casas/
[4] http://courses.media.mit.edu/2004fall/mas622j/04.projects/home/

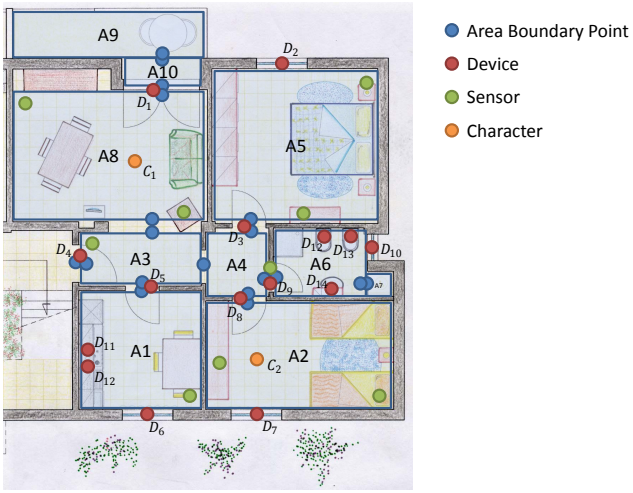

Figure 2.6. The employed indoor space model.

As another example, a character may fall on the ground but wearable inertial sensors could only provide numerical data about it.

Now, imagine we can reproduce the same environment into a simulation (see Figure 2.6). We suppose the house divided into a set of rectangular areas such that inner points are reachable one from each other. Different areas are connected through *area boundary points* (ABPs); leaving an area to enter in an adjacent one is only possible moving between to adjacent ABPs. Each area contains a set of objects (each of them has a specific dynamic position in the space) which can be divided into three *disjoint* classes, namely: *characters*, *devices* and *sensors*.

In order to generate a *realistic* dataset, the simulation environment should clearly separate the *execution* layer, which defines the sequence of actions that will be executed by characters, from the *logging* layer, which defines the set of sensors available for log generation. Applying the same planning layer to different logging layers will allow to produce datasets which differs for degree of detail. Realism is obviously in charge of the execution layer and can be, basically, obtained in two ways:

- Characters are driven by human users while performing habits. This strategy has the advantage of easily capturing the different strategies humans uses to perform habits. The cons is that is difficult, in this way, to rapidly obtain a critical mass of data;
- Plans are generated by the simulation environment, based on some, expert-made habit models. These models could be, for example, expressed using a declarative model. Indeed, while it would be difficult to hand-make a declarative model based on sensors output, this operation is easier if we can use high-level actions, as those that characters are enabled to execute into the simulation environment.

Characters driven using the former strategy are called *player characters* (PCs) while those following the latter, resumed in Figure 2.7, are called *non-player characters* (NPCs).

Each time an NPC has to perform an habit, a random model h is extracted from the *habit repository* $\mathcal{H}$. A habit modelled in a declarative model has many different habit instances conform to h, thus the execution layer randomly enacts one of them (see Section 2.3.2). The enactment of h produces a sequence of actions (called *h-actions*) to be performed in the environment; each of these actions impose some preconditions in order to be executed into the environment. For example, in order to perform an action over a specific device, a character must be in the same position of the device; additionally some devices may have particular features, such as a door needing a specific key in order to be open. In other words, in order to execute each single action during the enactment we need a *plan* for it where action execution is the final goal. This approach is generally called GOAP -

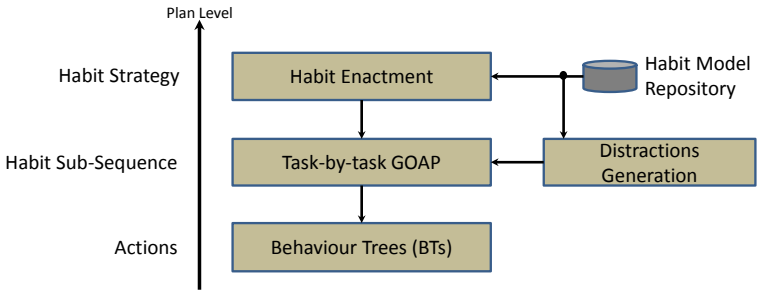

Figure 2.7. The dataset sample generation strategy

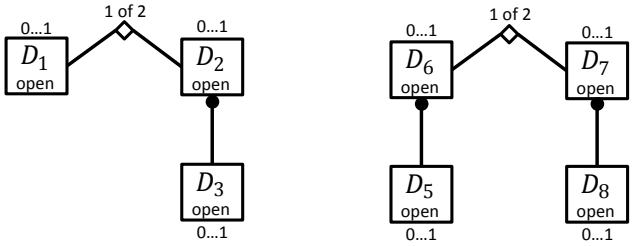

Figure 2.8. The `IncreaseAirFlow` habit DECLARE model.

Goal Oriented Action Planning [5]. Optionally, an entire subsequence contained into the enactment action sequence, composed by actions on which the habit model does not impose any precedence relation, may be inserted as final goal of a single plan.

In addition for each planning problem, containing the execution of a subsequence of actions as final goal, a set of randomly chosen irrelevant actions called distractions (not contained in the instance and compatible with the habit model) is added to the final goal. This mutation strategy introduces a set of random variations into the plan which, in turn, allow to improve dataset variability. Mutation strategy is designed to take into account the fact that, while performing an habit, a character may be distracted by something. For example the character could decide to close an open door or turn on a light. Mutation should take into account specific features of a character (such as laziness or disposition to distraction). For each planning problem generated by the execution layer a set of random actions is selected to be performed (distractions in the follows). These distraction has to not interfere with the ongoing habit, so, candidate distractions will be, for example, actions performed over devices not interested in the habit.

An additional degree of variability can be obtained exploiting *Behavior Trees* (BT) [6] connected to each action. For example each movement step may be executed in different ways following some random customization (for example not using a straight trajectory, taking a pause before the next action and so on). In addition BTs can be used to react to the presence of moving distractions such as other characters to interact with or a TV open to look at.

Consider as an example the `IncreaseAirFlow` habit model depicted in Figure 2.8 using the DECLARE notation. Tasks represented in the model refers to the example house described in Figure 2.6. The model is hand-made by an expert and impose, in order to execute the habit, to open at least one window for each side of the house. In particular windows D_1 and D_2 are candidate to be open in the upper side of the house, while windows D_6 and D_7 are candidate to be open in the bottom side of the house. For some windows (namely D_2, D_6 and D_7) the door of the room has to be open in order for the air flow to get through the house, but there is not an imposed order between opening a window and the corresponding door. Each task can be executed at most once.

A random instance of the `IncreaseAirFlow` habit is represented by the ordered opening sequence

[5] http://web.media.mit.edu/~jorkin/goap.html
[6] http://bjoernknafla.com/introduction-to-behavior-trees

$\langle D_1, D_6, D_5, D_7, D_8 \rangle$. Now a different GOAP problem is generated for different subsequence. So a GOAP problem will be generated in order to perform the task of opening D_1, one for $\langle D_6, D_5 \rangle$ and another for $\langle D_7, D_8 \rangle$. For each of these GOAP problems, the mutation strategy could decide to add some irrelevant tasks (such as closing the light in one room).

The logging layer is in charge of producing a log in the form we introduced in Section 2.1 for both PCs (manually driven by real users) and NCPs (driven by the execution layer). This log will contain the events generated by the set of virtual sensors available in the current simulation environment. How this log is various and realistic will depend on how much the planning layer is able to produce variegate habit instances.

Repeating this strategy many times for different user performing different habits will produce a huge dataset to be used for habit mining. Additional improvements to the dataset generation algorithm could be made allowing a single subject for the execution of multiple habits at the same time. In this case an additional design constraint is given by the fact that certain habits are not compatible for parallel execution.

Implementation Details

The data generation technique illustrated so far requires to continuously set up and solve planning problems. Among the objects available in the simulated environment planning mainly involves *devices* and *characters*. These objects has some state properties (expressed as boolean predicates or functions) and can be manipulated by functions which modify the state of the world. It is worth to note here that (in the case an execution layer is used to drive a NPC) actions used for habit models (h-actions) has to have an equivalent action in the planning language (called *r-actions*).

Planning into real world requires to think in terms of a not fully observable environment (characters have *incomplete knowledge*), so classical planning languages such as STRIPS are not sufficient. In particular it is not possible to think in terms of the closed world assumption (CWA), so a planner supporting the *open world assumption* (OWA is needed) [7]. An example of such a planner is given by PKS [BP98] [PB02] [PB04] (Planning with Knowledge and Sensing).

PKS [8] is a planner operating at the knowledge level, where knowledge is represented using a modal logic derived from first order logic. Actions are represented as updates to the database. Applying an action's effects simply involves adding or deleting the appropriate formulas from the database. PKS deals with incomplete knowledge using introducing a set of *sensing* actions.

For any class of characters or devices, a set of propositions, functions and actions is inserted into the domain together with a set of constants denoting each object instance. Every time a new planning problem instance has to be generated over this domain. The initial state is populated with the current knowledge, while the h-actions contained in the considered habit sub-sequence are translated into the correspondent r-actions and added as a goal together with distractions.

[7] The presence of multiple not coordinate characters in the scene also remove the assumption of a static world, but, for the moment, we will ignore this issue.

[8] An implementation is available at http://homepages.inf.ed.ac.uk/rpetrick/software/pks/

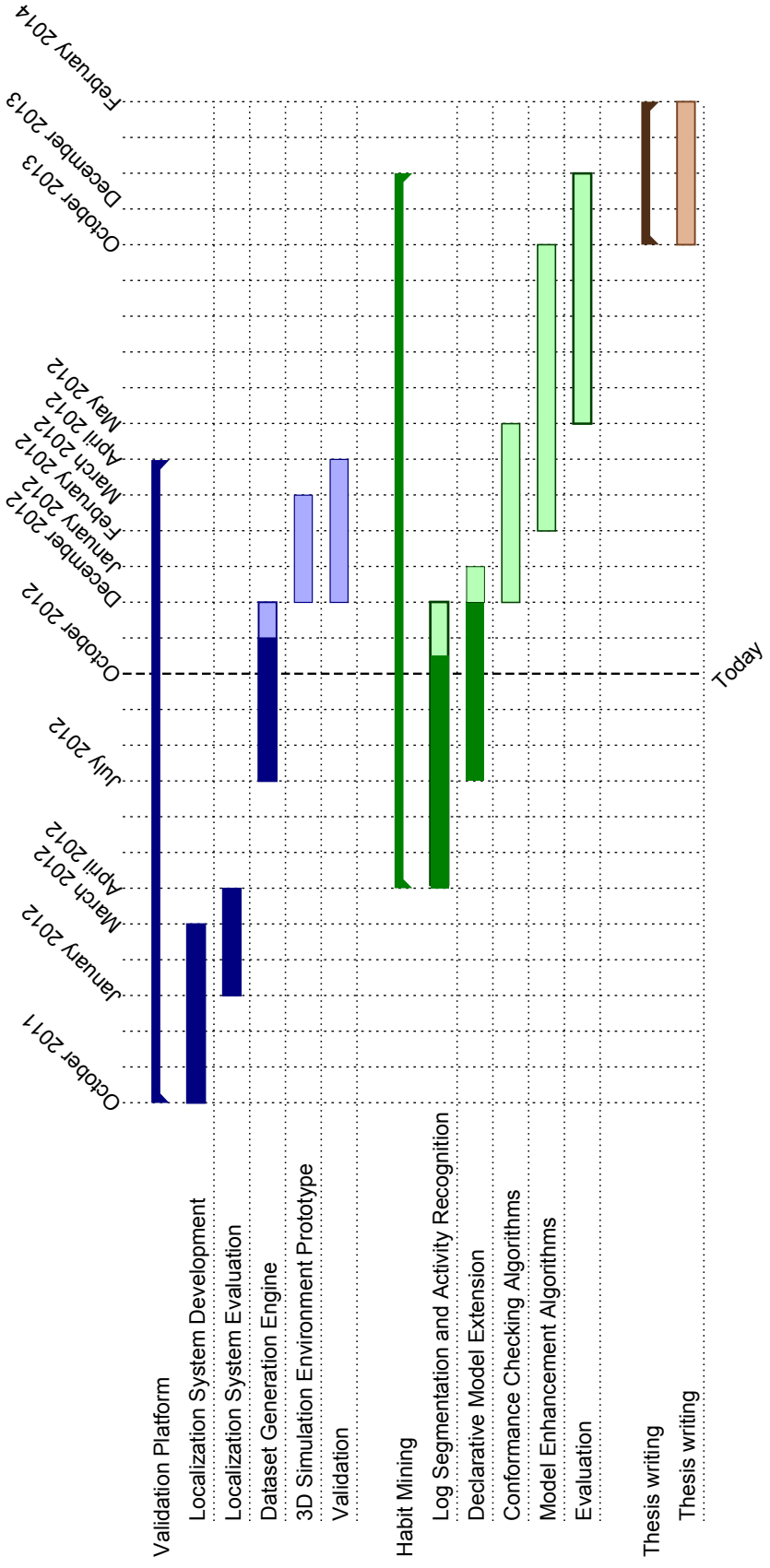

Figure 2.9. The time plan up to February 2014.

Bibliography

[AIB+10] A. Aztiria, A. Izaguirre, R. Basagoiti, J.C. Augusto, and D.J. Cook. Automatic modeling of frequent user behaviours in intelligent environments. In *Intelligent Environments (IE), 2010 Sixth International Conference on*, pages 7–12. Ieee, 2010.

[ALM+08] Juan Carlos Augusto, Jun Liu, Paul McCullagh, Hui Wang, and Jian-Bo Yang. Management of uncertainty and spatio-temporal aspects for monitoring and diagnosis in a smart home. *International Journal of Computational Intelligence Systems*, 1(4):361–378, 2008.

[AN04] J.C. Augusto and C.D. Nugent. The use of temporal reasoning and management of complex events in smart homes. In *ECAI*, volume 16, page 778, 2004.

[AN06] Juan Augusto and Chris Nugent. Smart homes can be smarter. In Juan Augusto and Chris Nugent, editors, *Designing Smart Homes*, volume 4008 of *Lecture Notes in Computer Science*, pages 1–15. Springer Berlin / Heidelberg, 2006.

[ANH07] C.B. Anagnostopoulos, Y. Ntarladimas, and S. Hadjiefthymiades. Situational computing: An innovative architecture with imprecise reasoning. *Journal of Systems and Software*, 80(12):1993–2014, 2007.

[AR11] JK Aggarwal and M.S. Ryoo. Human activity analysis: A review. *ACM Computing Surveys (CSUR)*, 43(3):16, 2011.

[BCR09] O. Brdiczka, J.L. Crowley, and P. Reignier. Learning situation models in a smart home. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 39(1):56–63, 2009.

[Bel06] C.M. Bishop and SpringerLink (Service en ligne). *Pattern recognition and machine learning*, volume 4. springer New York, 2006.

[BI04] L. Bao and S. Intille. Activity recognition from user-annotated acceleration data. *Pervasive Computing*, pages 1–17, 2004.

[BP98] F. Bacchus and R. Petrick. Modeling an agent's incomplete knowledge during planning and execution. In *PRINCIPLES OF KNOWLEDGE REPRESENTATION AND REASONING-INTERNATIONAL CONFERENCE-*, pages 432–443. Citeseer, 1998.

[BVW02] H.H. Bui, S. Venkatesh, and G. West. Policy recognition in the abstract hidden markov model. *Journal of Artificial Intelligence Research*, 17:451–499, 2002.

[CAJ09] D.J. Cook, J.C. Augusto, and V.R. Jakkula. Ambient intelligence: Technologies, applications, and opportunities. *Pervasive and Mobile Computing*, 5(4):277–298, 2009.

[CD07] Diane J. Cook and Sajal K. Das. How smart are our environments? an updated look at the state of the art. *Pervasive and Mobile Computing*, 3(2):53 – 73, 2007.

[CFJ03] H. Chen, T. Finin, and A. Joshi. An ontology for context-aware pervasive computing environments. *The Knowledge Engineering Review*, 18(03):197–207, 2003.

[CLMC12] M.G.C.A. Cimino, B. Lazzerini, F. Marcelloni, and A. Ciaramella. An adaptive rule-based approach for managing situation-awareness. *Expert Systems with Applications*, 2012.

[CNM+09] L. Chen, C. Nugent, M. Mulvenna, D. Finlay, and X. Hong. Semantic smart homes: towards knowledge rich assisted living environments. *Intelligent patient management*, pages 279–296, 2009.

[DCGA+06] P. Dockhorn Costa, G. Guizzardi, J.P.A. Almeida, L. Ferreira Pires, and M. van Sinderen. Situations in conceptual modeling of context. In *Enterprise Distributed Object Computing Conference Workshops, 2006. EDOCW '06. 10th IEEE International*, page 6, oct. 2006.

[DCM12a] C. Di Ciccio and M. Mecella. Minerful, a mining algorithm for declarative process constraints in mailofmine. *Department of Computer and System Sciences Antonio Ruberti Technical Reports*, 4(3), 2012.

[DCM12b] C. Di Ciccio and M. Mecella. Mining declarative workflows: a two-step algorithm. *Unpublished*, 2012.

[FLB09] C. Fernández, J. Lázaro, and J. Benedí. Workflow mining application to ambient intelligence behavior modeling. *Universal Access in Human-Computer Interaction. Intelligent and Ubiquitous Interaction Environments*, pages 160–167, 2009.

[GCTL10] T. Gu, S. Chen, X. Tao, and J. Lu. An unsupervised approach to activity recognition and segmentation based on object-use fingerprints. *Data & Knowledge Engineering*, 69(6):533–544, 2010.

[GGH06] BjÃűrn Gottfried, Hans Guesgen, and Sebastian HÃijbner. Spatiotemporal reasoning for smart homes. In Juan Augusto and Chris Nugent, editors, *Designing Smart Homes*, volume 4008 of *Lecture Notes in Computer Science*, pages 16–34. Springer Berlin / Heidelberg, 2006.

[GH01] D. Giannakopoulou and K. Havelund. Automata-based verification of temporal properties on running programs. In *Automated Software Engineering, 2001.(ASE 2001). Proceedings. 16th Annual International Conference on*, pages 412–416. IEEE, 2001.

[GPR06] M. Galushka, D. Patterson, and N. Rooney. Temporal data mining for smart homes. *Designing Smart Homes*, pages 85–108, 2006.

[GPZ+04] T. Gu, H.K. Pung, D.Q. Zhang, H.K. Pung, and D.Q. Zhang. A bayesian approach for dealing with uncertain contexts. In *Austrian Computer Society*. Citeseer, 2004.

[GPZ05] Tao Gu, Hung Keng Pung, and Da Qing Zhang. A service-oriented middleware for building context-aware services. *Journal of Network and Computer Applications*, 28(1):1 – 18, 2005.

[HI06] Karen Henricksen and Jadwiga Indulska. Developing context-aware pervasive computing applications: Models and approach. *Pervasive and Mobile Computing*, 2(1):37 – 64, 2006.

[HMJ+09] R. Hamid, S. Maddi, A. Johnson, A. Bobick, I. Essa, and C. Isbell. A novel sequence representation for unsupervised analysis of human activities. *Artificial Intelligence*, 173(14):1221–1244, 2009.

[HMP94] T.A. Henzinger, Z. Manna, and A. Pnueli. Temporal proof methodologies for timed transition systems. *Information and Computation*, 112(2):273–337, 1994.

[HNM+09] X. Hong, C. Nugent, M. Mulvenna, S. McClean, B. Scotney, and S. Devlin. Evidential fusion of sensor data for activity recognition in smart homes. *Pervasive and Mobile Computing*, 5(3):236–252, 2009.

[HRN+12] R. Helaoui, D. Riboni, M. Niepert, C. Bettini, and H. Stuckenschmidt. Towards activity recognition using probabilistic description logics. In *Workshops at the Twenty-Sixth AAAI Conference on Artificial Intelligence*, 2012.

[HYJK06] C. Hill, R. Yates, C. Jones, and S. L. Kogan. Beyond predictable workflows: Enhancing productivity in artful business processes. *IBM Systems Journal*, 45(4):663 –682, 2006.

[KGS05] A. Kalyan, S. Gopalan, and V. Sridhar. Hybrid context model based on multilevel situation theory and ontology for contact centers. In *Pervasive Computing and Communications Workshops, 2005. PerCom 2005 Workshops. Third IEEE International Conference on*, pages 3 – 7, march 2005.

[LHP+07] Beth Logan, Jennifer Healey, Matthai Philipose, Emmanuel Munguia Tapia, and Stephen Intille. A long-term evaluation of sensing modalities for activity recognition. In *Proceedings of the 9th international conference on Ubiquitous computing*, UbiComp '07, pages 483–500, Berlin, Heidelberg, 2007. Springer-Verlag.

[LMA11] Francesco Leotta, Massimo Mecella, and Fabio Aloise. Pericles: a performance evaluation platform for indoor localization systems. In *Proceedings of the 3rd ACM SIGSPATIAL International Workshop on Indoor Spatial Awareness*, ISA '11, pages 23–30, New York, NY, USA, 2011. ACM.

[Lok04] Seng W. Loke. Logic programming for context-aware pervasive computing: Language support, characterizing situations, and integration with the web. In *Proceedings of the 2004 IEEE/WIC/ACM International Conference on Web Intelligence*, WI '04, pages 44–50, Washington, DC, USA, 2004. IEEE Computer Society.

[Lok06] S. Loke. *Context-aware pervasive systems: architectures for a new breed of applications*. Auerbach Pub, 2006.

[Lok10] Seng W. Loke. Incremental awareness and compositionality: A design philosophy for context-aware pervasive systems. *Pervasive and Mobile Computing*, 6(2):239 – 253, 2010.

[MBvdA12] F. Maggi, R. Bose, and W. van der Aalst. Efficient discovery of understandable declarative process models from event logs. In *Advanced Information Systems Engineering*, pages 270–285. Springer, 2012.

[Mon10] Marco Montali. *Specification and Verification of Declarative Open Interaction Models: a Logic-Based Approach*, volume 56 of *Lecture Notes in Business Information Processing*. Springer, 2010.

[NM08] B. Neumann and R. Moller. On scene interpretation with description logics. *Image and Vision Computing*, 26(1):82 – 101, 2008. Cognitive Vision-Special Issue.

[NPVB05] N.T. Nguyen, D.Q. Phung, S. Venkatesh, and H. Bui. Learning and detecting activities from movement trajectories using the hierarchical hidden markov model. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 2, pages 955–960. IEEE, 2005.

[PB02] R. Petrick and F. Bacchus. A knowledge-based approach to planning with incomplete information and sensing. In *Proc. AIPSâĂŹ02*, pages 212–221, 2002.

[PB04] R.P.A. Petrick and F. Bacchus. Extending the knowledge-based approach to planning with incomplete information and sensing. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS-04)*, pages 2–11, 2004.

[Pop10] R. Poppe. A survey on vision-based human action recognition. *Image and vision computing*, 28(6):976–990, 2010.

[PPG+10] P. Palmes, H.K. Pung, T. Gu, W. Xue, and S. Chen. Object relevance weight pattern mining for activity recognition and segmentation. *Pervasive and Mobile Computing*, 6(1):43–57, 2010.

[PSvdA07] M. Pesic, H. Schonenberg, and W.M.P. van der Aalst. Declare: Full support for loosely-structured processes. In *Enterprise Distributed Object Computing Conference, 2007. EDOC 2007. 11th IEEE International*, pages 287–287. IEEE, 2007.

[RB66] J. Richard Büchi. Symposium on decision problems: On a decision method in restricted second order arithmetic. *Studies in Logic and the Foundations of Mathematics*, 44:1–11, 1966.

[RB09] D. Riboni and C. Bettini. Context-aware activity recognition through a combination of ontological and statistical reasoning. *Ubiquitous Intelligence and Computing*, pages 39–53, 2009.

[RMCM03] A. Ranganathan, R.E. McGrath, R.H. Campbell, and M.D. Mickunas. Use of ontologies in a pervasive computing environment. *The Knowledge Engineering Review*, 18(03):209–220, 2003.

[SFL02] K. Sentz, S. Ferson, and Sandia National Laboratories. *Combination of evidence in Dempster-Shafer theory*. Citeseer, 2002.

[Sha76] G. Shafer. *A mathematical theory of evidence*, volume 1. Princeton university press Princeton, 1976.

[SKDN09] Graeme Stevenson, Stephen Knox, Simon Dobson, and Paddy Nixon. Ontonym: a collection of upper ontologies for developing pervasive systems. In *Proceedings of the 1st Workshop on Context, Information and Ontologies*, CIAO '09, pages 9:1–9:8, New York, NY, USA, 2009. ACM.

[SLES11] M. Stikic, D. Larlus, S. Ebert, and B. Schiele. Weakly supervised recognition of daily life activities with wearable sensors. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 33(12):2521–2537, 2011.

[SS09] M. Stikic and B. Schiele. Activity recognition from sparsely labeled data using multi-instance learning. *Location and Context Awareness*, pages 156–173, 2009.

[UHM11] Dieter Uckelmann, Mark Harrison, and Florian Michahelles. An architectural approach towards the future internet of things. In Dieter Uckelmann, Mark Harrison, and Florian Michahelles, editors, *Architecting the Internet of Things*, pages 1–24. Springer Berlin Heidelberg, 2011.

[vdA11] Wil M. P. van der Aalst. *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Springer Publishing Company, Incorporated, 1st edition, 2011.

[VKNEK08] T. Van Kasteren, A. Noulas, G. Englebienne, and B. Kröse. Accurate activity recognition in a home setting. In *Proceedings of the 10th international conference on Ubiquitous computing*, pages 1–9. ACM, 2008.

[WM12] Michael Westergaard and Fabrizio Maria Maggi. Looking into the future: Using timed automata to provide a priori advice about timed declarative process models. In *Proc. of CoopIS*, 2012.

[YCDN07] J. Ye, L. Coyle, S. Dobson, and P. Nixon. Ontology-based models in pervasive computing systems. *The Knowledge Engineering Review*, 22(4):315–347, 2007.

[YDM12] Juan Ye, Simon Dobson, and Susan McKeever. Situation identification techniques in pervasive computing: A review. *Pervasive and Mobile Computing*, 8(1):36 – 66, 2012.

[Zad65] L.A. Zadeh. Fuzzy sets. *Information and control*, 8(3):338–353, 1965.