# Towards Tool Support for Computational Causal Behavior Models for Activity Recognition

Frank Krüger[1] , Kristina Yordanova[1], Veit Köppen[2], Thomas Kirste[1]

[1] Dept. of Computer Science, University of Rostock, Germany
{frank.krueger2, kristina.yordanova, thomas.kirste}@uni-rostock.de
[2] Center for Digital Engineering, Otto-von-Guericke-University Magdeburg, Germany
veit.koeppen@ovgu.de

**Abstract:** Complexity in Modeling and Development increases continuously. Subject of this paper is to show how developers of Computational Causal Behavior Models can be supported with a tool to improve the development process. Therefore, we analyze three case studies about probabilistic modeling of human behavior for activity recognition with respect to specific problems that might occur. We present a classification into five problem types that should be carefully considered in such projects. Finally, we present how tool support to increase the efficiency of developing Computational Causal Behavior Models might be created. In general, the paper provides useful guidelines for every model developer in the field of activity recognition, and highlights the need of tool support for such models.

## 1 Introduction

The use of models for activity recognition can help to increase the robustness of results and the performance of an activity inference engine. Several modeling approaches are introduced in the literature, that range from simple methods with hand crafted probabilistic models, to modeling languages [KYBK12, HH11] for specific domains. In contrast to data driven approaches [HFH+09], model driven approaches imply that a developer of such models is able to encode prior knowledge. In other words, either the developers of these models need background knowledge about the process or the domain experts should be able to use the modeling language, know common pitfalls, and be able to make accurate design decisions and understand their impacts on the performance.

There is a variety of modeling formalisms to describe the context of a problem and the corresponding user activities [Yor11]. In many approaches for model-based activity recognition, these formalisms are used to create a library of plans that is enriched by a developer and new cases and execution patterns are added [RGB+11]. One resulting problem is that every plan or its variations have to be added manually which could be tedious and time consuming. Thus, there are approaches with an initial set of plans that is manually created to populate a library, and later used to learn a new behavior and the library is automatically updated with newly observed plans or frequently occurring patterns [OCWS11]. This type of approaches solves the problem with completely manually model building, but from a robustness point of view still has the disadvantages of depending on previously executed behavior. In other words, the system is able to recognize only behavior that is already

observed and added to the library. However, there could be many variations of one behavior, all of them valid and probable, yet it is almost an impossible task to model all these variations of only a single behavior. For that reason, an approach based on causal models is introduced [YKK12], allowing modeling of single actions in terms of preconditions and effects, and consequently, during the model compilation, a support tool takes care of expanding causal links defined by preconditions and effects, and creating a complex probabilistic behavior leading to a goal, with multiple execution sequences, that could explain a given user state.

In this paper, we briefly introduce modeling mechanisms on causal models that we refer to as Computational Causal Behavior Models (CCBM). However, it is not a goal of this paper to present the modeling mechanism, as it is already done in [KYBK12], but rather to exemplify the need of a tool support for minimizing influence of possible problems occurring during the modeling of human actions. Developing such models is a creative process. In simple cases, a developer can just writes a model in her desired programming environment. However, modeling human behavior is complex. This also influences the development process. Furthermore, in many scenarios more than one developer collaborates. Therefore, cooperation and a knowledge transfer are necessary. In such a scenario, a tool support would increase the process of understanding and an efficiency gain is achieved.

Thus, we analyze three case studies describing model based activity recognition with respect to problems that might influence a project outcome in terms of results (accuracy, robustness, etc.) as well as execution time and modeling effort. Next, we show how tool support can improve the workflow of modeling human behavior with respect to recognition of activities.

In Section 3, we introduce CCBM and show how to use them for activity recognition. Section 4 describes three case studies, two of them using CCBM and we show what problems might occur. Based on our detailed analysis of problems, we discuss in Section 5 how a CCBM developer can be supported in a way that minimizes modeling effort.

## 2 A Motivation for Modeling Support

In software engineering, modeling languages such as UML [Obj10] or BPMN [Obj11] are widely used. These are not only used in the design phase or as documentation, but can also be included in the development process with their own techniques and process models. For example, model driven development [Sch06] is an approach to automatically generate software from formal models. In such an approach, a domain specific language is required. The graphical domain model is transformed to code. Model-driven engineering enables developers to achieve a higher productivity due to a high compatibility to heterogeneous environments, which are described and specified in the domain language, and a simplification of the design process, due to reuse of model parts and usage of patterns.

Many different models can be used to describe one system. These views are necessary to create an understanding and to improve the creative process due to focussing. Hence, to decrease effort and to avoid errors in the development an integrated tool is required to enable users, domain experts, modeler, and programmers, to create a holistic and consistent view. For efficiency issues regarding the development process, it is beneficial that

```
(:action sit-down
   :parameters (?p - person ?s - seat
       )
   :saliency 5.0
   :agent ?p
   :duration (normal (sit-dur)(sit-sd
       ))
   :precondition (and
               (at ?p ?s)
               (not (seated ?p))
               (empty ?s))
   :effect (and
             (not (empty ?s))
             (seated ?p))
   :observation (when (at ?p ?s)
          (signal (x-coord ?s)
              (y-coord ?s)))
)
```

Figure 1: Action *sit-down* in extended PDDL notation

```
(define (problem meeting)
    (:domain meeting)
    (:objects alice bob
        charlie- person
          a b c - seat)
    (:init (and
           (not (seated bob))
           (not (seated charlie))
           (empty a)
         (empty b)
           (seated alice)
           (= (sit-dur) 200)
           (= (sit-sd) 30)
    (:goal (and
           (seated bob)
           (seated charlie))
        )
)
```

Figure 2: Problem definition in extended PDDL notation

transformations between different models are possible, see for instance process model to software model transformation [GKDS05].

Another aspect is the usage of patterns. Due to abstraction, a pattern catalogue is more easily applied to graphical models. In software engineering, software design patterns [GHJV95] reduce development effort dramatically on the one hand and increase software quality on the other hand. We aim at the transfer of these properties from software engineering to CCBM. Therefore, we describe in the next section computational causal behavioral modeling. We also use different views and aim at a modeling process for CCBM.

## 3 Computational Causal Behavior Models

Computational Causal Behavior Models (CCBM) are introduced in [KYBK12]. They represent probabilistic models generated from causal human behavior models. There are several reasons for using causal models for generating probabilistic models. The most important is, that it is much easier from a designer's point of view to model causal dependencies instead of probabilistic ones [Pea09]. Additionally, they allow a generation of multiple hypotheses without explicit modeling. Details on the process of generating probabilistic models from causal models are presented in [KYBK12]. Next, we describe the process of creating these CCBM and their influence on the inference performance.

CCBM uses a PDDL[1]-like notation that allows the description of actions in terms of preconditions and effects, representing the causality of human acting. Furthermore, it has an additional benefit of modeling actions as abstract parameterizable templates. Figure 1 describes the abstract action *sit-down* modeled in terms of precondition and effects where standard elements of a CCBM can be seen.

[1] Planning Domain Definition Language

**:parameters** The *parameters* part of the declaration lists the names and types of all parameters. The compiler will later generate a grounded operator for each possible combination of parameters. In the given example these are of type *person* and *seat* which means that later the compiler will ground the abstract action *sit-down* with all objects from these types.

**:saliency** An a-priori probability of the action template which can be used to either increase or decrease the probability of selecting this action.

**:agent** It defines the execution thread for the action, and when more than one agents of the given type are defined, then their actions will be executed in parallel.

**:precondition** Specifies the preconditions to the world state that need to be met for the operator to be applied. In our example these are that the person is at the seat position, that he is not seated, and that the seat is empty.

**:effect** Describes the state of the world after the operator is applied. In Figure 1 this is that the seat is no longer empty, and that the person is seated. Predicates that are not explicitly defined in the effect clause do stay unchanged. It has to be mentioned that the action effects take place at the beginning of the actions execution, not at the end.

**:duration** Probabilistic actions durations can be used to specify the duration of actions. In the example it is a normal duration distribution with duration *sit-dur* and standard deviation given by *sit-sd*.

**:observation** In addition to the action elements presented above, we are also able to specify the action observations which link the domain specification with the observation model $p(y|x)$ needed for performing probabilistic inference. Figure 1 shows such observation clause that links the abstract user location *(at ?p ?s)* to a concrete coordinates *(x-coord ?s)* and *(y-coord ?s)*.

After defining the abstract actions, the problem specific information has to be described. Figure 2 shows such problem description, where **:objects** provides the list of objects that are to replace the parameters in the abstract action. Additionally, the initial world state is given by **:init**. In our example from Figure 2 the users Bob and Charlie are not sitting, while Alice is seated and two of the three seats are empty. Predicates that are not explicitly specified in the initial world state are assumed false. Finally, the goal has to be defined, which is achieved be the **:goal** clause. The goal in our example is that both Bob and Charlie are seated.

Although the modeling of actions in terms of preconditions and effects may seem straightforward, there are some techniques that one should have in mind before beginning to create the model.

**lock flags:** Especially when a multi-agent behavior is modeled, it is essential to use lock flags which are nothing more than predicates that lock or unlock given activities. For example, in a *walk* action, we could introduce the predicate *may-walk* that depending on whether it is set to true or false, could force the agent to execute another action in order to unlock a specific behavior (e.g., if the *walk* effect sets the flag to false, then the agent will be forced to execute another action, unlocking the flag, before being able to walk again.)

**durative actions:** Sometimes just using the *:duration* clause is not enough to model durative behavior. This is due to the fact that effects are executed at the beginning of the

action and not at the end, although logically thinking, there are effects that should be true during the action execution and other that should become valid only after the action was executed. For that reason, it is recommended to define begin-end action pairs, where in the begin-action, the duration of the action is specified, as well as the effect that will be true during the action execution. Then, in the end-action, the effects that no longer apply after the action is over, are once again set to false, and those that are valid– set to true.

# 4 Case Studies

We illustrate the need for tool support for model based behavior analysis and activity recognition by describing and analyzing three data analysis projects with several failures which resulted in either a postponed time plan or in the project failing.

## 4.1 Case Study 1: Analyzing Activities

The first project is supposed to show that changes to movement pattern of elderly people may indicate advancing dementia. To support early results, the project started with collecting data from people suffering from dementia and healthy people as control group. After the initial data collection, experienced colleagues created a data analysis workflow that encourages the initial hypothesis. Model-based methods to distinguish the two groups of patients were used. Two models of the activities were created; one was trained with the first group and the other with the second group. The likelihood ($p(y|M_i)$) was then used to determine whether the first or the second model fits better to the subject. At this point the project goal seemed to be reached and the project was handed over to a less experienced researcher.

The data collection was extended but at some point it turned out that the batteries were to weak and some features of the sensors had to be turned off to enable a runtime of more than one day. After adjusting the sensors to the needs of the study, the data collection continued until the whole dataset contained sensor readings of 64 persons, each of them lasting 48 hours. The data set for analysis was extended as new data arrived and it seemed that the results stayed good. While recording new data the whole data analysis workflow was adjusted; the preprocessing was changed, new models were developed, different performance measures were tried etc.

At some point, after adding a new data record to the data set, it turned out that the results were getting worse. After a detailed analysis of the whole project and its results it turned out that several problems, starting from the beginning, influenced the results. The most important are listed below and divided into the five problem groups: sensor hardware, sensor data, documentation, modeling, problem identification.

**Sensor Hardware.** A detailed analysis of the sensor before starting the data collection was missing. Neither the sensor nor the firmware was ever assumed to be faulty. Additionally, the sensors were never calibrated. With changing the firmware the scale for the accelerometer was incorrectly exchanged. Due to a missing check, this was never recognized and resulted in faulty data.

**Sensor Data.** After recording, the data should be analyzed in detail. In our study this step was done after the initial data collection but was skipped for later recordings. This fact also supported hiding sensor problems.

**Documentation.** To ensure reproducibility of experiments and analysis, the whole workflow and any changes should be documented. In the first place, there was no detailed questionnaire for patients, to record also information about them, for later analysis. Also the assignment of specific sensors to a given patient was missing. The workflow, consisting of a set of shell scripts without comments, was copied from folder to folder.

**Modeling.** Changes to the initial model that were done by the less experienced researcher did not follow a hypothesis but just tried to increase the number of distinguished people. The models that were used to reflect the activity course of elderly people with and without dementia were created by using a domain specific language for (hierarchical) Hidden Markov Models (HHMMs). The usage of such domain specific languages requires knowledge of the process behind the models. Unfortunately the less experienced researcher was neither familiar with the probabilistic modeling nor with the modeling language. As described above different modeling paradigms were used, e.g., simple and hierarchical HMMs. The repeating change between these paradigms without concrete (and not documented) reasons resulted in a unsorted set of models without documentation.

**Problem Identification.** One of the most time intensive points in this project was identifying problems. After problems occurred searching for problems was always executed backwards in the workflow, which means problems were always assumed to be in the last steps, namely classification or the modeling. Thus the model was *adjusted* and *fine-tuned* without any success. Distinguishing the different groups of people was based on comparing the likelihood of two different models without checking whether the model fits the behavior of the people. Thus, the results of the comparison were misinterpreted.

### 4.2 Case Study 2: Modeling Multiple Agents

The second study used CCBM to infer the behavior and social interaction of multiple users in smart environments. One to three agents were modeled, each of them trying to reach the same goal.

We asked a group of students to model this scenario in regard to distinguishing different situations. Afterwards, we asked them to present their results and discussed problems. Below the problems are listed and categorized to the problem classes above. Due to the kind of the task most problems occurred while modeling the behavior.

**Sensor Hardware.** The smart environment where the study took place [BRK10] is equipped with capacity measuring sensor mats. These mats can work in different modes, first the self adjustment mode, where the system measures the difference in capacity and adjusts the base value slowly over time. The second mode is the *normal mode*, where the system signals the measured capacity difference to a reference value. The sensor floor is typically set to self adjustment mode[2]. To increase accuracy of the observation models,

[2]This is valid due to the ongoing movement of persons.

students tried to compute sensor noise, but did not change the position during a long term measurement. Due to self calibration the measured capacity decreased with time.

**Sensor Data.** The sensor data itself consists of one struct containing six numbers (0 or 1) per second. The students computed Type 1 and Type 2 error of the sensors cells but did not add this information to the observation model, to increase the recognition rate.

**Documentation.** One part of the project was to create different models and check whether one of them fits better to the sensor data than the others. The students created eight models. During final presentation, they showed, that their models are able to distinguish different situations, but it was not documented which model recognizes which goals.

**Modeling.** The students tried to build a model that should recognize activities in real time. Therefore, they used the *:duration* parameter. But instead of delaying the effect of the action by setting duration to *start action*, they added a duration parameter to *end action*, which did not result in strange agent interaction. Another question that was raised very late, what happens if there are actions without agent specification in a multi-agent model? It was unclear that these actions were executed in the standard agent slot called *null*.

**Problem Identification.** To use the goal distance as heuristic for tracking the behavior, an additional tool, the state space analyzer, is created. This tool analyzes the whole state space and computes the goal distance from each possible state. Additionally it gives information about the reachability of the goal and about possible deadlocks. This tool was used by the students for calculating the heuristic but they ignored the output, telling that 50% of all possible plans lead into deadlocks. When the students finished creating a model, they were surprised by the low recognition and the high resampling rate, but did not realize that the number of deadlocks has any impact on this.

### 4.3 Case Study 3: Modeling General Meeting Behavior

The third study also used CCBMs to infer the user behavior in a general meeting scenario. The purpose of the project was to create a reusable model that is able to recognize the activities of meetings with different settings and durations. The scenario that was modeled was a 3-Person meeting. The model was supposed to be robust enough to cope with different actions durations or behavior variations.

To test the model, 20 small meetings lasting about three minutes each, were recorded, where the users presented and then discussed their topics; and an additional long meeting was recorded which lasted about 53 minutes and was not staged. The participants in this meeting presented real topics as part of a project seminar, and did not follow a beforehand established behavior pattern.

**Sensor Hardware.** The meetings took place in our Smart Appliance Lab (or SmartLab for short) [BRK10] which is equipped with various sensors and actuators, among which the Ubisense technology which estimates users' location by transmitting ultra-wide band radio signals, captured by a fixed network of receivers [Ubi]. To test our model, the Ubisense sensors were used which transmitted the user location in x, y and z coordinates whenever

the location changes. Due to ongoing movement of persons the single measurements were assumed to be equidistant in time. Unfortunately this was not the case.

**Sensor Data.** There were problems with the data format, as the 20 small meeting contained a *time* column, while the long meeting did not, which resulted in erroneous results during the state estimation before the authors noticed that the format was not the same. Furthermore, the values of the 20 small meetings were recorded in centimeters, while those of the long meeting – in meters. Additionally, the positions of the stages and the seats for the small meetings differed from those for the long, which all together with the varying measurement units and actions durations resulted in the need of creating two separate observation models. Finally, the too coarse-grained level of annotation resulted in the need to make assumptions based on the sensor data, regarding the multi-agent behavior (for the small meetings only the team behavior was annotated regardless of the fact that the actions of the separate agents were tracked).

**Documentation.** Although the project was seemingly well planned, and the roles carefully distributed, no documentation was created during the execution phase itself, but rather after the project was finalized, which resulted in forgetting to add information that could be important for later projects. Regardless of the fact that version control was used, not every version was submitted, resulting in the inability to identify the exact modeling mechanisms responsible for unexpected artifacts in the estimated behavior. The assumptions on which a certain modeling mechanism was used as well as the design decisions were not documented, leading to inability to explain why they were used at all.

**Modeling.** As the model was meant to describe both the twenty short meeting, and the one long meeting, where they differ in structure and duration, it had to allow for variations in behavior and order of actions executions. Although the model produced relatively good results, it showed that it is not so straightforward to model team behavior emerging from the single users' behavior and that one should be extremely careful with using lock mechanisms. In this case they resulted in half of the model states not reaching the goal state and in some situations forced the inference engine to make seemingly illogical state estimation. Additionally, during most of the model adjustment, only exponential duration was used, which resulted in poor results for the long meeting, and almost at the end of the project it was discovered that the normal durations perform comparatively better. Another error regarding the durations, was adjusting them experimentally to fit the sensor data, instead of computing them. Furthermore, when the model did not perform well, it was adjusted on the principle of trials and errors without trying to find the exact reasons for its behavior or probably build a new model from scratch. Moreover, as the evaluation scripts were not fixed at an early stage, only the results of the most likely particle were checked, resulting in not discovering modeling problems that were later evident after performing the final evaluation.

**Problem Identification.** As the observation models were reused and adjusted from previous experiments, it did not occur to the authors to check them for errors, which later resulted in unexpected model behavior, the cause of which turned out to be faulty observation models. During most of the project, the model was validated only against the

annotations, and only by applying one run which resulted in the too late discovery of a whole set of problems varying from ones caused by the modeling mechanism to such based on the nature of the approximate inference. Furthermore, when the model did not fit the annotation, the annotation was searched for errors, instead of assuming that there is a problem in the model, or that the errors result from the nature of the inference engine, or sensor data.

### 4.4 Problem Classification

In this section, we briefly summarize the identified problems. Problems that arise in model based data analysis projects can be categorized into five classes: sensor hardware, sensor data, documentation, modeling, and problem identification. We state mechanisms to prevent typical problems in the following.

**Sensor Hardware:** Without *sensor calibration*, comparison of different sensor types is impossible. Sensor calibration may also help discovering the behavior of the sensor at the end of the measurement range. The *boundary behavior* of sensors is important to interpret measurement values that are out of range. To correctly interpret sensor measurements, an *error model* of a sensor should contain as much knowledge of the sensor as possible.

**Sensor Data:** A *(visual) analysis* of recorded sensor data can support discovering problems with data, such as exchange of sensor axes. Problems with sensors themselves might also be discovered. The usage of sensor data in a *common data format* can help to prevent errors for I/O implementation.

**Documentation:** A missing *documentation of design decisions* encourages repeating experimental modeling. The use of a *version control* system supports documentation and documentation of single model parts such as prepositions or operators in CCBM. Also the evolvement of models including design decisions should be documented.

**Modeling:** Typically designing a model for activity recognition requires a hypothesis of the behavior to recognize. This hypothesis is either created by analyzing the data or by analyzing the behavior that should be tracked. Modeling without such an hypothesis does not lead to a good reflection of reality. Before modeling, a *hypothesis* on process structure should be established. The lack of such a hypothesis leads to *uncontrolled* modeling experiments. To create models in domain specific *modeling languages*, all aspects of a language should be known. The impact of modeling decisions should be clear. A change in existing models should not be done without a *detailed understanding*.

**Problem Identification:** If outcomes are not expected, the complete workflow should be checked. Therefore, it is necessary to ensure deep *knowledge of tools* used in the workflow. Ignoring or misunderstanding features can lead to a disorganized problem search. In case of our experiment reviewing, the *system output* was important.

## 5 Developing Computational Causal Behavior Models

In this section, we want to address some possibilities, how to support the human behavior domain expert with a tool. Consequently, a tool should include our five classes from

our case studies. This means, in the context of CCBM, sensor hardware and sensor data have to be supported in modeling as well as documentation and the modeling process itself. Furthermore, the problem identification should be supported with problem analyses and outcome evaluations. Note, not all challenges can be overcome by a single tool. However, the process of model creation, as comparable to software development (see Section 2), can be supported with tools, techniques, and a modeling process guide, including best practices, patterns, or conventions. There exist many different approaches, such as context-aware modeling or application specific development that is often hard-wired in the systems, e.g., [HDE04] or [SPSS11]. In the following, we focus on a tool support in a more general development. Challenges that occur due to system architecture of human behavior assistance systems are addressed in [KKS11]. These challenges can be seen as requirements for the architectural design, namely, openness, robustness, adaptivity, understandability, and standardization. In the context of CCBM, it is necessary that all requirements are carefully considered. Due to interweaving of requirements, a modeler should be assisted in the development process within a CCBM tool. In such a way, an integrated development system does not only support, but also increases quality and enhances efficiency of the complete development process.

In the domain of CCBM, a tool support should address technical, functional, and non-functional properties of the complex system. Due to given restrictions, new approaches in the development, especially in the phases of design and implementation, have to be considered. This is also true for different granularity levels in the models, e.g., precondition and effect modeling. Therefore, navigation in the hierarchies, consistency, and transaction concepts are required.

Within the CCBM tool a differentiation of model views is necessary. This includes not only **sensor data** and behavior model, but also documentation aspects have to be considered. The decoupling or abstraction to a model view on a complex system enables the user to easily understand and apply changes. However, the inherent relationships have to be consistently transferred to all models for the same system. This means that the decoupling of **sensor hardware**, sensor data, and behavior model has to be internally merged in a holistic and consistent model.

We are interested in the development of a graphical, but also formal modeling language. A graphical model decreases the effort to understand the system, and a formal model enables an efficient transfer in the implementation of a model. This includes in the CCBM domain formal semantics. Another important efficiency aspect is the usage of different abstraction levels, e.g., actions, activities, or processes. These different abstraction levels have to be supported for the **modeling**. This can be applied with the usage of patterns or templates that enable a support in **problem identification**.

Another challenging task is adaptation of models. In such a case, variability of models and model elements has to be supported, too. Therefore, a CCBM tool should include an integrated version control. A possible approach can be adapted from feature oriented development. This means, that a system is described by its functionalities and different variants can easily be generated by a selection of features. These variation points enable a decreased development effort and provide a high degree of reuse. For the description of features and their relationships feature models are used. This enables the selection of

desired features and a generation of the system can be automatically performed. Applied to CCBM, this means with the selection of provided elements the user is enabled to easily construct her desired system. Reuse of models, elements, or sub models is possible.

Another property of a CCBM tool is the transformation of code fragments automatically into the corresponding graphical model. This enables a automatic **documentation**. Furthermore, other techniques from software engineering such as refactoring, e.g., renaming elements or descriptions, should also be implemented in the CCBM tool.

From our described case studies in Section 4, we see different views and consequently different model types. Compared to software engineering, see Section 2, a holistic and consistent complete model is necessary. Therefore, a suitable differentiation in the model views is necessary. We divide this in the CCBM domain into actions (with hierarchies) and agents or observation/sensor model. This is for instance comparable to static and dynamic models in UML.

## 6 Summary and Conclusion

In this paper, we saw that, even if the use of causal models for behavior analysis, such as CCBM can increase the efficiency and decrease the effort of the developing process, problems known from other modeling areas occur. An analysis of three case studies, two of them using CCBM, showed what kind of problems typically occurs. After a detailed analysis of the problems, we classified them into five classes which should be carefully considered by the model developer before building a new model. Furthermore, we discussed the benefits of having a tool support for CCBM and the features it should possess in order to provide the needed developer assistance.

This is a first approach to support a computational causal behavior modeler in the development of new models or in the identification of existing problems in the models. We try to adapt methods and technologies from software engineering, where we see many connections. However, due to only first attempts many questions have to be investigated in more detail.

## References


[BRK10] Sebastian Bader, Gernot Ruscher, and Thomas Kirste. A Middleware for Rapid Prototyping Smart Environments. In *Proceedings of the 12th ACM international conference adjunct papers on Ubiquitous computing*, Copenhagen, Denmark, 2010.

[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1995.

[GKDS05] Christiane Gernert, Veit Köppen, Oliver Darkow, and Thorsten Schwarz. Von ARIS zur UML: Transformationen in der Prozessmodellierung. *ObjektSpektrum*, 6:53–59, November/December 2005.

[HDE04] Andreas Hub, Joachim Diepstraten, and Thomas Ertl. Design and development of an indoor navigation and object identification system for the blind. In *Proceedings of the 6th international ACM SIGACCESS conference on Computers and accessibility*, Assets '04, pages 147–152, New York, NY, USA, 2004. ACM.

[HFH+09] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The WEKA data mining software: an update. *SIGKDD Explor. Newsl.*, 11(1):10–18, November 2009.

[HH11] LM Hiatt and AM Harrison. Accommodating Human Variability in Human-Robot Teams through Theory of Mind. In *IJCAI*, pages 2066–2071, 2011.

[KKS11] Veit Köppen, Thomas Kirste, and Gunter Saake. Challenges in an Assistance World. In *LWA 2011: Lernen, Wissen & Adaptivität, Workshop Proceedings*, 2011.

[KYBK12] Frank Krüger, Kristina Yordanova, Christoph Burghardt, and Thomas Kirste. Towards Creating Assistive Software by Employing Human Behavior Models. *Journal of Ambient Intelligence and Smart Environments*, 4(3), Match 2012. accepted.

[Obj10] Object Management Group. OMG Unified Modeling Language (OMG UML), Infrastructure, V2.3. Technical report, May 2010.

[Obj11] Object Management Group. Business Process Model and Notation (BPMN) - Version 2.0. Technical report, 2011.

[OCWS11] G. Okeyo, L. Chen, H. Wang, and R. Sterritt. Ontology-Based Learning Framework for Activity Assistance in an Adaptive Smart Home. In L. Chen, C. D. Nugent, J. Biswas, J. Hoey, and I. Khalil, editors, *Activity Recognition in Pervasive Intelligent Environments*, volume 4 of *Atlantis Ambient and Pervasive Intelligence*. Atlantis Press, 2011. 10.2991/978-94-91216-05-3_11.

[Pea09] J. Pearl. *Causality: Models, Reasoning, and Inference*. Cambridge University Press, 2009.

[RGB+11] P. C. Roy, S. Giroux, B. Bouchard, A. Bouzouane, C. Phua, A. Tolstikov, and J. Biswas. A Possibilistic Approach for Activity Recognition in Smart Homes for Cognitive Assistance to Alzheimer's Patients. In L. Chen, C. D. Nugent, J. Biswas, J. Hoey, and I. Khalil, editors, *Activity Recognition in Pervasive Intelligent Environments*, volume 4 of *Atlantis Ambient and Pervasive Intelligence*, pages 33–58. Atlantis Press, 2011. 10.2991/978-94-91216-05-3_2.

[Sch06] Douglas C. Schmidt. Guest Editor's Introduction: Model-Driven Engineering. *Computer*, 39(2):25 – 31, Feb. 2006.

[SPSS11] Jaeyong Sung, Colin Ponce, Bart Selman, and Ashutosh Saxena. Human Activity Detection from RGBD Images. In *AAAI workshop on Pattern, Activity and Intent Recognition (PAIR)*, 2011.

[Ubi] Ubisense. Real-time location systems (RTLS) and geospatial consulting. http://www.ubisense.net, Retrieved: 24th January 2012.

[YKK12] Kristina Yordanova, Frank Krüger, and Thomas Kirste. Context Aware Approach for Activity Recognition Based on Precondition-Effect Rules. In *In Proceedings of the workshop COMOREA at PerCom'12*, Lugano, Switzerland, 2012.

[Yor11] Kristina Yordanova. Toward A Unified Human Behaviour Modelling Approach. Technical Report CS-02-11, Institut für Informatik, Universität Rostock, Rostock, Germany, May 2011. ISSN 0944-5900.