Modeling Temporal Dependencies Between Observed Activities

Svenja Kahn Tobias Klug Felix Flentge

{kahn,klug,felix}@tk.informatik.tu-darmstadt.de
Department of Computer Science, Telecooperation Group
Technische Universität Darmstadt, Germany

ABSTRACT

The modeling of parallel activities requires a notation which can represent the temporal dependencies as well as variations of the execution order of the activities. This paper introduces ART (Activity Relation Trees), a notation to describe temporal dependencies between activities. ART is based on ConcurTaskTrees (CTT) that are extended with the means to describe temporal relationships. Furthermore, we present an algorithm that allows to automatically generate ART models from observed examples. Because former approaches for automatic model acquisition were restricted to strictly sequential data and cannot be applied in the case of parallel activities, we developed a method to reduce the problem of automatic modeling of parallel activities to the simpler task of modeling sequential data. By grouping activities and distinguishing different phases we are able to form general descriptions of a scenario that include variations in the execution order. The paper defines all necessary concepts and describes the algorithm in detail. The evaluation of the algorithm shows that precise models can be generated by using only few examples.

Categories and Subject Descriptors: I.2.4 Knowledge Representation Formalisms and Methods: Temporal Model
General Terms: Algorithms, Human Factors

1. INTRODUCTION

Observing parallel activities and building coherent models from these observations is necessary in a number of settings. In software engineering or interaction design one of the first steps is to survey user requirements which can be facilitated if corresponding models are available. When designing wearable applications, such models can be used to predict the user's next step and to adapt the user interface accordingly. They can also be used to facilitate searching and tagging of multimodal material that has been gathered in parallel to the activity modeled. For example if a model of a person's day could be built, we could later on ask which material was gathered during breakfast, lunch or dinner.

However, building such a model is not an easy task. First, a notation that is flexible enough to represent all possible temporal dependencies as well as variations in the execution order is necessary. Therefore, we developed a notation called ART (Activity Relation Trees) based on Concurrent Task Trees (CTT) [10]. ART models are well suited for humans as they illustrate the possible variations of a scenario in a compact way. They could also be used for task recognition by creating a probabilistic model, for example a Dynamic Bayesian Network (DBN), from the ART model. This could be done in a similar way as described by Giersich et al. who create DBNs from CTT models [5]. Because the amount of work required to construct a model manually is often prohibitive, automatic methods are required. We have developed an algorithm that is able to extract an ART model of a scenario from a series of examples. Each example is a collection of events associated with a number of activities. Our algorithm analyzes these examples to find commonalities and organizes these into an ART model.

Our motivation for developing this algorithm was to extract temporal models from user observations in mobile settings to facilitate the design of wearable computing solutions that can be used in parallel to the scenario observed. Figure 1 shows an example of the data gathered from observations in a medical examination scenario. However activity recognition algorithms can be used to create these kinds of examples from all kinds of data gathered automatically, removing the need for manual observations.

Continuous activities have a temporal extension. They can either be executed sequentially or in parallel. Activities which can be executed independently of each other are called "concurrent activities". An example of two concurrent activities are "holding a pen" and "talking to another person". If the execution of one activity temporally depends of the execution of the other activity, these activities are called activities with temporal dependencies. There is a temporal dependency between holding a pen and writing a note as a pen is always being held when something is written.

We seek to model continuous activities which are executed in parallel and the temporal dependencies between the execution times of these activities. Time intervals represent the time during which an activity is being executed. Several time intervals represent the repeated execution of an activity. Figure 1 visualizes the time intervals of 20 activities which constitute a medical examination. This data can be gathered with tools like the TaskObserver [8].

ICMI Š07 Workshop on Tagging, Mining and Retrieval of Human-Related Activity Information November 15, 2007, Nagoya, Japan

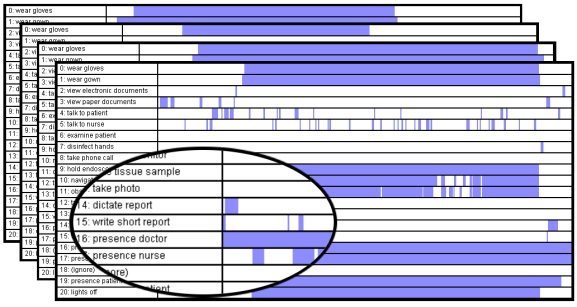


Figure 1: Four observed examples of a medical examination. Each row represents one type of activity and each bar indicates that this activity was observed during this interval.

Each obervation becomes one example of the scenario. Our objective is to create a model which represents not only one but several examples. The model should incorporate the basic structure of the observed scenario as well as possible variations. Therefore, the examples must be compared to find out which parts are identical and which temporal dependencies can be found.

2. RELATED WORK

2.1 Modelling Parallel Activities

Our goal is to model parallel activities. This is why we need a representation which allows to represent the temporal dependencies not only between sequential but also between parallel activities. This representation should be scalable and readable by humans to facilitate the creation and discussion of models with domain experts. The model should also represent possible variations, e.g. whether the execution order of two successive activities can be exchanged or the execution of an activity is optional. We evaluated whether Petri Nets (the most commonly used notation for parallel events) or task models (which are often used for the modeling of activities and goals) fulfill these requirements.

Extensions of Petri Nets ("time Petri nets" or "timed Petri nets") can be used to model temporal dependencies between parallel activities. Yao showed that the temporal relations which were identified by Allen [1] can be expressed with Petri Nets [13]. Petri Nets are well suited for simulations but their visual representation is complex, so they are difficult to read for humans. Another disadvantage is that they store information in an implicit way (via markings), so information about the current state is difficult to obtain.

Task models are used to represent the execution of several activities. We compared several approaches for task modeling (GOMS [7], Diane+ [12], UAN [6] and ConcurTaskTrees (CTT) [10]). Except for the purely visual representation of CPM-GOMS, these modeling notations do not incorporate the representation of temporal dependencies between parallel activities. CTT, UAN and Diane+ offer the possibility to represent the fact that two activities are executed in parallel. However, this is restricted to mark activities as "concurrent", i.e. no further information about their temporal relation is given.

Except for the missing operators for the description of the exact temporal relations between parallel activities, CTT is well suited for our needs. CTT models have a hierarchical structure which makes CTT models scalable because there are different abstraction levels. A CTT model corresponds to a tree whose leaves represent the executed tasks. The nodes of the tree are connected by operators. Unitary operators allow to mark nodes as optional or iterative, binary operators like "order independence" or "choice" describe the relations between nodes which have the same parent node and allow the representation of possible variations. Because of its hierarchical tree structure, CTT offers an easily understandable representation for complex models and the possibility to represent variations. For this reason, we extend the CTT notation with operators describing temporal dependencies between parallel activities.

2.2 Automatic Model Acquisition

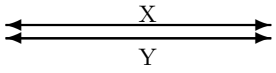
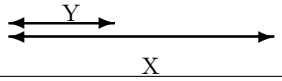
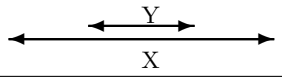
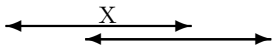
There are a number of approaches to create models automatically from observed examples, e.g. in the domains of task modeling, linguistic or bioinformatic. These approaches are usually based on sequential data. Garland and Lesh desribe how hierarchical task models can be automatically created from annotated examples [4]. Each example corresponds to one execution of the scenario to be modeled. Every (sequential) example is annotated, i.e. a human expert annotates which subtasks form a task. The annotated examples are combined to a model by an inference mechanism which infers possible variations like optionality or order independence. The hierarchy of the tasks models has to be given manually.

Eyharabide and Amandi present an approach for the automatic creation of CTT models [3] which automatically creates the model hierarchy and which does not assume annotations. Every user interaction with an application corresponds to a sequence of successing events. Repetitive patterns (for example if the subsequence *abc* occurs in several sequences) are considered to be tasks. Whenever a new sequence is given to the algorithm, the tasks are adapted to these new sequences, i.e. every new sequence leads to an update of the previously seen tasks. This adds possible variations to the model.

3. TEMPORAL MODEL OF HUMAN ACTIVITIES

The statement "two activities are concurrent" is not precise enough for the modeling of the temporal dependencies between parallel activities. The relation between the execution of two activities (respectively between the time intervals which represent the execution times) can be represented by the relations identified by J.F. Allen. Allen presents 13 relations (equals, before, after, meets, met by, started by, starts, includes, during, finished by, finishes, overlaps and overlapped by) between time intervals [1]. Every possible relation between two intervals is described by one of these relations. For modeling temporal dependencies, distinguishing all of Allen's relations can be too specific. For example, Allen distinguishes "A before B" (interval B starts after A stopped) from "A meets B" (interval B starts at the instant in which interval A stops). Consider the following situation: The execution of activity A is observed. Immediately afterwards, activity B is executed. If Allen's relations are used for the modeling of these activities, it is not clear whether "A before B" or "A meets B" should be used. This problem

Table 1: Combining Allens relations

Relations identified by Allen	Visual representation	Combined version
X equals Y (Allen)		X equals Y
X before Y (Allen) Y after X (Allen)		X before Y (=Y after X)
X meets Y (Allen) X met by Y (Allen)		
X started by Y (Allen) Y starts X (Allen)		
X includes Y (Allen) Y during X (Allen)		X contains Y (=Y during X)
X finished by Y (Allen) Y finishes X (Allen)		
X overlaps Y (Allen) Y overlapped by X(Allen)		X overlaps Y (=Y overlapped by X)

becomes even more difficult when we take into account inaccuracies from the observation of activities. Inaccuracies can cause different descriptions (e.g. "A before B" versus "A meets B") for the same state. This makes the comparison of created models difficult.

For the modeling of observed activities, "A before B" and "A meets B" should be combined. The same applies for the relations "during", "starts" and "finishes". The relations identified by Allen are shown in the left column of table 1. In ART, the relations shown in the right column are used. Each relation combines one or several of Allens relations. A relation which is inverse to another relation can also be represented by the other relation: "before" ist invers to "after", i.e. "B after A" can be represented by "A before B". Likewise, "contains" is inverse to "during" and "overlaps" is inverse to "overlapped by". This is why all temporal relations between two intervals can be represented by one of the four relations equals, before, contains and overlaps.

Based on ConcurTaskTrees (CTT), we use a notation which corresponds to a tree structure. In our representation, every leaf of the tree represents the execution of an activity. The other nodes are called "abstract nodes". These nodes can be used to build a hierarchy: Every node inherits the relations which are valid for its parent node (a relation between two abstract nodes also applies to the children of these nodes). Abstract nodes can be used to resolve ambiguities concerning the execution order: To describe (a before b) before c, a and b are grouped under a common abstract node. To describe a before (b before c), b and c are grouped under an abstract node.

We use the CTT tree structure which has the characteristic that relations (e.g. "Choice" or "Before") connect neighboured nodes and are written between the nodes to which they refer. The relations we use are given in Table 2. The first three relations were adopted from CTT, the relations "Before", "Equals", "Contains" and "Overlaps" allow to model the temporal dependencies between parallel activities. In CTT, relations are represented by symbols whereas in this paper, relations are represented by words to facilitate the understandability. As in CTT, tree nodes can be iterative or optional.

Table 2: ART Relations

Concurrent (A concurrent B)	The execution of A is independent of the execution of B
Choice (A choice B)	Either A or B can be executed.
Order Independent (A order independent B)	A and B are executed sequentially but the order of their execution is arbitrary.
Before	Before allows to describe activities which are executed sequentially.
Equals	Both activities are executed at the same time.
Contains	If A Contains B, B is executed during the time in which A is executed.
Overlaps	If "A overlaps B", the execution of activity A begins before B and ends while B is being executed.

- Iteration A*: Nodes which are marked as iterative can be executed several times
- Iteration A*(m-n): These nodes can be executed between m and n times
- Optionality [A]: Optional nodes can be executed but don't have to

An example for an ART model which combines the tree structure used in CTT with the temporal relations we introduce in this paper is shown in Figure 2. The abstract nodes *2* and *3* are concurrent. This means that the execution of

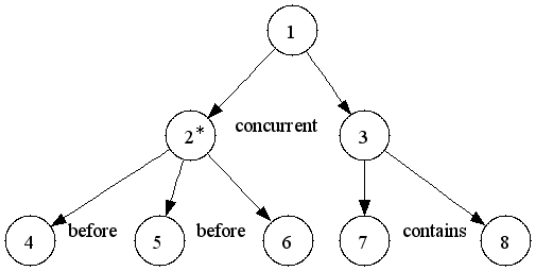


Figure 2: ART model

the activities modeled by the nodes *4*,*5* and *6* is independent of the execution of the activities modeled by the nodes *7* and *8*. The activities *4*, *5* and *6* must be executed one after another (this can be repeated because node 2 is iterative) and activity *8* is executed during the execution of activity *7*.

ART combines the advantages of CTT (hierarchical description of a notation which can be adapted manually) with the possibility to represent the temporal dependencies between parallel activities. The ART representation of temporal dependencies is much easier to understand visually than a representation with Petri Nets (see Fig. 3).

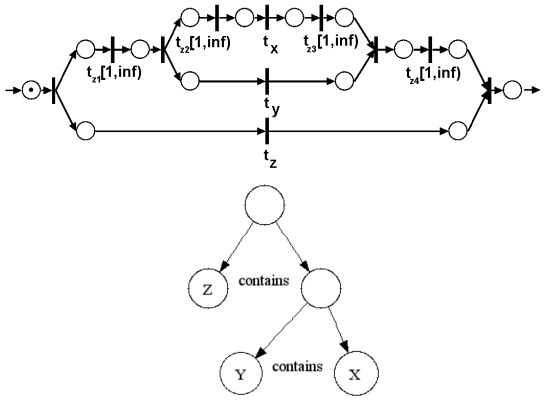


Figure 3: (X during Y) during Z represented with a Petri Net and with ART

4. AUTOMATIC MODEL ACQUISITION

The automatic model acquisition for parallel activities is more complex than automatic model acquisition for sequential data. Several temporal dependencies have to be taken into account instead of just "before" and "after". Complex relationships have to be modeled and it is more difficult to compare different configurations because the temporal relationships between parallel activities are more entangled than relationships between strictly sequential activities.

Previous approaches for automatic model acquisition which assume sequential data cannot be applied directly to the modeling of parallel activities. This is why we developed the AMPA (Automatic Modeling of Parallel Activities) algorithm. By adapting an idea Mörchen used in an approach for Time Series Knowledge Mining (TSKM) [9], we are able to map the automatic model generation problem for parallel activities to the easier problem of model generation for sequential events. Mörchen divides parallel data into sequences of parallel events in order to detect patterns in time series. We adapt this idea by dividing examples of a scenario into different groups. These groups are compared to the groups of the other examples and similar groups are assigned to identical "phases". Each example is thus transformed to a sequence of successive phases (see Fig. 4). The order in which the phases occur can be compared and modeling approaches for sequential data (e.g.[3]) can be used to create a model which incorporates variations between the occurences and execution orders of phases. Furthermore, the groups which are assigned to the same phase can be compared to detect possible variations within a phase.

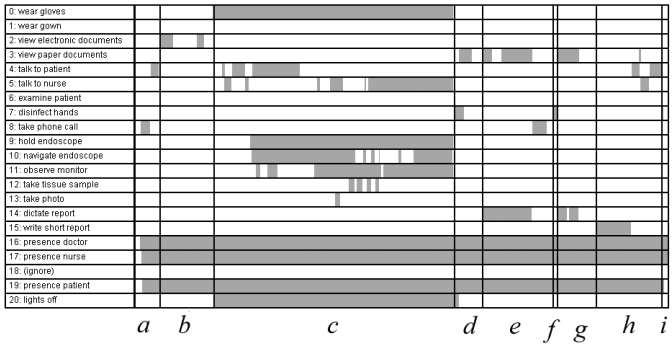


Figure 4: Phases of an example

Figure 5 shows the five steps of the AMPA algorithm for automatic model acquisition. First a preprocessing step corrects errors in the input data introduced by human observations. This is important because the next steps assume correct observations (all observed variations are incorporated into the model). Then the temporal relations between the time intervals ("before", "equals", "contains" or "overlaps" as introduced in Table 2) and activities (e.g. whether two actitities can be executed independently of each other) are calculated for each given example. These values are used to create the groups and assign them to phases. The next step (tree formation) builds an ART model which reflects the temporal dependencies and possible variations on the level of whole phases. In the last step, the temporal dependencies and possible variations within a phase are incorporated into the tree.

4.1 Preprocessing

If human observations are the source of the examples, some kind of preprocessing is likely necessary. For example, if there are subsequent activities the end of the first activity is often not detected in time and both activities appear as overlapping. In this case, there is only a small overlap of both activities. Such small overlaps can be detected and removed automatically. The observation errors of our example scenario could be removed that way. Depending on the kind of data available other kinds of automatic preprocessing are also possible, e.g. we can look for outliers or use additional data for plausibility checks. For manual data correction visualisations like the one in Figure 4 can be used to discuss the observations with domain experts and correct them [8].

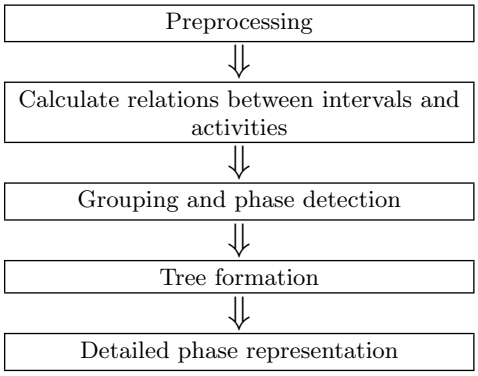


Figure 5: Automatic Modeling of Parallel Activities (AMPA)

4.2 Relations between intervals and activities

For each example the pairwise relationship between all activity intervals are calculated. These relationships are described by the combined versions of Allens relations as given in Table 1. If k different activity intervals are observed, (k-1) such relations are calculated for every interval.

The dependencies between activity intervals are used to investigate which pairs of activities are concurrent (i.e. can be executed independently of each other) and to calculate the coherence between pairs of activities. The coherence (as well as the concurrency) between two activities is a property of the scenario which is identical for all examples. If we find evidence in any of the examples that two activities are concurrent, we know that these activities can be executed independently in all of the examples. Activities are considered to be concurrent if they appear at least once at the same time and at least once when the other activity is not executed (if activity A is always executed while B is executed, these activities are not concurrent). This definition of concurrency is adequate for our scenarios (e.g. the medical examination) and can be altered for other scenarios.

For the next AMPA step "grouping and phase detection", we need to know the coherence between two activities. It is assumed that activities have a strong coherence if they tend to appear in parallel. To determine the coherence the overall duration $d(A)$ of an activity A, the sum of the length of all activity intervals is calculated. For each pair of intervals A and B, the common duration $d(A,B)$ is calculated by summing the time in which both activities are executed in parallel. We can define the overlap $o(A,B)$ between two activities A and B by

$$o(A,B) = \begin{cases} 0 & \text{if d(A)=0} \\ \frac{d(A,B)}{d(A)} & \text{else} \end{cases}$$

Note that the overlap is not symmetric. If the duration of A is not equal to the duration of B, $o(A,B)$ is not equal to $o(B,A)$.

The coherence coh of two activites A_1 and A_2 is defined as

$$coh(A_1, A_2, neutralValue)$$
$$= \begin{cases} neutralValue & \text{if } A_1 \text{ and } A_2 \text{ are concurrent} \\ max(o(A_1,A_2), o(A_2,A_1)) & \text{else} \end{cases}$$

If two activities can be executed independently of each other, the coherence corresponds to a neutral value which is given as an input parameter (see section 4.3). If the activities are not concurrent, the coherence corresponds to the maximum overlap value. In contrast to the overlap, the coherence of two activities is symmetric. Using the maximum allows to take into account that if an activity A is always executed during an activity B, they are strongly related.

4.3 Grouping and Phase Detection

In order to facilitate comparisons between several observations of the same task and to structure the observations, intervals can be combined to groups of intervals. Often tasks have different phases and certain activites only appear in certain phases. An example is given in Figure 4. First, we form groups individually for each example of the scenario. Then we assign groups to phases such that the groups (from all examples) which are similar are assigned to the same phase.

We divide each example into several groups by proceeding forward in time and creating a new group whenever an "unexpected" activity occurs. The execution of an activity is unexpected in regard to the activities of a group if it is usually not executed at the same time as the other activities of the group. If someone holds a speech and interrupts it to answers the phone, this would be an unexpected activity because "holding a speech" and "answering the phone" have a low coherence. The activity "answering the phone" would thus cause the creation of a new group.

A new activity A is unexpected in regard to the activities A_1, ..., A_n already contained in a group and a threshhold alpha if the following conditions hold:

1. $n > 0$
2. $A \notin \{A_1, ...A_n\}$
3. $\frac{1}{n} \sum_{i=1}^{n} coh(A, A_i, alpha) \leq alpha$

The threshhold alpha controls how many groups are created and must be set manually to a value which fits to the scenario to be modeled. Activities which are concurrent to the new activity should not influence the decision whether the new activity is unexpected or not. This can be ensured by using the threshhold as the neutral value for the calculation of coherence.

Each example is divided into several groups by using the following algorithm: The k-th group of the j-th examples is denoted G_k^j. We assume that the time intervals T_i are ordered according to their start time and that $A(T_i)$ is the activity related to time interval T_i. The algorithm works by proceeding forward in time and adding an activity to a group if it is not unexpected, otherwise starting a group:

Initialize $k = 1$; $G_k^j = \{\}$
For all time intervals T_i, $i = 1, \ldots, n$
  If $A(T_i)$ is unexpected with regard to G_k^j
    then $k = k + 1$; $G_k^j = \{A(T_i)\}$
    else $G_k^j = G_k^j \cup A(T_i)$

After each example was divided into several groups, the groups found in the different examples are assigned to phases. The first group is assigned to a first phase. Then for each group we calculate the most similar phase. If its similarity is over a certain threshhold, the group is assigned to this

phase. Else a new phase is created and the group is assigned to the new phase. The similarity between a group G and a phase P is defined as

$$similarity(G,P) = \begin{cases} \frac{m}{n-i} & \text{if n}\neq\text{i} \\ 0 & \text{else} \end{cases}$$

where m is the number of activities which occur in G and P, n is the number of activities which occur in G or P and i is the number of activities which occur either in G or in P and which are concurrent to all other activities in G or P.

The assignment of groups to phases allows to compare the observed tasks on the level of the phases: Which phases appear in all examples? Which are optional? Is there a specific order? In the next section, the acquisition of an ART model on the level of the phases is described.

4.4 Tree Formation

The phases described in the previous section are used to create an ART tree which represents the temporal dependencies between the phases. Similar to [3], we first create a simple model and successively adapt it to incorporate possible variations. By adjusting the relations (for example by replacing "before" relations by "order independence") or adding new nodes (if a phase did not yet occur in the previous examples), an arbitrary number of example sequences can be incorporated into the tree.

If the same phase is part of several steps of a scenario (e.g. the same phase occurs in the preparation and in the main step of a scenario), the question to which step the phase belongs must be clarified before the tree formation step starts (e.g. by renaming all occurences of the phase in the preparation step). This could be done manually or by investigating the context of the phase. For the description of the tree formation step, we require that every phase occurs only in one step of the algorithm (but may be executed several times due to iterations). Garland and Lesh call this the alignment problem and use a corresponding restriction [4].

4.4.1 Create a tree from the first sequence

We first create a model which represents the first sequence. The phases which occur in the first sequence correspond to leaves of the tree. A single example does not show possible variations. This is why in this step all nodes are connected by a "before" relation. When creating the first version of the tree, we take into account the fact that a sequence can contain iterative subsequences which are executed repeatedly. Iterative nodes are marked by the symbol *. The sequence "abcbcd" of successive phases is thus transformed to the tree shown in Figure 6. Similar subsequences can be identified by comparing successive subsequences by the means of edit distance [2] or by comparing whether similar phases occur in the subsequences.

The next step is to find phases which are optional or which may be replaced by other phases. Figure 7 shows how a tree is adapted if a phase i can be replaced by j or k. To find out whether a phase e is optional or can be replaced by other phases, we take into account all occurences of e in the sequences. For each occurence, we determine the predecessor and successor of e. If there are examples in which there are other elments between the predecessor and the succesor, in most cases this means that e can be replaced by these other phases. If the predecessor and successor always either surround e or are subsequent, e is probably optional.

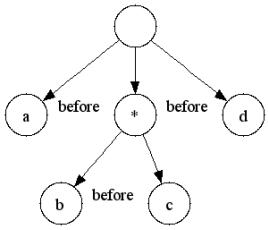


Figure 6: Representation of the sequence abcbcd as a tree

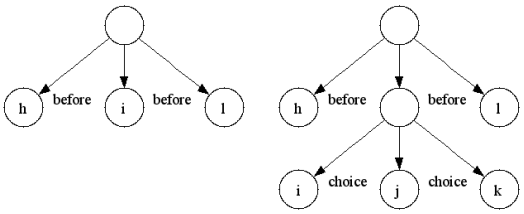


Figure 7: Choice relation

4.4.2 Structure the tree (bottom-up approach)

The next step is to structure the tree hierarchically by a bottom-up approach. Neigboured nodes which represent phases that have the same temporal relation ("before" or "order independence") in all (respectively most if we take into account errors in the sequences) examples are grouped.

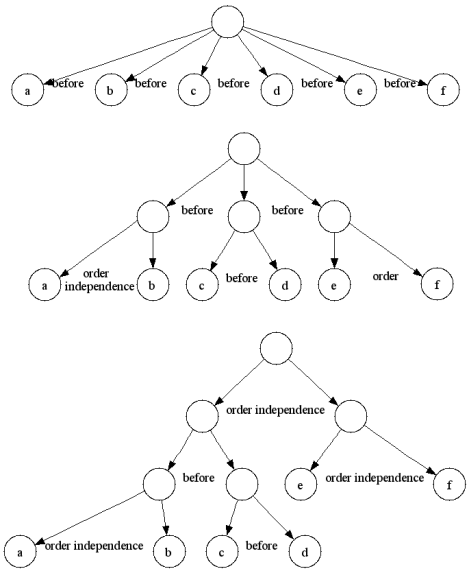


Figure 8: Bottom up approach

We group neighboured nodes in an iterative way from the leaves towards the root of the tree. Figure 8 shows how first leaves and then abstract nodes are grouped and their temporal relationships are adapted. If the example sequences contain neither optional phases nor errors, nodes can be grouped up to the root of the tree.

4.4.3 Adapt the tree (top-down approach)

Our last step to create a tree from sequences of phases corresponds to a top-down approach. It allows to adapt the tree to sequences whether they contain optional phases or not. Additionally, it allows to determine iterations which did not occur in the first sequence and to determine how often each iteration was executed at least and at most. The tree is successively adapted to each sequence. For each sequence we test whether it can already be represented by the tree.

If the sequence contains phases which are not yet represented by the tree, these are added to the tree as a new leaf. New leaves are added next to the corresponding predecessor or successor in the sequence. As the predecessor and successor can be in different subtrees, we have to add the new leaf next to the neighbour to which it is more strongly related (i.e. to the neighbour which is more often executed directly before or after the new phase). The new node can be added next to the chosen neighbour or on a higher level in the hierarchy, depending on whether it always occurs next to the neighbour or also next to phases represented by other nodes in the subtree to which the neighbour belongs.

We divide the sequence in subsequences so that every subsequence corresponds to one of the children of the tree's root node. Given the tree from Figure 9, the sequence "*acbfgeabcdfgfg*" is divided to the subsequences "*acb*", "*fg*", "*e*", "*abc*", "*d*" and "*fgfg*". Each subtree is adapted according to the subsequences of the sequence observed, e.g. the "before" relation which connects *b* and *c* in the first subtree is replaced by "order independence" and *subTree3* as well as the root of the whole tree are marked as iterative.

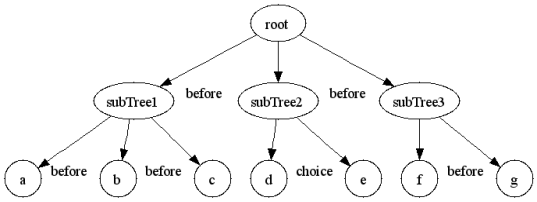


Figure 9: Top down approach

4.5 Detailled Phase Representation

After an ART model was created on the level of phases, the leaves can be replaced by subtrees. Each subtree represents the temporal relations of the activity intervals which belong to this phase. The operators "before", "equal", "contains" and "overlaps" are used to describe the temporal relations between the activitiy intervals. Variations within a phase can be detected by comparing all groups which were assigned to this phase and represented by ART operators like "choice" or "optionality".

5. EVALUATION

The evaluation of the AMPA algorithm, especially of the main phases "Grouping and Phase Detection" and "Tree Formation" ,was done in two steps. In the first step, we estimated the amount of examples that are necessary to correctly reconstruct the model from examples. In the second step, precision and recall of our algorithm was calculated using a number of randomly created ART models. For both tests we required an algorithm that created random examples from a given ART model. Figure 10 shows the source model used in step 1. Two examples created from this model can be seen in Figure 11. The vertical lines indicate boundaries between the different phases.

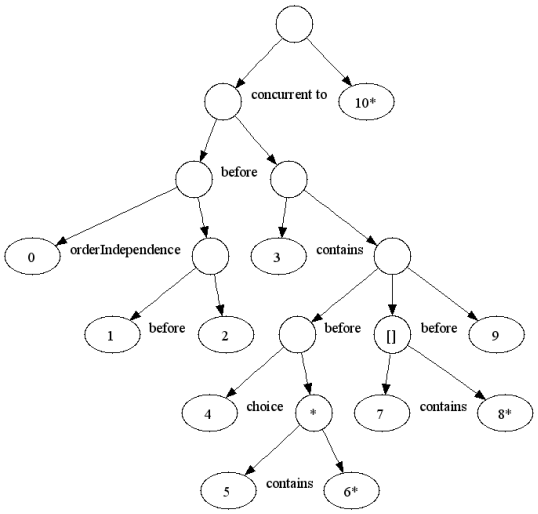


Figure 10: Source model used in the evaluation. Each leaf represents an activity.

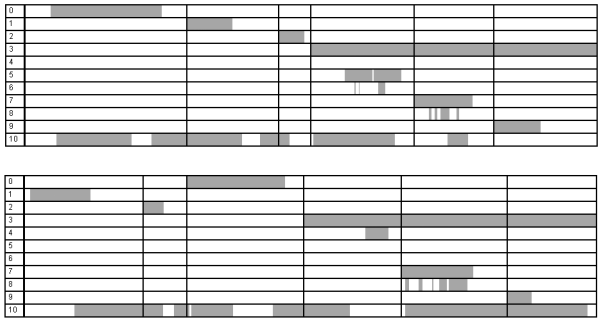


Figure 11: Two example interval datasets created from the source tree in Figure 10.

5.1 Number of Examples

The model in Figure 10 was used to create 20 sets of examples for each of the sizes 5, 10, 15. We applied our model acquisition algorithm to each of the sets and compared the results of the phase detection and tree formation steps with the original model.

Of the sets with 5 examples each, the phases were correctly identified in 14 of 20 tests (70%) and the model was correct in 12 of the tests (60%). Ten examples raised these values to 100% respectively 95% and when we used 15 examples, both steps were correct in all tests. These results show that a small number of examples is sufficient for our model creation algorithm. This is important, if we compare the numbers with similar algorithms in other areas such as linguistics. These algorithms require thousands of examples before good results can be achieved. Therefore, our algorithm is better suited for application areas where it is difficult to capture this many examples. The number of ex-

amples needed for a correct model construction depends on the amount of possible variations.

5.2 Precision and Recall

The substep "tree formation" of our algorithm was further tested with a method described by Solan et al. [11] to calculate precision and recall: A randomly created ART model M_1 was used to create test data records. These are divided into two disjunct subsets. We use the first subset ($C_{Training}$) as input for our model creation algorithm which creates the model M_2. The other subset is called C_{Target}. A third set of examples ($C_{Learner}$) is created from M_2. A data set is accepted by a model if it corresponds to a possible execution of the scenario represented by the model. This is how precision and recall can be calculated:

$$Precision = \frac{nr\ of\ datasets\ of\ C_{Learner}\ accepted\ by\ M_1}{total\ nr\ of\ datasets\ of\ C_{Learner}}$$

$$Recall = \frac{nr\ of\ datasets\ of\ C_{Target}\ accepted\ by\ M_2}{total\ nr\ of\ datasets\ of\ C_{Target}}$$

Table 3: Precision and Recall

P("before")	0.5	0.6	0.6	0.5	0.5
P("order ind.")	0.3	0.3	0.3	0.3	0.25
P("choice")	0.2	0.1	0.1	0.2	0.25
P("iterative")	0.0	0.0	0.0	0.0	0.05
P("optional")	0.0	0.00	0.05	0.05	0.1
Avg. Recall	1.0	0.976	0.937	0.915	0.741
Avg. Precision	0.959	0.886	0.722	0.713	0.523

Table 3 indicates the average precision and recall values for several parameter settings which adhere to realistic scenarios. For every pair of precision and recall, 200 ART models were randomly created. The first three rows of the table show the probability that the corresponding relation was chosen to connect neighboured nodes. P("iterative") and P("optional") are the probabilities that a node was chosen to be iterative or optional. The average size of the randomly created models was 24 nodes. Every ART model was used to create 20 interval data sets for $C_{Training}$ and 10 interval data sets für C_{Target}. The data sets of $C_{Training}$ were used to create an ART model with our algorithm from which 10 interval data sets were generated for $C_{Learner}$.

The precision and recall values show that the tree formation step of our algorithm can be used for the automatic creation of ART models. If the model created by our algorithm differed in any way from the original model (respectively the accepted examples), it was considered to be completely "wrong" when precision and recall were calculated although many times there were only minor differences. The precision and recall values show that in many cases there were no differences at all.

6. CONCLUSION AND FUTURE WORK

We have presented ART, a notation for the temporal dependencies between activities with the focus on parallel activities and an algorithm which can be used for the automatic generation of ART models from example interval data sets. The main concept of our algorithm is the identification of successive phases which reduces the problem of modeling parallel activities to the simpler task of modeling sequential data. The paper defines concepts like the overlap and the coherence of two activities that are the basis for the phase detection algorithm. An evaluation of the algorithm showed that precise models can be generated by using less than 20 examples.

The focus of future work will be the question how identical phases in different substeps of a scenario can be distinguished (e.g. by taking into account the context of a phase). Another interesting aspect is the comparison of ART models: How can the differences between two ART models be calculated and how can two ART subtrees be compared to each other?

7. REFERENCES

[1] J. F. Allen. Maintaining knowledge about temporal intervals. *Commununications of the ACM*, 26(11):832–843, 1983.

[2] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. Wiley-Interscience, 2nd edition, 2001.

[3] V. Eyharabide and A. Amandi. Automatic task model generation for interface agent development. *Inteligencia Artificial*, 9(26):49–57, 2005.

[4] A. Garland and N. Lesh. Learning hierarchical task models by demonstration. Technical Report TR-2003-01, Mitsubishi Electric Research Laboratories, 2003.

[5] M. Giersich, P. Forbrig, G. Fuchs, T. Kirste, D. Reichart, and H. Schumann. Towards an integrated approach for task modeling and human behavior recognition. In J. A. Jacko, editor, *HCI (1)*, volume 4550 of *Lecture Notes in Computer Science*, pages 1109–1118. Springer, 2007.

[6] H. R. Hartson and P. D. Gray. Temporal aspects of tasks in the User Action Notation. Technical Report TR-91-22, Virginia Polytechnic Institute & State University, USA, 1991.

[7] B. John and D. Kieras. The GOMS family of user interface analysis techniques. *ACM Transactions on Computer-Human Interaction*, 3(4), 1996.

[8] T. Klug. Computer aided observations of complex mobile situations. In *CHI '07: CHI '07 extended abstracts on Human factors in computing systems*, pages 2507–2512. ACM Press, 2007.

[9] F. Moerchen. *Time Series Knowledge Mining*. PhD thesis, Philipps-University Marburg, Germany, 2006.

[10] G. Mori, F. Paternò, and C. Santoro. Ctte: support for developing and analyzing task models for interactive system design. *IEEE Transactions on Software Engineering*, 28(8):797–813, 2002.

[11] Z. Solan, D. Horn, E. Ruppin, and S. Edelman. Unsupervised learning of natural languages. *PNAS*, 102(33):11629–11634, 2005.

[12] J.-C. Tarby and M.-F. Barthet. The DIANE+ Method. In *Proceedings of the Second International Workshop on Computer-Aided Design of User Interfaces (CADUI)*, pages 95–120, 1996.

[13] Y. Yao. A Petri net model for temporal knowledge representation and reasoning. *IEEE Transactions on Systems Man and Cybernetics*, 24(9):1374–1382, 1994.