Scalable Dynamic Business Process Discovery with the Constructs Competition Miner

David Redlich[1,2], Thomas Molka[1,3], Wasif Gilani[1], Gordon Blair[2], and Awais Rashid[2]

[1] SAP Research Center Belfast, United Kingdom,
[david.redlich|thomas.molka|wasif.gilani]@sap.com
[2] Lancaster University, United Kingdom,
[gordon|marash]@comp.lancs.ac.uk
[3] University of Manchester, United Kingdom

Abstract. Since the environment for businesses is becoming more competitive by the day, business organizations have to be more adaptive to environmental changes and are constantly in a process of optimization. Fundamental parts of these organizations are their business processes. Discovering and understanding the actual execution flow of the processes deployed in organizations is an important enabler for the management, analysis, and optimization of both, the processes and the business. This has become increasingly difficult since business processes are now often dynamically changing and may produce hundreds of events per second. The basis for this paper is the Constructs Competition Miner (CCM): A divide-and-conquer algorithm which discovers block-structured processes from event logs possibly consisting of exceptional behaviour. In this paper we propose a set of modifications for the CCM to enable scalable dynamic business process discovery of a run-time process model from a stream of events. We describe the different modifications and carry out an evaluation, investigating the behaviour of the algorithm on event streams of dynamically changing processes.

Key words: run-time models, business process management, process mining, complex event processing, event streaming, big data

1 Introduction

The success of modern organizations has become increasingly dependent on the efficiency and performance of their employed business processes (BPs). These processes dictate the execution order of singular tasks to achieve certain business goals and hence represent fundamental parts of most organizations. In the context of business process management, the recent emergence of Big Data yields new challenges, e.g. more analytical possibilities but also additional run-time constraints. An important discipline in this area is Process Discovery: It is concerned with deriving process-related information from event logs and, thus, enabling the business analyst to extract and understand the actual behaviour of a business process. Even though they are now increasingly used in commercial settings, many of the developed process discovery algorithms were designed to work in a static fashion, e.g. as provided by the ProM framework [15], but are

not easily applicable for processing real-time event streams. Additionally, the emergence of Big Data results in a new set of challenges for process discovery on event streams, for instance [11, 16]: (1) *diversity of event formats* from different sources, (2) *high event frequency* (e.g. thousands of events per second), and (3) *less rigid processes* (e.g. BPs found on the operational level of e-Health and security use-cases are usually subjected to frequent changes).

With the focus on addressing the latter two of these challenges, we propose in this paper modifications for the Constructs Competition Miner (CCM) [10] to enable Scalable Dynamic Process Discovery as proposed in [11]. The CCM is a process discovery algorithm that follows a divide-and-conquer approach to directly mine a block-structured process model which consists of common BP-domain constructs and represents the main behaviour of the process. This is achieved by calculating global relations between activities and letting different *constructs* compete with each other for the most suitable solution from top to bottom using "soft" constraints and behaviour approximations. The CCM was designed to deal with noise and not-supported behaviour. To apply the CCM on event streams the algorithm was split up into two individually operating parts:

1. **Run-time footprint calculation**, i.e. the current footprint[1], which represents the abstract "state" of the system, is updated with occurrence of each event. Since every occurring event constitutes a system state transition, the algorithmic execution-time needs to be kept to a minimum.
2. **Scheduled footprint interpretation**, i.e. from the footprint the current business process is discovered in a scheduled, reoccurring fashion. Since this part is executed in a different lifecycle it has less execution-time constraints. In this step the abstract "computer-centric" footprint is transformed into a "human-centric" business process representation.

The remainder of this paper provides essential background information (Section 2), a discussion of related work (Section 3), a summarized description of the original CCM (Section 4), the modifications that were carried out on top of the CCM to enable Scalable Dynamic Process Discovery (Section 5), an evaluation of the behaviour of the resulting algorithm for event streams of dynamically changing processes (Section 6), and an outlook of future work (Section 7).

2 Background

Business Processes are an integral part of modern organizations, describing the set of activities that need to be performed, their order of execution, and the entities that execute them. Prominent BP examples are Order-to-Cash or Procure-to-Pay. According to Ko et al. BPs are defined as *"...a series or network of value-added activities, performed by their relevant roles or collaborators, to purposefully achieve the common business goal"* [4]. A BP is usually described by a *process model* conforming to a business process standard, e.g. Business Process Model and Notation (BPMN) [9], or Yet Another Workflow Language (YAWL) [13]. In this paper, we will focus on business processes consisting of a set of common

[1] footprint is a term used in the process discovery domain, abstractly representing existent "behaviour" of a log, e.g. activity "a" is followed by activity "b"

control-flow elements, supported by most of the existing BP standards: start and end events, activities (i.e. process steps), parallel gateways (AND-Split/Join), and exclusive gateways (XOR-Split/Join) (see [9, 13]). In Figure 1 an example process involving all the introduced elements is displayed. Formally, we define a business process model as follows [10]:

Definition 1 *A business process model is a tupel* $BP = (A, S, J, E_s, E_e, C)$ *where* A *is a finite set of activities,* S *a finite set of splits,* J *a finite set of joins,* E_s *a finite set of start events,* E_e *a finite set of end events, and* $C \subseteq F \times F$ *the path connection relation, with* $F = A \cup S \cup J \cup E_s \cup E_e$*, such that*

- $C = \{(c_1, c_2) \in F \times F \mid c_1 \neq c_2 \wedge c_1 \notin E_e \wedge c_2 \notin E_s\}$,
- $\forall a \in A \cup J \cup E_s : |\{(a, b) \in C \mid b \in F\}| = 1$,
- $\forall a \in A \cup S \cup E_e : |\{(b, a) \in C \mid b \in F\}| = 1$,
- $\forall a \in J : |\{(b, a) \in C \mid b \in F\}| \geq 2$,
- $\forall a \in S : |\{(a, b) \in C \mid b \in F\}| \geq 2$, *and*
- *all elements* $e \in F$ *in the graph* (F, C) *are on a path from a start event* $a \in E_s$ *to an end event* $b \in E_e$.

For a block-structured BP model it is furthermore required that the process is hierarchically organised [10], i.e. it consists of unique join-split-pairs, each representing either a single entry or a single exit point of a non-sequential BP construct, e.g. Choice, Parallel, Loop, etc. The example process in Figure 1 is a block-structured process. A similar representation gaining popularity in recent years is the *process tree*, as defined based on Petri nets/workflow nets in [5].

When a business process is automatically or semi-automatically executed with a BP execution engine, e.g. with a Business Process Management System (BPMS), an *event log* is produced, i.e. a all occurred events are logged and stored. These logs and their contained events may capture different aspects of a process execution, e.g. a different granularity of events are logged. In this paper however, we only focus on a minimal set of event features: In order to allow the discovery of the control-flow, every event is required to have a reference (1) to the associated *process instance* and (2) to the corresponding *activity*. Furthermore, we assume that the log contains exactly one event for each activity execution, i.e. activity lifecycle events are not regarded. All events resulting from the execution of the same *process instance* are captured in one *trace*. A trace is assumed to be independent from other traces, i.e. the execution order of a process instance is not in any way dependent on the execution of a second instance. Accordingly, an event e is represented by a pair $e = (t, a)$ where $t \in \mathbf{N}$ is the unique identifier of the trace and $a \in A$ is a unique reference to the executed activity.

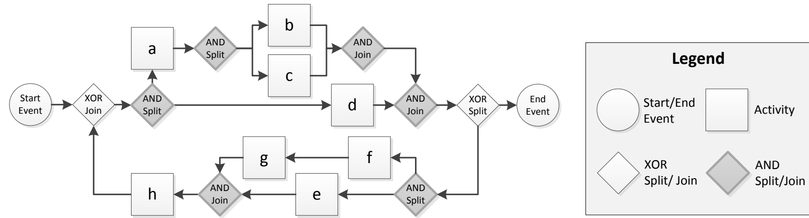

Fig. 1. Example business process with all element types included

The research area of *Process Discovery* is concerned with the extraction of a business process model from event logs without using any a-priori information [17]. Conventional challenges in process discovery originate from the motivation to achieve a high quality of results, i.e. discovered processes should support as accurately as possible the behaviour contained in the log. In particular that means, process discovery algorithms have to deal with multiple objectives, e.g. precision, simplicity, fitness - over-fitting vs. under-fitting (see [17]). Process discovery algorithms are usually assumed to be carried out in an static way as an "offline" method. This is reflected by the fact that the input for these algorithms is an entire log as conceptually shown by the following definition:

Definition 2 *Let the log $L_n = [e_0, e_1, ...e_n]$ be a sequence of $n+1$ events ordered by time of occurrence ($\forall i < j \wedge e_i, e_j \in L_n : time(e_i) \leq time(e_j)$) and BP_n be the business process model representing the behaviour in L_n, then process discovery is defined as a function that maps a log L_n to a process BP_n:*

$$ProcessDiscovery : [e_0, e_1, ..., e_n] \Rightarrow BP_n$$

3 Related Work

A large number of process discovery algorithms exist, e.g. Inductive Miner [5], HeuristicsMiner [19], alpha-miner [14] and CCM [10]. These and many algorithms have in common that at first a *footprint* of the log is created based on which the process is constructed. Similar to the CCM, the following related algorithms also discover block-structured processes: (1) Genetic process discovery algorithms that restrict the search space to block-structured process models, e.g. [2]. However, these are non-deterministic and generally have a high execution time due to exponentially expanding search space. (2) Another relevant approach that is conceptually similar to the CCM is proposed in [5], the Inductive Miner (IM): A top-down approach is applied to discover block-structured Petri nets. The original algorithm evaluates constraints based on local relationships between activities in order to identify the representing construct in an inductive fashion. In recent work, the IM has also been extended to deal with noise [6]. Generally, in all discovery approaches based on footprints known to the authors the footprint is represented by a *direct neighbours* matrix representing information about the local relations between the activities, e.g. for the BP of Figure 1: h can only appear *directly* after g or e. As discussed in Section 4 the CCM on the other hand extracts the process from a footprint based on global relations between activities, e.g. h appears *at some point* after g or e.

However, of little importance for conventional process discovery algorithms is their practicality with regards to an application during run-time: as defined in Definition 2 process discovery is a static method that analyses an event log in its entirety. An alternative to this approach is the immediate processing of events when they occur to information of an higher abstraction level in order to enable a real-time analysis. This approach is called Complex Event Processing (CEP): a method that deals with the event-driven behaviour of large, distributed enterprise systems [7]. More specifically, in CEP events produced by the systems are captured, filtered, aggregated, and finally abstracted to complex events

representing high-level information about the situational status of the system, e.g. performance, control-flow, etc. The need for monitoring aspects of business processes at run-time by applying CEP methodologies has been identified by Ammon et al., thus coining the term Event-Driven Business Process Management (EDBPM) - a combination of two disciplines: Business Process Management (BPM) and Complex Event Processing [1]. The dynamic process discovery solution proposed in this paper is an application of EDBPM (see Section 5).

In the context of process discovery, an often used term for discovering processes from event streams is *Streaming Process Discovery*. In [3] the HeuristicsMiner has been modified for this purpose by maintaining queues of fixed size $n \in \mathbf{N}$ containing the latest n events, i.e. the queues function as a "sliding window" over the event stream. Three different approaches of how to process these queues to a footprint have been proposed: (1) Stationary - every queue entry has the same weight, (2) Ageing - older entries have a decreasing weight, and (3) Self-Adapting Ageing - the factor with which the influence of older entries decreases is dependent on whether a concept drift[2] has been detected (quickly decreasing) or the process is assumed to be stationary (slowly decreasing). Additionally, Lossy Counting, a technique using approximate frequency count, has been investigated as a modification. A second approach for discovering concept drifts on event streams is presented in [8]: an incremental discovery of declarative process models using the stationary approach and Lossy Counting.

4 Static Constructs Competition Miner

The CCM as described in [10] is a deterministic process discovery algorithm that operates in a static fashion and follows a divide-and-conquer approach which, from a given event log, directly mines a block-structured process model that represents the main behaviour of the process. The CCM has the following main features [10]: (1) A deadlock-free, block-structured business process without duplicated activities is mined; (2) The following BP constructs are supported and can be discovered for single activities: Normal, Optional, Loopover, and Loopback; or for a set of activities: Choice, Sequence, Parallel, Loop, Loopover-Sequence, Loopover-Choice, Loopover-Parallel (see Figure 2), and additionally all of them as optional constructs - these are constructs supported by the majority of business process standards like BPMN or YAWL; (3) If conflicting or exceptional behaviour exists in the log, the CCM picks the "best" fitting BP construct.

Algorithm 1 shows the conceptual methodology of the CCM algorithm in pseudocode. The CCM applies the divide-and-conquer paradigm and is implemented in a recursive fashion (see lines 7, 16, and 17). At the beginning `getFootprintAndBuildConstruct` is initially called for all involved activities ($A_m = A$) with the process *bp* consisting of only a start and end element. The recursive function is first creating a footprint *fp* from the given log L only considering the activities specified in set A_m (at the beginning all involved activities). In a next step it will be decided which is the best construct to represent the behaviour captured by *fp*: (1) if the activity set A_m only consists of one element,

[2] A concept drift in this context is a behavioural change in the monitored process

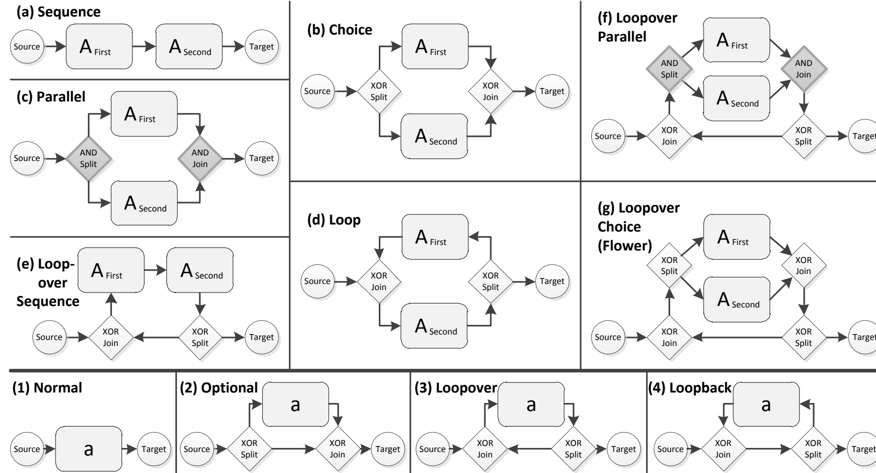

Fig. 2. Business Process Constructs Supported by the CCM [10]

Algorithm 1: Methodology of the CCM in Pseudocode

```
   Data: Log L
   Result: BP bp
 1 begin
 2   A ← getSetOfAllActivitiesInLog(L);
 3   BP bp ← buildInitialBPWithStartAndEnd();
 4   bp ← getFootprintAndBuildConstruct(A, L, bp);
 5   return bp;

 6 Function getFootprintAndBuildConstruct(A_m, Log L, BP bp)
 7   Footprint fp = extractFootprintForActivities(A_m, L);
 8   if |A_m| = 1 then
 9     Construct c ← analyseConstructForSingleActivity(fp);
10     bp ← createSingleActivityConstruct(c, A_m);
11   else
12     ConstructsSuitability[] cs ← calculateSuitabilityForConstructs(fp, A_m);
13     (Construct c, A_first, A_second) ← constructCompetition(cs, A_m);
14     bp ← createBlockConstruct(c, bp);
15     bp ← getFootprintAndBuildConstruct(A_first, L, bp);
16     bp ← getFootprintAndBuildConstruct(A_second, L, bp);
17   return bp;
```

it will be decided which of the single activity constructs (see bottom of Figure 2) fits best - the process bp will then be enriched with the new single activity construct (see line 11); (2) If the activity set A_m contains more than one element, the suitability for each of the different constructs is calculated for any two activities $x, y \in A_m$ based on "soft" constraints and behaviour approximations, e.g. activities a and b are in a strong Sequence relationship. The result of this calculation (line 13) is a number of suitability matrices, one for each construct. In the subsequent competition algorithm it is determined what is the best combination of (A) the construct type $c \in \{Sequence, Choice, Loop, ...\}$, and (B) the two subsets A_{first} and A_{second} of A_m with $A_{first} \cup A_{second} = A_m$, $A_{first} \cap A_{second} = \{\}$, and $A_{first}, A_{second} \neq \{\}$, that best accommodate all x, y-pair relations of the corresponding matrix of construct c (line 14). The construct is then created and

added to the existing process model *bp* (line 15), e.g. XOR-split and -join if the winning construct *c* was *Choice*. At this stage the recursive method calls will be executed to analyse and construct the respective behaviour for the subsets A_{first} and A_{second}. The split up of the set A_m continues in a recursive fashion until it cannot be divided any more, i.e. the set consists of a single activity (see case (1)). The process is completely constructed when the top recursive call returns.

Of particular interest for the transformation of the CCM algorithm to a solution for scalable dynamic process discovery is the composition of the footprint and its calculation from the log. As opposed to many other process discovery algorithms, e.g. alpha-miner [14], the footprint does not consist of absolute relations, e.g. h is followed by a (see example in Figure 1), but instead holds relative relation values, e.g. a is eventually followed by g in $0.4 \cong 40\%$ of the traces. Furthermore, the footprint only contains global relations between activities in order to guarantee a low polynomial execution time for the footprint interpretation [10]. The footprint of the CCM contains information about: (1) the occurrence of each involved activities $x \in A_m$, i.e. how many times x appears at least once per trace, how many times an x appears on average per trace, and how many times the trace started with x; (2) the global relations of each activity pair $x, y \in A_m$, i.e. in how many traces x appears sometime before the *first* occurrence of y in the trace, and in how many traces x appears sometime before *any* occurrence of y in the trace[3]. All measures in the footprint are relative to the number of traces in the log. Furthermore, not only one overall footprint is created for the CCM but also for every subset A_{first} and A_{second}, that is created during execution, a new sub-footprint is created (see Algorithm 1).

5 Dynamic Constructs Competition Miner

As established in Section 1, increasingly dynamic processes and the need for immediate insight require current research in the domain of process mining to be driven by a set of additional challenges. To address these challenges the concept of Scalable Dynamic Process Discovery (SDPD), an interdisciplinary concept employing principles of CEP, Process Discovery, and EDBPM, has been introduced in [11]: "SDPD describes the method of monitoring one or more BPMSs in order to provide at any point in time a reasonably accurate representation of the current state of the processes deployed in the systems with regards to their control-flow, resource, and performance perspectives as well as the state of still open traces." That means, any potential changes in the mentioned aspects of the processes in the system that occur during run-time have to be recognized and reflected in the continuously updated "current state" of the process. Due to its purpose, for solutions of SDPD an additional set of requirements applies. For this paper, the most relevant of them are [11]:

– *Detection of Change*: An SDPD solution is required to detect change in two different levels defined in [12]: (1) Reflectivity: A change in a process instance

[3] This stands in contrast to existing discovery solutions since in the CCM the footprint and its interpretation is not based on *local* relationships between activity occurrences, e.g. direct neighbours, but based on *global* relationships between them.

(trace), i.e. every single event represents a change in the state of the associated trace. (2) Dynamism: A change on the business process level, e.g. because events/traces occurred that contradicts with the currently assumed process.
– *Scalability/Algorithmic Run-time*: An SDPD solution is applied as CEP concept and has to be able deal with large business processes operating with a high frequency, i.e. the actual run-time of the algorithms becomes very important. Additionally, the key algorithms are required to be scalable to cope with increasing workload at minimal possible additional computational cost.

Motivated by these challenges the initial process discovery approach was altered to allow for dynamic process discovery. As opposed to the traditional *static* methodology (see Definition 2), *dynamic* process discovery is an iterative approach as defined in the following:

Definition 3 *Let log* $L_n = [e_0, e_1, ...e_n]$ *be a sequence of* $n+1$ *events ordered by time of occurrence (* $\forall i < j \wedge e_i, e_j \in L_n : time(e_i) \leq time(e_j)$*) and* BP_n *be the business process model representing the behaviour in* L_n*, then dynamic process discovery is defined as a function that projects the tuple* (e_n, BP_{n-1}) *to* BP_n*:*

$$DynamicProcessDiscovery : (e_n, BP_{n-1}) \Rightarrow BP_n$$

As described in Section 4, the CCM is a static mining algorithm and has to be modified in order to enable SDPD. The result of this modifications is called Dynamic CCM (DCCM). However, two restrictions for the DCCM with regards to the previously mentioned requirements of SDPD apply: (1) instead of discovering change on the BP perspectives control-flow, resources, and performance perspective, the DCCM described in this paper only focuses on discovering change in the control-flow, and (2) only change on the abstraction level of Dynamism is detected, i.e. whether or not the control-flow of the process has changed - the detection of change on the abstraction level of Reflectivity will not be supported by the DCCM. Additionally to the requirements of SDPD the DCCM features the following important aspects: (1) *robust*: if conflicting, exceptional, or not representable behaviour occurs in the event stream, the DCCM does not fail but always picks the BP construct that best accommodates the recorded behaviour; (2) *deterministic*: the DCCM yields the exact same output BP for the same input stream of events.

The following modifications were applied to the default CCM to create the DCCM and are described in more detail in the remainder of this section:

1. Splitting up the algorithm in two separate parts: one for dynamically updating the current footprint(s) complying to the scalability requirement, and one for interpreting the footprint into a BP model which has less restrictions with regards to its execution-time.
2. In the CCM the footprint is calculated in relation to all occurring traces. This is not applicable for SDPD since the number of traces should not have an influence on the execution-time of any component of an SDPD solution. For this reason the footprint has to be calculated in a dynamic fashion, i.e. an event-wise footprint update independent from the previously occurred number of events or traces.

3. The original behaviour of the CCM to carry out a footprint calculation for every subset that has been created by the divide-and-conquer approach is not optimal as then the DCCM would have to extract up to $2 * n + 1$ different footprints if only one activity was split-up from the main set for each recursion.[4] This has been improved for the DCCM: for the most common constructs Choice and Sequence the sub-footprints are automatically derived from the parent footprint.
4. In rare cases it can happen that for every appearing event the state of the process is alternating between a number of different control-flows. This is caused by "footprint equivalent" BP models, i.e. two models are footprint equivalent if they both express the behaviour captured by the footprint. We introduce a measure which favours the last control-flow state in order to prevent the described behaviour.

5.1 Methodology of the Dynamic CCM

The original CCM algorithm had to be split up into two separate parts in order to comply to the scalability requirement of SDPD. A component triggered by the occurrence of a new event to update the dynamic footprint and a component decoupled from the event processing which interprets the footprint into a BP Model. The conceptual methodology of the DCCM is depicted in Figure 3. The components, models, and functionality of the DCCM are described in the following: Events from the monitored *Enterprise System*, in which the end-to-end process is deployed, are fed into an event stream. The *Footprint Update* component is the receiver of these events and processes them directly into changes on the overall *Dynamic Footprint* which represents the abstract state of the monitored business process. If additional footprints for subsets of activities are required as specified by the *Sub-Footprint Configurations*, e.g. if a Loop or Parallel construct was identified, then these sub-footprints are also updated (or created if they were not existent before). The *Dynamic Footprint(s)* can then at any point in time be compiled to a human-centric representation of the business process by the *Footprint Interpretation* component, i.e. the abstract footprint representation is interpreted into knowledge conforming to a block-structured BP model. In the DCCM this interpretation is scheduled dependent on how many new completed traces appeared, e.g. the footprint interpretation is executed once every 10 terminated traces. If the *interpretation frequency* $m \in \mathbb{N}$ of the DCCM is set to 1 a footprint interpretation is executed for every single trace that terminated. The *Footprint Interpretation* algorithm works similar to the CCM algorithm shown in Algorithm 1; but instead of extracting footprints from a log (line 8), the modified algorithm requests the readily available *Dynamic Footprint(s)*. If a sub-footprint is not yet available (e.g. at the beginning or if the process changed) the *Footprint Interpretation* specifies the request for a sub-footprint in the *Sub-Footprint Configurations* in the fashion of a feedback loop.

[4] e.g. for $A = \{a,b,c,d\} : (a,b,c,d) \rightarrow ((a,b,c),(d)) \rightarrow (((a),(b,c)),(d)) \rightarrow (((a),((b),(c))),(d))$, seven different footprints for sets $\{a,b,c,d\}$, $\{a,b,c\}$, $\{b,c\}$, $\{a\}$, $\{b\}$, $\{c\}$, $\{d\}$ need to be created - $(,)$ denote the nested blocks that emerge while splitting the sets recursively.

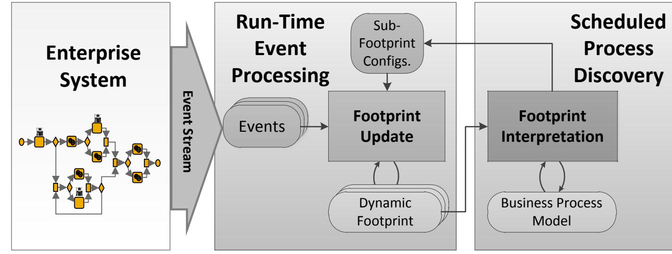

Fig. 3. Conceptual Methodology of the Dynamic CCM

Thus, *Sub-Footprint Configurations* and *Dynamic Footprints* act as interfaces between the two components, *Footprint Update* and *Footprint Interpretation*. The *Footprint Interpretation* cannot continue to analyse the subsets if no sub-footprint for these exist yet. In this case, usually occurring in the warm-up or transition phase, an intermediate BP model is created with activities containing all elements of the unresolved sets as depicted in Figure 4.

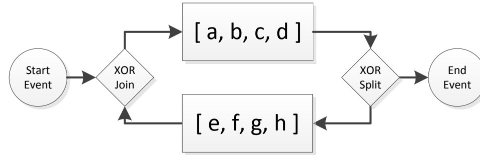

Fig. 4. Result of the *Footprint Interpretation* on an event stream produced by the example from Figure 1 if no sub-footprints for $\{a, b, c, d\}$ and $\{e, f, g, h\}$ are available yet - only the top-level loop has been discovered

5.2 Run-time Update of the Dynamic Footprint

The *Footprint Update* component processes events to changes in the *Dynamic Footprint*, i.e. updates the abstract representation of the process state. The original footprint extraction of the CCM algorithm calculates all values in relation to the number of occurred traces, i.e. every trace's influence on the footprint is equal: $\frac{1}{|traces|}$. To comply to the scalability requirement of SDPD the footprint update calculation should only take a fixed amount of time, independent from the total number of previously occurred events or traces. An increase of the total number of involved activities can cause, however, a linear increase of the execution-time due to the recalculation of the relations between the occurred activity and, in the worst case, all other activities. The independence from previous traces is the reason the footprint is calculated in a dynamic fashion, i.e. the dynamic footprint is incrementally updated in a way that older events "age" and thus have less influence than more recent events.

The ageing approach that is utilized in the *Footprint Update* of the DCCM is the creation of an individual *trace footprint*[5] (*TFP*) for each trace and add it multiplied by the *trace influence factor* $t_{if} \in \mathbb{R}$ to the current dynamic overall footprint (*DFP*) multiplied by $1 - t_{if}$, e.g. for $t_{if} = 0.01$:

[5] the occurrence values for activities as well as the global relations (see end of Section 4) are represented in the trace footprint by absolute statements *true* $\equiv 1$ if it occurred and *false* $\equiv 0$ if not

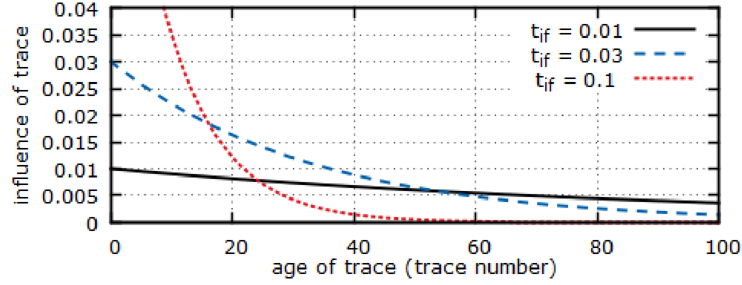

Fig. 5. Development of the influence of a trace for different trace influence factors(t_{if})

$DFP = 0.01 * TFP + 0.99 * DFP$. That means, a trace footprint TFP_i has at the beginning the influence of 0.01, after another TFP_{i+1} has been added the influence of TFP_i decreases to $0.01 * 0.99$, and after another $0.01 * 0.99^2$ and so on. By applying this incremental method, older TFP are losing influence in the overall dynamic footprint. Figure 5 shows how the influence of a trace is dependent on its "age": If $t_{if} = 0.1$, the influence of a trace that appeared 60 traces ago became almost irrelevant. At the same time if $t_{if} = 0.01$ the influence of a trace of the same age is still a little more than half of its initial influence when it first appeared. Essentially, the purpose of the *trace influence factor* t_{if} is to configure the "memory" and adaptation rate of the footprint update component.

Another important dynamism feature that had to be implemented was the possibility to add an activity that has not appeared before. A new activity is first recorded in the respective trace footprint. When the trace is terminated it will be added to the overall footprint in which it is not contained yet. The factored summation of both footprints to build the new dynamic footprint is carried out by assuming that a not previously in the dynamic overall footprint contained relation value is 0. An exception of this behaviour is the "warm-up" phase of the *Footprint Update*, i.e. if the amount of occurred traces is $< \frac{1}{t_{if}}$ then the influence of the dynamic footprint is $\frac{|traces|}{|traces|+1}$ and of the trace footprint $1 - \frac{|traces|}{|traces|+1}$. For instance if $t_{if} = 0.01$ and $|traces| = 9$ then is a new dynamic footprint calculated with $DFP_{10} = \frac{1}{10} * TFP + \frac{9}{10} * DFP_9$ and for the next trace $DFP_{11} = \frac{1}{11} * TFP + \frac{10}{11} * DFP_9$. Because of this implementation the "warm-up" phase of the *Footprint Update* could be drastically reduced, i.e. processes were already completely discovered a few traces after the start of the monitoring.

Furthermore, activities that do not appear any more during operation should be removed from the dynamic footprint. This was implemented in the DCCM in the following way: If the *occurrence once* value of an activity drops below a removal threshold $t_r \in \mathbb{R}, t_r < t_{if}$ it will be removed from the dynamic footprint, i.e. all values and relations to other activities are discarded.

The fact that especially many Choice and Sequence constructs are present in common business processes, motivates an automated sub-footprint creation in the *Footprint Interpretation* based on the parent footprint rather then creating the sub-footprint from the event stream. This step helps to decrease the execution-time of the *Footprint Update* and was achieved by introducing an ex-

tra relation to the footprint[6] - the *direct neighbours* relation as used by other mining algorithms (see Section 3). In the *Footprint Interpretation* this relation is then used for creating the respective sub-footprints for Sequence and Choice constructs but not for identifying BP constructs since the *direct neighbours* relation does not represent a global relation between activities.

5.3 Modifications in the Footprint Interpretation Component

As analysed in the beginning of this section, the original behaviour of the CCM to retrieve a sub-footprint for each subset that has been created by the divide-and-conquer approach is not optimal. This is why, in the *Footprint Interpretation* the DCCM calculates the sub-footprints for the most common constructs, Choice and Sequence, from the available parent footprint: (1) For the Choice construct the probability of the exclusive paths are calculated with $p_{first} = \sum_{x \in A_{first}} Fel(x)$ and $p_{second} = \sum_{x \in A_{second}} Fel(x)$ with $Fel(x)$ being the occurrences of x as first element (see CCM footprint description in Section 4). Then the relevant values of the parent footprint are copied into their respective new sub-footprints and normalized, i.e. multiplied with $\frac{1}{p_{first}}$ and $\frac{1}{p_{second}}$, respectively. (2) The sub-footprints for the Sequence construct are similarly built, but without the normalization. Instead, the direct neighbours relation, now also part of the dynamic footprint, is used to calculate the new overall probabilities of the sub-footprints.

If two or more BP constructs are almost identically suitable for one and the same footprint, a slight change of the dynamic footprint might result in a differently discovered BP. This may cause an alternating behaviour for the footprint interpretation, i.e. with almost every footprint update the result of the interpretation changes. This is undesirable behaviour which is why the competition algorithm was additionally modified as follows: All combinations of BP construct and subsets are by default penalized by a very small value, e.g. $\frac{t_{if}}{10}$, with the exception of the combination corresponding to the previously discovered BP model, hence reducing the risk of discovering alternating BP models.

6 Evaluation

The static CCM algorithm has been tested for its accuracy in [10]: (1) in a qualitative analysis the CCM was able to rediscover 64 out of 67 processes for which a log was produced through simulation. (2) in the second part of the evaluation the discovery performance of the CCM was compared to the mining algorithms HeuristicsMiner (HM) [19], Inductive Miner (IM) [6], and the Flower Miner (FM), all of which are readily available in the ProM nightly build [15]. For ten given logs (including real-life logs and publicly available logs) the results of the algorithms (each configured with their default parameters) were evaluated for their *trace fitness* f_{tf}, *precision* f_{pr}, *generalization* f_g, and *simplicity* f_s with the help of the *PNetReplayer* plugin [18]. The averaged results of this analysis are shown in Table 1; Note, that a lower simplicity value is better.

[6] In rare cases (if Loop and Parallel constructs dominate) this modification can have a negative effect on the execution-time since extra information needs to be extracted without the benefit of mining less sub-footprints

Table 1. Conformance results of the different discovery algorithms

Trace Fitness f_{tf}				Precision f_{pr}				Generalization f_g				Simplicity f_s			
HM	IM	FM	CCM	HM	IM	FM	CCM	HM	IM	FM	CCM	HM	IM	FM	CCM
0.919	0.966	1.0	0.979	0.718	0.622	0.124	0.663	0.941	0.915	0.992	0.930	155.3	122.8	56.4	111.9

In the remainder of this section early evaluation results of the DCCM are presented with regards to its capability of detecting certain basic changes of a real-time monitored business process. The basis of this evaluation is the example model in Figure 1 which is simulated and the resulting event stream fed into the DCCM. The CCM core is again configured with its default noise parameters. Figure 6 shows the different values we want to measure. In the figure BP_1 and BP_2 are the business processes deployed in the monitored system and BP'_1 to BP'_n are the discovered models by DCCM. Additionally, BP_1 and BP'_m are equivalent ($BP_1 \equiv BP'_m$) as well as BP_2 and BP'_n ($BP_2 \equiv BP'_n$). For this part of the evaluation the following measures are of interest:

- Warm-up: $t_w \in \mathbb{N}$ the amount of completed traces the DCCM needs as input at the start until the resulting model equivalently represents the process in the system, i.e. until $BP_1 \equiv BP'_m$.
- Change Detection: $t_d \in \mathbb{N}$ the amount of completed traces it takes to detect a certain change in the monitored process - from the point at which the process changed in the system to the point at which a different process was detected. When the change is detected the newly discovered process is usually not equivalent to the new process in the system BP_2 but instead represents parts of the behaviour of both processes, BP_1 and BP_2.
- Change Transition Period: $t_tr \in \mathbb{N}$ the amount of completed traces it takes to re-detect a changed process - from the point at which the process change was detected to the point at which the correct process representation was identified, i.e. until $BP_2 \equiv BP'_n$. In this period multiple different business processes may be detected, each best representing the dynamic footprint at the respective point in time.

The first test will evaluate how the DCCM behaves at the beginning when first exposed to the event stream, more particularly, we want to determine t_w. Figure 7 shows a selection of the first few bp models extracted with *trace influence factor* $t_{if} = 0.01$ (see Section 5.2) and *interpretation frequency* $m = 10$, i.e. an interpretation is executed every 10 completed traces: After the first trace the discovered process is a sequence reflecting the single trace that defines the process at that point in time. At trace 10, which is the next scheduled footprint interpretation, the the algorithm discovered a Loop construct but cannot further analyse the subsets since the corresponding sub-footprint was not requested yet. Because of that, the feedback mechanism via the *Sub-Footprint Configurations* is utilized by the *Footprint Interpretation* algorithm to register the creation of

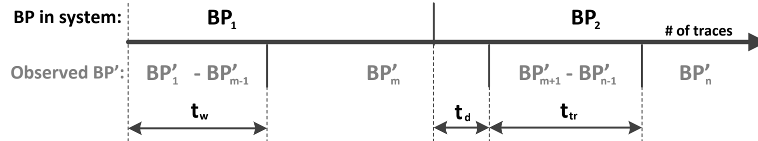

Fig. 6. Measures for Detection of BP Change in System

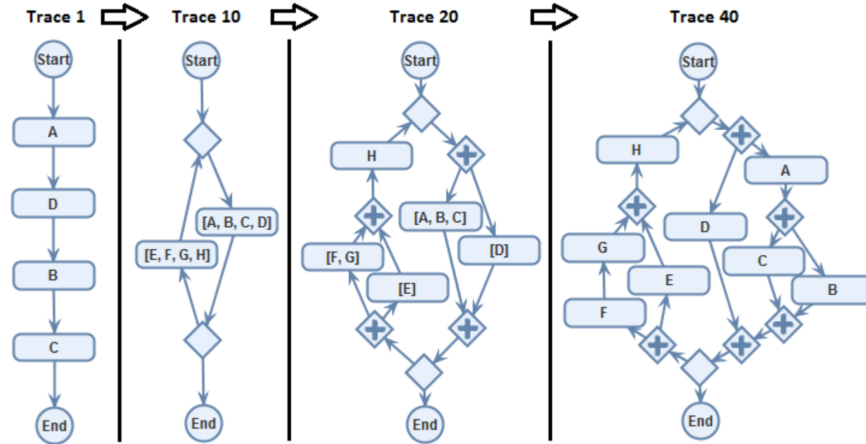

Fig. 7. The Evolution of the Discovered BP Model During the Warm-up Phase

the missing sub-footprints. In the next scheduled run of the footprint interpretation, the Parallel construct of a, b, c, and d is discovered but again the analysis can not advance since a sub-footprint for the individual activity subsets has not been created yet. Activities e, f, g, and h seem to have appeared only in exactly this sequence until trace 20. Skipping one of the interpretation steps, we can see that at trace 40 the complete process has been mined, i.e. $t_w = 40$.

In Figure 8 the development of t_w for different $m \in \{1, 2, 3, 6, 10\}$ and $t_{if} \in \{0.001, 0.005, 0.01, 0.03\}$ is depicted. The warm-up phase seems generally very short and not strongly influenced by t_{if}. For $m = 10$ the warm-up phase cannot be any shorter because the example process consists of a block-depth of 3: Parallel-in-Parallel-in-Loop, i.e. 3 subsequent requests for sub-footprints have to be made. This is an indicator that the modification effort to shorten the warm-up phase had a positive effect. A small decrease of t_w can be noticed when increasing the *trace influence factor* t_{if} for small m, e.g. $m \in \{1, 2, 3\}$.

In a second test we applied a change to the business process in the monitored system and are interested in the behaviour of the DCCM as well as in the change detection t_d and the change transition period t_{tr}. Figure 9 shows the evolution of the discovered BP model with *trace influence factor* $t_{if} = 0.01$ and *interpretation frequency* $m = 10$. The change applied is the move of activity a from the position before the inner Parallel construct to the position behind it (see traces 5750 and 6310). The change was applied after 5753 traces. The footprint interpretation

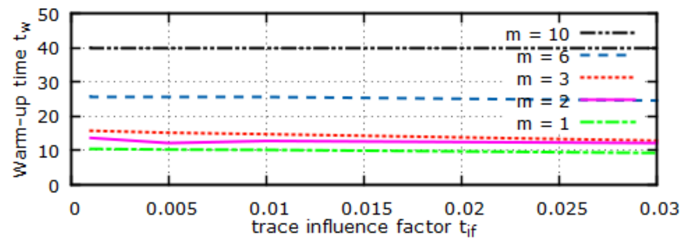

Fig. 8. The Warm-up Time in Relation to the Trace Influence Factor

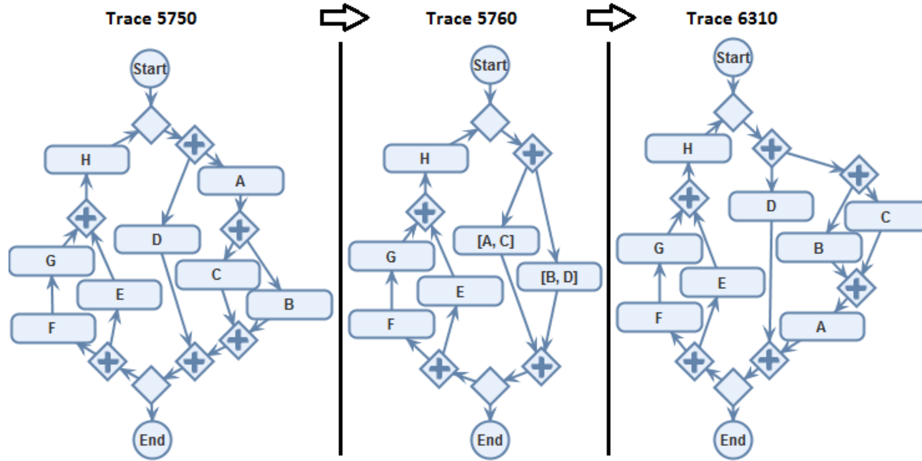

Fig. 9. The Evolution of the Discovered BP Model During a Change (Move of A)

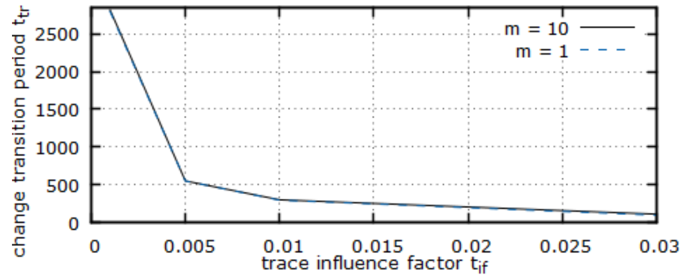

Fig. 10. The Change Transition Period in Relation to the Trace Influence Factor

detects at the first chance to discover the change (trace 5760) a concept drift and finds via competition the best fitting construct: Parallel of a, c and b, d. The change detection t_d seemed to be independent from m and t_{if} and was in all cases immediately recognized[7]. In Figure 10 the development of t_{tr} for different $m \in \{1, 10\}$ and $t_{if} \in \{0.001, 0.005, 0.01, 0.03\}$ is shown. The change transition period t_{tr} was strongly influenced by t_{if}. If the value was very small ($t_{if} = 0.001$) a change took up to almost 5000 traces in order to be reflected correctly in the discovered BP model. On the other hand if the trace influence factor is chosen too high, e.g. $t_{if} = 0.05$, not all variations of the process are included in the dynamic footprint which results in frequently changing/alternating discovered BP models. This is more likely to occur in large business processes containing rarely executed but still relevant behaviour.

Additionally, first performance tests have been carried out for large artificially produced processes (without change). For a randomly created and strongly nested process consisting of 100 activities the throughput of the footprint update was close to $100,000$ events per second and the footprint interpretation successfully discovered the process in a matter of seconds. Although not tested yet in a

[7] Note, that other changes like deletion of an activity will take longer to recognise, since their existence still "linger" in the footprints "memory" for some time.

real-life setting, the shown results indicate that the DCCM is very suitable for discovering and monitoring large enterprise processes.

7 Conclusion and Future Work

In this paper we suggested modifications for the Constructs Competition Miner to enable Scalable Dynamic Process Discovery as proposed in [11]. The CCM is a process discovery algorithm that follows a divide-and-conquer approach to directly mine a block-structured process model which consists of common BP-domain constructs and represents the main behaviour of the process. This is achieved by calculating global relations between activities and letting the different supported constructs compete with each other for the most suitable solution from top to bottom using "soft" constraints and behaviour approximations. The CCM was designed to deal with noise and not-supported behaviour. To apply the CCM in a real-time environment it was split up into two separate parts, executed on different occasions: (1) the *footprint update* which is called for every occurring event and updates the dynamic footprint(s) and (2) the footprint interpretation which derives the BP model from the dynamic footprint through applying a modified top-down competition approach of the original CCM algorithm. The modifications on the CCM were mostly motivated by the scalability requirement of SDPD and successfully implemented which is shown by the performance results in the evaluation section. It was furthermore shown that changes in the monitored process are almost instantly detected.

The presented approach of Dynamic CCM (DCCM) is driven by the requirements of real life industrial use cases provided by business partners within the EU funded project TIMBUS. During the evaluation in the context of the use-cases it became apparent that this concept still has a number of limitations which are considered to be future work: (1) Changes in the state of the business process are usually detected almost immediately but it may take a long time until the new state of the system is reflected appropriately in the extracted business process model. This behaviour originates from the fact that the footprint and the interpreted business process are in a sort of intermediate state for a while until the influence of the old version of the business process has disappeared. Furthermore, the trace influence factor t_{if} is a pre-specified value but in reality it is dependent on how many traces we need to regard to represent all the "behaviour" of the model[8]. This in turn is strongly dependent on the amount of activities in the model, since more activities usually mean more control-flow behaviour. A possible future modification could be to have the influence factor dynamically adapt, i.e. similar to the self-adapting ageing proposed in [3]. (2) If no sub-footprint is available for a set of activities, the footprint interpreter does not further analyse this set. Through approximations or the use of the direct neighbours relation at least a "close enough" control-flow for the subset could be retrieved. (3) The discovery of the state of a business process should also comprise information of other perspectives than the control-flow, e.g. resource and performance.

[8] if t_{if} is set too high normal behaviour unintentionally becomes exceptional behaviour

References

1. von Ammon, R., Ertlmaier, T., Etzion, O., Kofman, A., Paulus, T.: Integrating Complex Events for Collaborating and Dynamically Changing Business Processes. In: ICSOC/ServiceWave 2009 Workshops. LNCS, pp. 370–384. Springer, 2010
2. Buijs, J., Van Dongen, B., Van Der Aalst, W.: A genetic algorithm for discovering process trees. In: Evolutionary Computation (CEC). pp. 1-8, IEEE, 2012
3. Burattin, A., Sperduti, A., Van Der Aalst, W.: Heuristics Miners for Streaming Event Data. In: CoRR abs/1212.6383, 2012
4. Ko, Ryan K. L.: A computer scientist's introductory guide to business process management (BPM), In: Crossroads Journal, ACM, 2009
5. Leemans, S., Fahland, D., Van Der Aalst, W.: Discovering Block-Structured Process Models from Event Logs - A Constructive Approach. In: Application and Theory of Petri Nets and Concurrency, LNCS, pp. 311–329, Springer, 2013
6. Leemans, S., Fahland, D., Van Der Aalst, W.: Discovering Block-Structured Process Models from Event Logs Containing Infrequent Behaviour, In: Business Process Management Workshops 2013, LNBIP, pp. 66–78, Springer, 2013
7. Luckham, D.: The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems. Addison-Wesley Professional, Reading, 2002
8. Maggi, F. M., Burattin, A., Cimitile, M., Sperduti, A.: Online Process Discovery to Detect Concept Drifts in LTL-Based Declarative Process Models, OTM 2013, LNCS, pp. 94–111, Springer, 2013
9. OMG Inc: Business Process Model and Notation (BPMN) Specification 2.0, `http://www.omg.org/spec/BPMN/2.0/PDF`. formal/2011-01-03, 2011
10. Redlich, D., Molka, T., Rashid, A., Blair, G., Gilani, W.: Constructs Competition Miner: Process Control-flow Discovery of BP-domain Constructs. In: 12th Int. Conf. on Business Process Management, LNCS, pp. 134–150, Springer, 2014
11. Redlich, D., Gilani, W., Molka, T., Drobek, M., Rashid, A., Blair, G.: Introducing a Framework for Scalable Dynamic Process Discovery. In: 4th Enterprise Engineering Working Conference (EEWC), LNBIP 174, pp. 151–166. Springer, 2014
12. Redlich, D., Blair, G., Rashid, A., Molka, T., Gilani, W.: Research Challenges for Business Process Models at Run-time. In: LNCS State-of-the-Art Survey Volume on Models@run.time, 2014
13. Van Der Aalst, W., Ter Hofstede, A.: YAWL: Yet Another Workflow Language, 2003
14. Van Der Aalst, W., Weijters, A., Maruster, L.: Workflow Mining: Discovering Process Models from Event Logs. IEEE Transactions on Knowledge and Data Engineering. 16(9):1128-1142, 2004
15. Van Der Aalst, W., Van Dongen, B.: *ProM : The Process Mining Toolkit.* Industrial Engineering. 489: 1-4, 2009
16. Van Der Aalst et al., *Process Mining Manifesto.* BPM 2011 Int. Workshops, 2011
17. Van Der Aalst, W.: Process Mining - Discovery, Conformance and Enhancement of Business Processes, Springer, 2011
18. Van Der Aalst, W., Adriansyah, A., Van Dongen, B.: Replaying history on process models for conformance checking and performance analysis. WIREs Data Mining and Knowledge Discovery, 2(2), 182-192, 2012
19. Weijters, A., Van Der Aalst, W., Alves de Medeiros, A.: Process Mining with the Heuristics Miner-algorithm. BETA Working Paper Series, WP 166, Eindhoven University of Technology, 2006.

Project partially funded by the European Commission under the 7th Framework Programme for research and technological development and demonstration activities under grant agreement 269940, TIMBUS project (http://timbusproject.net/).