# COnfECt: An Approach to Learn Models of Component-based Systems

Sébastien Salva, Elliott Blot

# COnfECt: An Approach to Learn Models of Component-based Systems.

Sébastien Salva[1] and Elliott Blot[1]
[1]*LIMOS - UMR CNRS 6158, Clermont Auvergne University, France*
*sebastien.salva@uca.fr, eblot@isima.fr*

Keywords: Model learning, Passive learning, Component-based systems, Callable-EFSM.

Abstract: This paper addresses the problem of learning models of component-based systems. We focus on model learning approaches that generate state diagram models of software or systems. We present COnfECt, a method that supplements passive model learning approaches to generate models of component-based systems seen as black-boxes. We define the behaviours of components that call each other with Callable Extended FSMs (CEFSM). COnfECt tries to detect component behaviours from execution traces and generates systems of CEFSMs. To reach that purpose, COnfECt is based on the notions of trace analysis, event correlation, model similarity and data clustering. We describe the two main steps of COnfECt in the paper and show an example of integration with the passive model learning approach Gk-tail.

## 1 INTRODUCTION

Delivering high quality software to end-users has become a high priority in the software industry. To help develop high quality products, the software engineering field suggests to use models, which can serve as documentation, for verification or testing. But models are often written by hand, and such a task is difficult and error-prone, even for experts. To make this task easier, model learning approaches have proven to be valuable for recovering the model of a system. In this paper, we consider one specific type of formal models, namely state machines, which are crucial for describing system behaviours. In a nutshell, model learning approaches infer a behavioural formal model of a system seen as a black-box, either by interacting with it (active approaches), e.g., with test cases, or by analysing a set of execution traces resulting from the monitoring of the system (passive approaches).

Although it is possible to infer models from some realistic systems, several points require further investigation before entering in an industrial phase. Among them, we observed that the current approaches consider a black-box system as a whole, which takes input events from an external environment and produces output events. Yet, most of the systems being currently developed are made up of reusable features or components that interact together. The modelling of these components and of their compositions would bring a better readability and understanding of the functioning of the system under learning.

We focus on this open problem in this paper and propose a method called COnfECt (COrrelate Extract Compose) for learning a system of CEFSMs (Callable Extended FSMs), which describes a component-based system. COnfECt aims at completing the passive model learning approaches, which take execution traces as inputs. The fundamental idea considered in COnfECt is that a component of a system can be identified from the others by its behaviour. COnfECt analyses execution traces, detects sequences of distinctive behaviours, extracts them into new trace sets from which CEFSMs are generated. To do this, COnfECt uses the notions of event correlation, similarities of models and data clustering. More precisely, the contributions of our work are:

- the definitions of the CEFSM model and of a system of CEFSMs allowing to express the behaviours of components calling each other;
- COnfECt, a method supplementing the passive model learning approaches that generate EFSMs (Extended FSMs). COnfECt consists of two steps called *Trace Analysis & Extraction* and *CEFSM Synchronisation*. The first step splits traces into event sequences that are analysed to build new trace sets and to prepare the CEFSM synchronisation. The second step proposes three strategies of CEFSM synchronisation, which help manage the over-generalisation problem, i.e., the problem of generating models expressing more behaviours than those given in the initial trace set. This step returns a system of CEFSMs. We briefly show

how COnfECt can be combined with the passive approach Gk-tail (Lorenzoli et al., 2008). We call the new approach Ck-tail. We show how to arrange the steps of Gk-tail and COnfECt to generate a system of CEFSMs from the traces of a black-box system.

The remainder of the paper is organised as follows: Section 2 presents some related work. Section 3 provides some definitions about the CEFSM model. The COnfECt method is presented in Section 4. We finally conclude and give some perspectives for future work in Section 6.

## 2 RELATED WORK

We consider in this paper that model learning is defined as *a set of methods that infer a specification by gathering and analysing system executions and concisely summarising the frequent interaction patterns as state machines that capture the system behaviour* (Ammons et al., 2002). Models can be generated from different kinds of data samples such as affirmative/negative answers (Angluin, 1987), execution traces (Krka et al., 2010; Antunes et al., 2011; Durand and Salva, 2015), or source code (Pradel and Gross, 2009). Two kinds of approaches emerge from the literature: active and passive model learning methods.

Active learning approaches repeatedly query systems or humans to collect positive or negative observations, which are studied to build models. Many existing active techniques have been conceived upon the $\mathcal{L}^*$ algorithm (Angluin, 1987). Active learning cannot be applied on all systems though. For instance, uncontrollable systems cannot be queried easily, or the use of active testing techniques may lead a system to abnormal functioning because it has to be reset many times. The second category includes the techniques that passively generate models from a given set of samples, e.g., a set of execution traces. These techniques are said to be passive since there is no interaction with the system to model. Models are often constructed with these passive approaches by representing sample sets with automata whose equivalent states are merged. The state equivalence is usually defined by means of event sequence abstractions or state based abstractions. With event sequence abstractions, the abstraction level of the models is raised by merging the states having the same event sequences. This process stands on two main algorithms: kTail (Biermann and Feldman, 1972) and kBehavior (Mariani and Pezze, 2007). Both algorithms were enhanced to support events combined with data values (Lorenzoli et al., 2008; Mariani and Pastore, 2008). In particular, kTail has been enhanced with Gk-tail to generate EFSMs (Lorenzoli et al., 2008; Mariani et al., 2017). The approaches that use state-based abstraction, e.g., (Meinke and Sindhu, 2011), adopted the generation of state-based invariants to define equivalence classes of states that are combined together to form final models. The *Daikon* tool (Ernst et al., 1999) were originally proposed to infer invariants composed of data values and variables found in execution traces.

None of the current model learning approaches support the generation of models describing the behaviours of components of a system under learning. This work tackles this research problem and proposes an original method for inferring models as systems of CEFSMs. Our main contribution is the detection of component behaviours in an execution trace set by means of trace analysis, event correlation, model similarity and data clustering.

## 3 CALLABLE EXTENDED FINITE STATE MACHINE

We propose in this section a model of components-based systems called Callable Extended Finite State Machine (CEFSM), which is a specialised FSM including parameters and guards restricting the firing of transitions. Parameters and symbols are combined together to constitute events. A CEFSM describes the behaviours of a component, which interacts with the external environment, accepting input valued events (i.e. symbols associated with parameter assignments) and producing output valued events. In addition, the CEFSM model is equipped by a special internal (unobservable) event denoted *call(CEFSM)* to trigger the execution of another CEFSM. This event means that the current CEFSM is being paused while another CEFSM $C_2$ starts its execution at its initial state. Once $C_2$ reaches a final state, the calling CEFSM resumes its execution after the event *call(CEFSM)*. We do not consider in this paper that a component is able to provide results to another one.

Before giving the CEFSM definition, we assume that there exist a finite set of symbols $\mathcal{E}$, a domain of values denoted $D$ and a variable set $X$ taking values in $D$. The assignment of variables in $Y \subseteq X$ to elements of $D$ is denoted with a mapping $\alpha : Y \rightarrow D$. We denote $D_Y$ the assignment set over $Y$. For instance, $\alpha = \{x := 1, y := 3\}$ is a variable assignment of $D_{\{x,y\}}$. $\alpha(x) = \{x := 1\}$ is the variable assignment related to the variable $x$.

**Definition 1 (CEFSM)** *A Callable Extended Finite State Machine (CEFSM) is a 5-tuple* $\langle S, s0, \Sigma, P, T \rangle$

*where :*

- *$S$ is a finite set of states, $S_F \subseteq S$ is the non-empty set of final states, $s0$ is the initial state,*
- *$\Sigma \subseteq \mathcal{E} = \Sigma_I \cup \Sigma_O \cup \{call\}$ is the finite set of symbols, with $\Sigma_I$ the set of input symbols, $\Sigma_O$ the set of output symbols and call an internal action,*
- *$P$ is a finite set of parameters, which can be assigned to values of $D_P$,*
- *$T$ is a finite set of transitions. A transition $(s_1, e(p), G, s_2)$ is a 4-tuple also denoted $s_1 \xrightarrow{e(p),G} s_2$ where :*
  - *$s_1, s_2 \in S$ are the source and destination states,*
  - *$e(p)$ is an event with $e \in \Sigma$ and $p = \langle p_1, ..., p_k \rangle$ a finite tuple of parameters in $P^k (k \in \mathbb{N})$,*
  - *$G : D_P \rightarrow \{true, false\}$ is a guard that restricts the firing of the transition.*

A component-based system is often made up of several components. This is why we talk about systems of CEFSMs in the remainder of the paper. A system of CEFSMs *SC* consists of a CEFSM set *C* and of a set of initial states *S*0, which also are the initial states of some CEFSMs of *C*. *SC* is assumed to include at least one CEFSM that calls others CEFSMs and whose initial state is in *S*0:

**Definition 2 (System of CEFSMs)** *A System of CEFSMs is a 2-tuple $\langle C, S0 \rangle$ where :*

- *$C$ is a non-empty and finite set of CEFSMs,*
- *$S0$ is a non-empty set of initial states such that $\forall s \in S0, \exists C_1 = \langle S, s0, \Sigma, P, T \rangle \in C : s = s0$.*

We also say that a CEFSM $C_1$ is callable-complete over a system of CEFSMs *SC*, iff the CEFSMs of *SC* can be called from any state of $C_1$:

**Definition 3 (Callable-complete CEFSM)** *Let $SC = \langle C, S0 \rangle$ be a system of CEFSMs. A CEFSM $C_1 = \langle S, s0, \Sigma, P, T \rangle$ is said callable-complete over SC iff $\forall s \in S, \exists s_2 \in S : s \xrightarrow{call(EFSM),G} s_2$, with $G : \bigvee_{C_2 \in C \setminus \{C_1\}} CEFSM = C_2$*

A trace is a finite sequence of observable valued events in $(\mathcal{E} \times D_X)^*$. We use $\varepsilon$ to denote the empty sequence.

# 4 THE CONFECT APPROACH

COnfECt (COrrelate Extract Compose) is an approach for learning a system of CEFSMs from the execution traces of a black-box system. COnfECt analyses traces and tries to detect components and theirs respective behaviours, which are modelled with CEFSMs. COnfECt aims to complement the passive model learning methods and requires a trace set to analyse them and identify the components of a black-box system. And the more traces, the more correct the component detection will be.

The system under learning SUL can be indeterministic, uncontrollable (it may provide output valued events without querying it with a valued input event) or can have cycles among its internal states. However, SUL and its trace set denoted *Traces* have to obey certain restrictions to avoid the interleaving of events. We consider that SUL is constituted of components whose observable behaviours are not carried out in parallel. One component is executed at a time; a caller component is being paused until the callee terminates its execution. Furthermore, we consider having a set *Traces* composed of traces collected from SUL in a synchronous manner (traces are collected by means of a synchronous environment with synchronous communications). Traces can be collected by means of monitoring tools or extracted from log files. Furthermore, we do not focus this work on the trace formatting, hence, we assume having a mapper (Aarts et al., 2010) performing abstraction and returning traces as sequences of valued events of the form $e(p_1 := d_1, \ldots, p_k := d_k)$ where $p_1 := d_1, \ldots, p_k := d_k$ are parameter assignments.

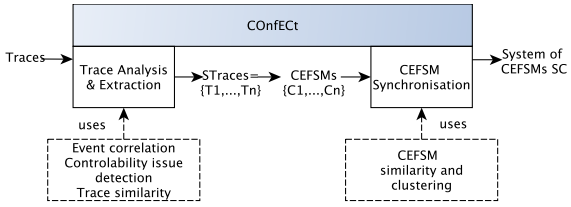


Figure 1: The COnfECt approach overview

As depicted in Figure 1, COnfECt is composed of two main stages called *Trace Analysis & Extraction* and *CEFSM Synchronisation*. The former tries to detect components in the traces of *Traces* and segments them into a set of trace sets called *STraces*. The second stage proposes three CEFSM synchronisation strategies and provides a system of CEFSMs *SC*. Theses stages are presented below. We believe they can be interleaved with the steps of several passive model learning techniques, e.g., (Mariani and Pastore, 2008; Lorenzoli et al., 2008).

## 4.1 Trace Analysis & Extraction

This stage tries to identify components in the traces of *Traces* by means of Algorithm 1. This algorithm is based on three notions implemented by three procedures. It analyses every trace of *Traces* with *Inspect*, it segments them and builds the new trace sets $T_1, \ldots, T_n$ with *Extract*. Finally, it analyses the first trace set $T_1$ to detect other components with *Separate*. The algorithm returns the set *STraces*, which is itself composed of trace sets. Each will give birth to a CEFSM.

**Algorithm 1:** Inspect&Extract Algorithm

**input :** $Traces = \{\sigma_1, \ldots, \sigma_m\}$
**output:** $STraces = \{T_1, \ldots, T_n\}$
1 $T_1 = \{\}$;
2 $STraces = \{T_1\}$;
3 **foreach** $\sigma \in Traces$ **do**
4 $\quad \sigma'_1\sigma'_2 \ldots \sigma'_k$=Inspect($\sigma$);
5 $\quad$ *STraces*=Extract($\sigma'_1\sigma'_2 \ldots \sigma'_k, T_1, STraces$);
6 *STraces*=Separate($T_1$,*STraces*);
7 return *STraces*;

### 4.1.1 Trace analysis

We assume that a component can be identified by its behaviour, which is materialised by valued events composed of symbols and data. We also observed in many systems, in particular in embedded devices, that the observation of controllability issues, i.e., observing output events without giving any input event before, is often the result of a component interacting with the external environment.

From these observations, we firstly analyse traces by means of a Correlation coefficient. This coefficient aims to evaluate the correlation of successive valued events, in other words, their links or relations. We define the Correlation coefficient between two valued events by means of a utility function, which involves a weighting process for representing user priorities and preferences, here towards some correlation factors. We have chosen the technique *Simple Additive Weighting* (SAW) (Yoon and Hwang, 1995), which allows the interpretation of these preferences with weights:

**Definition 4 (Correlation coefficient)** *Let* $e_1(\alpha_1)$, $e_2(\alpha_2)$ *be two valued events of* $(\mathcal{E} \times D_X)$, *and* $f_1(e_1(\alpha_1), e_2(\alpha_2)), \ldots f_k(e_1(\alpha_1), e_2(\alpha_2))$ *be correlation factors.*
$Corr(e_1(\alpha_1), e_2(\alpha_2))$ *is a utility function, defined as:* $0 \leq Corr(e_1(\alpha_1), e_2(\alpha_2)) = \sum_{i=1}^{k} f_i(e_1(\alpha_1), e_2(\alpha_2)).w_i \leq 1$ *with* $w_i \in \mathcal{R}_0^+$ *and* $\sum_{i=1}^{k} w_i = 1$.

The factors must give a value between 0 and 1. They can have a general form or be established with regard to the system context and addressed by an expert. We give below two general factor examples:

- $f_1(e_1(\alpha_1), e_2(\alpha_2)) = \frac{\text{freq}(e_1e_2)}{\text{freq}(e_1)} + \frac{\text{freq}(e_1e_2)}{\text{freq}(e_2)}$ with $\text{freq}(e_1e_2)$ the frequency of having the two symbols one after the other in *Traces* and $\text{freq}(e)$ the frequency of having the symbol $e$. This factor, used in text mining, computes the frequency of the term $e_1e_2$ in *Traces* over $e_1$ and over $e_2$ to avoid the bias of getting a low factor when $e_1$ is greatly encountered (resp. $e_2$);
- $f_2(e_1(\alpha_1), e_2(\alpha_2)) = |param(\alpha_1) \cap param(\alpha_2)| / min(|param(\alpha_1)|, |param(\alpha_2)|)$ with $param(\alpha) = \{p \mid (p := v) \in \alpha\}$ is the overlap of the shared parameters between two valued events $e_1(\alpha_1), e_2(\alpha_2)$. We have chosen the Overlap coefficient because it is more suited for comparing sets of different sizes. We recall that the overlap of two sets $X$ and $Y$ is defined by $|X \cap Y| / min(|X|, |Y|)$.

From this Correlation coefficient, we define two relations to express what a strong and weak event correlations are. Unfortunately, experts in data mining often claim that this depends on the considered context. This is why we use two thresholds X and Y in the following. Both are factors between 0 and 1, which need to be appraised, for instance after some iterative attempts.

**Definition 5 (Strong and Week event Correlations)** *Let* $e_1(\alpha_1)$, $e_2(\alpha_2)$ *be two valued events of* $(\mathcal{E} \times D_X)$ *such that* $e_1 \neq call$ *and* $e_2 \neq call$.
$e_1(\alpha_1)$ *weak-corr* $e_2(\alpha_2) \Leftrightarrow_{def} Corr(e_1(\alpha_1), e_2(\alpha_2)) < X$.
$e_1(\alpha_1)$ *strong-corr* $e_2(\alpha_2) \Leftrightarrow_{def} Corr(e_1(\alpha_1), e_2(\alpha_2)) > Y$.

These relations are specialised on two valued events. We complete them to formalise the strong correlation of valued event sequences. We say that strong-corr($\sigma_1$) holds when $\sigma_1$ has successive valued events that strongly correlate. We are now ready to identify the behaviours of components. We define the relation $\sigma_1$ *mismatch* $\sigma_2$, which holds when the last event of $\sigma_1$ weakly correlates with the first one of $\sigma_2$ or when a controllability issue is observed between $\sigma_1$ and $\sigma_2$:

**Definition 6 (Valued event sequence correlation)** *strong-corr*($\sigma$) *iff*

$$\begin{cases} \sigma = e(\alpha) \in (\mathcal{E} \times D_X), \\ \sigma = e_1(\alpha_1)\dots e_k(\alpha_k)(k>1), \forall (1 \le i < k): \\ e_i(\alpha_i) \textit{ strong-corr } e_{i+1}(\alpha_{i+1}) \end{cases}$$

*Let* $\sigma_1 = e_1(\alpha_1)\dots e_k(\alpha_k)$, $\sigma_2 = e'_1(\alpha'_1)\dots e'_l(\alpha'_l) \in (\mathcal{E} \times D_X)^*$. $\sigma_1$ *mismatch* $\sigma_2$ *iff*

$$\begin{cases} \sigma_2 = \varepsilon, \\ e_k(\alpha_k) \textit{ weak-corr } e'_1(\alpha'_1), \\ e'_1 \textit{ is an output symbol} \wedge e_k \textit{ is an output symbol} \end{cases}$$

The trace analysis is performed with the procedure Inspect given in Algorithm 2, which covers every trace $\sigma$ of *Traces* and tries to segment $\sigma$ into sub-sequences such that each sub-sequence has a strong correlation and has a weak correlation with the next sub-sequence. We consider that these sub-sequences result from the execution of components.

### 4.1.2 Trace extraction

Every trace $\sigma \in$ *Traces* was segmented into $\sigma'_1\sigma'_2\dots\sigma'_k$ by means of the relations strong-corr and mismatch. Every time $\sigma'_i$ mismatch $\sigma'_{i+1}$ holds between two successive sub-sequences, we consider having the call of other components by the current one because both sub-sequences exhibit different behaviours. $\sigma$ is modified by the procedure Extract to express these calls.

The procedure Extract$(\sigma, T, STraces)$, given in Algorithm 2, takes the trace $\sigma = \sigma_1\dots\sigma_k$, transforms it and then adds the new trace into the trace set $T$. For a sub-sequence $\sigma_{id}$ of the trace $\sigma = \sigma_1\dots\sigma_k$, the procedure *Extract* tries to find another sub-sequence $\sigma_i$ such that strong-corr$(\sigma_{id}\sigma_i)$ holds (lines 10,11). The sequence $\sigma_{id+1}\dots\sigma_{i-1}$ or $\sigma_{id+1}\dots\sigma_k$ (when $\sigma_i$ is not found) exposes the behaviour of other components that are called by the current one. If this sequence is itself composed of more than two sub-sequences, then the procedure *Extract* is recursively called (lines 13,14). Otherwise, the sequence is added to a new trace set $T_n$. In $\sigma$, the sequence $\sigma_{id+1}\dots\sigma_{i-1}$ (or $\sigma_{id+1}\dots\sigma_k$) is removed and replaced by the valued event $call(CEFSM := C_n)$ (lines 12,19). Once, the sequence $\sigma$ is covered by the procedure Extract, it is placed into the set $T$.

Let us consider the example of Figure 2, which illustrates the transformation of a trace $\sigma$. This trace was initially segmented into 6 sub-sequences. A) We start with $\sigma_1$. We suppose the first sequence that is strongly correlated with $\sigma_1$ is $\sigma_5$. $\sigma$ is transformed into $\sigma_1 call(CEFSM := C_2)\sigma_5\sigma_6$. Recursively, Extract$(\sigma_2\sigma_3\sigma_4, T_2)$ is called to split $\sigma' = \sigma_2\sigma_3\sigma_4$. B) We suppose $\sigma_2\sigma_4$ strongly correlates, hence, $\sigma'$ is modified and is equal to $\sigma' = \sigma_2 call(CEFSM := C_3)\sigma_4$. The sequence $\sigma_3$ is a new trace of the new set $T_3$. As $\sigma'$ is completely covered,

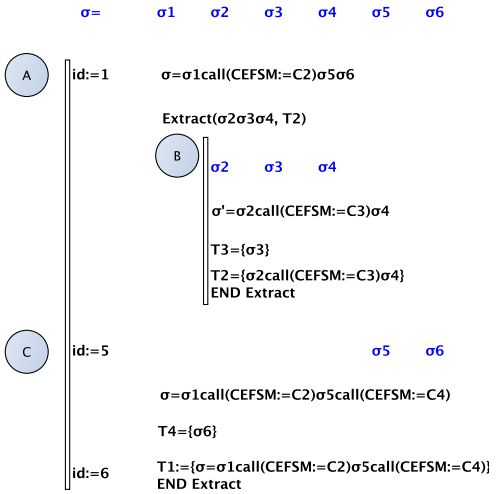


Figure 2: Sequence extraction example

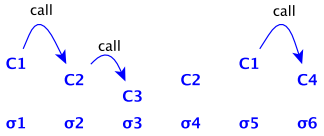


Figure 3: Component call example

$\sigma'$ is added to the new trace set $T_2$. C) We go back to the trace $\sigma$ at the sub-sequence $\sigma_5$. As there is no more sub-sequence that strongly correlates with $\sigma_5$, the end of the sequence $\sigma$, i.e., $\sigma_6$, is extracted and placed into the new trace set $T_4$. The trace $\sigma$ is now equals to $\sigma_1 call(CEFSM := C_2)\sigma_5 call(CEFSM := C_4)$. This trace is placed into the trace set $T_1$. At the end of this process, we have recovered the hierarchical component call depicted in Figure 3. And we get four trace sets.

When the procedure Extract terminates, Algorithm 1 yields the set $STraces = \{T_1, T_2, \dots, T_n\}$ with $T_2, \dots, T_n$ some sets including one trace and $T_1$ a set of modified traces, originating from *Traces*. As we do not suppose that *Traces* expresses the behaviours of only one component, $T_1$ may include traces resulting from different components. Hence, $T_1$ needs to be analysed as well and possibly partitioned.

### 4.1.3 Trace clustering

The trace set $T_1$ is analysed with the procedure Separate, which returns an updated set *STraces*. The procedure aims at partitioning $T_1$ into trace sets exclusively composed of similar traces. We consider that similar traces exhibit a behaviour provided by the same component. We evaluate the trace similarity with regard to the symbols and parameters shared

between pairs of traces. Several general similarity coefficients are available in the literature for comparing the similarity and diversity of sets, e.g., the well-known Jaccard coefficient. We have once more chosen the Overlap coefficient because the symbol or parameter sets used by two traces may have different sizes.

**Definition 7 (Trace Similarity coefficient)** *Let* $\sigma_i$ $(i = 1,2)$ *be two traces in* $(\mathcal{E} \times D_X)^*$.
$\Sigma(\sigma_i) = \{e \mid e(\alpha)$ *is a valued event of* $\sigma_i\}$ *is the symbol set of* $\sigma_i$.
$P(\sigma_i) = \{p \mid e(\alpha)$ *is a valued event of* $\sigma_i, (p := v) \in \alpha\}$ *is the parameter set of* $\sigma_i$.
$Similarity_{Trace}(\sigma_1,\sigma_2) = Overlap(\Sigma(\sigma_1),\Sigma(\sigma_2)) + Overlap(P(\sigma_1),P(\sigma_2))/2$.

With this coefficient, the procedure Separate builds the sets of similar traces from $T_1$ by means of a clustering technique. In short, the coefficient is evaluated for every pair of traces to build a similarity matrix, which can be used by several clustering algorithms to find equivalence classes. The clustering techniques here return the clusters of similar traces $T_{11}^S, \ldots T_{1k}^S$. These sets are added into S*Traces*. The sets $T_{1i}^S$ are marked with the exponent $S$ to denote they are composed of execution traces observed from components that were not called by other components at the beginning of these executions.

## 4.2 The CEFSM Synchronisation Stage

This stage aims to organise the component synchronisation with regard to the event $call(CEFSM)$. The choice of integration of this stage within an existing model learning approach mainly depends on the steps of this approach. But it sounds natural to focus on models, here CEFSMs, for applying different synchronisation strategies. Thus, we consider that the set STraces has been lifted to a system of CEFSMs $SC = \langle C, S0\rangle$ by means of a passive learning method, e.g., (Lorenzoli et al., 2008). $C$ is composed of the CEFSM $C_i$ such that $C_i$ is derived from a trace $T_i \in STraces$. In particular, a marked set $T_j^S$ (composed of traces observed from components that were not called by other components) gives the CEFSM $C_j = \langle S_j, s0_j, \Sigma_j, P_j, T_j\rangle$ whose initial state $s0_j$ is also an initial state of the system of CEFMs $SC$ ($s0_j \in S0$).

We propose three general CEFSM synchronisation strategies in the paper, which provide systems of CEFSMs having different levels of generalisation. These strategies are implemented in Algorithm 3 and described below:

**Strict synchronisation** (Algorithm 3 lines(1,2)). We want a system of CEFSMs $SC$ in such a way that a CEFSM of $SC$ cannot repetitively call another CEFSM. The callee CEFSM must be composed of one acyclic path only (one behaviour). This strategy aims to limit the over-generalisation problem, i.e. the fact of generating models expressing more behaviours than those given in the initial trace set $Traces$. This strategy was already almost achieved by the previous stage *Trace Analysis & Extraction*. Indeed, each subsequence extracted from a trace is placed into new trace set $T_i$ and is replaced by one valued event of the form $Call(CEFSM := C_i)$. Hence, it remains to transform the trace sets of $STraces$ into CEFSMs for obtaining a system of CEFSMs organised with a strict synchronisation.

**Weak synchronisation** (Algorithm 3 lines(3-16)). This strategy aims at reducing the number of

**Algorithm 2:** Procedures Inspect, Extract and Separate

```
1  Procedure Inspect(σ) : σ'_1 σ'_2 ... σ'_k is
2      Find the no-empty sequences σ'_1 σ'_2 ... σ'_k such that:
         σ = σ'_1 σ'_2 ... σ'_k, strong-corr(σ'_i)_(1≤i≤k), (σ'_i mismatch
         σ'_{i+1})_(1≤i≤k−1);
3  Procedure Extract(σ = σ_1 σ_2 ... σ_k, T, STraces): STraces is
4      id := 1;
5      while id < k do
6          n := |STraces| + 1;
7          T_n := {};
8          STraces := STraces ∪ {T_n};
9          σ_p is the prefix of σ up to σ_id;
10         if ∃i > id: strong-corr(σ_id σ_i) then
11             σ_i is the first sequence in σ_id ... σ_k such that
                 strong-corr(σ_id σ_i);
12             σ := σ_p σ_id call(CEFSM := C_n) σ_i ... σ_k;
13             if (i − id) > 2 then
14                 Extract(σ_{id+1} ... σ_{i−1}, T_n);
15             else
16                 T_n := T_n ∪ {σ_{id+1}};
17             id := i;
18         else
19             σ := σ_p σ_id call(CEFSM := C_n);
20             if (k − id) > 1 then
21                 Extract(σ_{id+1} ... σ_k, T_n);
22             else
23                 T_n := T_n ∪ {σ_k};
24             id := k;
25     T := T ∪ {σ};
26     return STraces;
27 Procedure Separate(T, STraces): STraces is
28     ∀(σ_i, σ_j) ∈ T^2 Compute Similarity_Trace(σ_i, σ_j);
29     Build a similarity matrix;
30     Group the similar traces into clusters {T_11, ... T_1k};
31     STraces = STraces \ {T_1} ∪ {T_11^S, ..., T_1k^S};
```

components and allows repetitive component calls. The previous stage has possibly created too much trace sets, therefore the system of CEFSMs *SC* may include several similar CEFSMs modelling the functioning of the same component. The similarity notion is once more defined and evaluated by a Similarity coefficient.

**Definition 8 (CEFSM Similarity coefficient)** *Let* $C_i = \langle S_i, s0_i, \Sigma_i, P_i, T_i \rangle$ $(i = 1,2)$ *be two CEFSMs.* $Similarity_{CEFSM}(C_1, C_2) = Overlap(\Sigma_1, \Sigma_2) + Overlap(P_1, P_2)/2$.

The similar CEFSMs of *SC* are once more grouped by means of a clustering technique, which uses the Similarity coefficient. The CEFSMs of the same cluster are joined by means of a disjoint union. Furthermore, the guards of the transitions $s_1 \xrightarrow{call(CEFSM),G} s_2$ are updated accordingly so that the correct CEFSMs are being called. In addition, every transition $s_1 \xrightarrow{call(CEFSM),G} s_2$ is replaced by a self loop $(s_1, s_2) \xrightarrow{call(CEFSM),G} (s_1, s_2)$ by merging the states $s_1$ and $s_2$.

**Strong synchronisation** (Algorithm 3 lines(4-20)). This strategy provides more over-generalised models by generating callable-complete CEFSMs. It is based on the previous strategy: we join the similar CEFMSs of *SC* into bigger CEFSMs and we transform the transitions labelled by *call* as previously. In addition, we complete every state $s$ with new self-loop transitions of the form $s \xrightarrow{call(CEFSM),G} s$ so that all the CEFSMs become callable-complete over the system of CEFSMs *SC*. This strategy seems particularly interesting for modelling component-based systems having independent components that are started any time.

We studied the integration of COnfECt with several passive learning approaches. We have implemented a combination of the approach with kTail to generate Labelled Transition Systems (LTS). The source code as well as examples are available in (Salva et al., 2018).

We are also studying the integration of COnfECt with Gk-tail to generate systems of CEFSMs. Figure 4 illustrates how the COnfECt and Gk-tail steps can be organised. The COnfECt steps are given with white boxes. We call the resulting approach Ck-tail. Step 2 corresponds to the first step of COnfECt. We placed it after Step 1 (trace merging) to have less trace to analyse, and before Step 3 (guard generation) to measure the event correlation on symbols and real values. The CEFSM Synchronisation step of COnfECt is the fifth step of Ck-tail. It is performed after the CEFSM tree generation, and before Step 6 (state merging) because it sounds more interesting to group the similar CEFMS and to merge their equivalent states after, as more equivalent states should be merged if we follow this step order. We illustrate this integration with a example based upon a real system (an IOT (Internet Of Things) thermostat) in (Salva et al., 2018).

**Algorithm 3:** CEFSM synchronisation strategies

```
   input : System of CEFSMs SC = ⟨C, S0⟩, strategy
   output: System of CEFSMs SC_f = ⟨C_f, S0_f⟩
1  if strategy = Strict synchronisation then
2      return SC;
3  else
4      ∀(C_i, C_j) ∈ C^2 Compute Similarity_CEFSM(C_i, C_j);
5      Build a similarity matrix;
6      Group the similar CEFSMs into clusters {Cl_1, ... Cl_k};
7      foreach cluster Cl = {C_1, ..., C_l} do
8          C_Cl:=Disjoint Union of the CEFSMs C_1, ..., C_l;
9          if s0_i ∈ S0 (1 ≤ i ≤ l) then
10             S0_f = S0_f ∪ s0_Cl;
11         C_f = C_f ∪ {C_Cl};
12     foreach C_i = ⟨S, s0, Σ, P, V, T⟩ ∈ C_f do
13         foreach s_1 --call(CEFSM),G--> s_2 ∈ T with G : CEFSM = C_m
           do
14             Find the Cluster Cl such that C_m ∈ Cl;
15             Replace G by G : CEFSM = C_Cl;
16             Merge (s_1, s_2);
17     if strategy = Strong synchronisation then
18         foreach C_i = ⟨S, s0, Σ, P, T⟩ ∈ C_f do
19             Complete the states of S with self-loop transitions
               so that C_i is callable-complete;
20     return SC_f
```

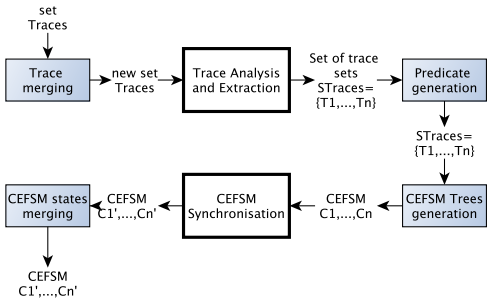


Figure 4: Ck-tail: Integration of COnfECt with Gk-tail

# 5 ACKNOWLEDGMENT

Research supported by the French Project VASOC (Auvergne-Rhne-Alpes Region) https://vasoc.limos.fr/

# 6 CONCLUSION

We have presented COnfECT, a method that complements existing passive model learning approaches to infer systems of CEFSMs from execution traces. COnfECT is able to detect component behaviours by analysing traces by means of a Correlation coefficient and a Similarity coefficient. In addition, COnfECT proposes three model synchronisation strategies, which help manage the over-generalisation of systems of CEFSMs.

In future work, we intend to carry out more evaluations of COnfECT on several kinds of systems. The main issue concerns the implementation of monitors and mappers, which are required to format traces. We also intend to tackle the raise of the abstraction level of CEFSMs. Indeed, while the trace analysis, the successive computations of the Correlation coefficient could also be used to perform event aggregation in accordance with event correlation and some CEFSM structural restrictions.

# REFERENCES


Aarts, F., Jonsson, B., and Uijen, J. (2010). Generating models of infinite-state communication protocols using regular inference with abstraction. In Petrenko, A., Simão, A., and Maldonado, J. C., editors, *Testing Software and Systems*, pages 188–204, Berlin, Heidelberg. Springer Berlin Heidelberg.

Ammons, G., Bodík, R., and Larus, J. R. (2002). Mining specifications. *SIGPLAN Not.*, 37(1):4–16.

Angluin, D. (1987). Learning regular sets from queries and counterexamples. *Information and Computation*, 75(2):87 – 106.

Antunes, J., Neves, N., and Verissimo, P. (2011). Reverse engineering of protocols from network traces. In *Reverse Engineering (WCRE), 2011 18th Working Conference on*, pages 169–178.

Biermann, A. and Feldman, J. (1972). On the synthesis of finite-state machines from samples of their behavior. *Computers, IEEE Transactions on*, C-21(6):592–597.

Durand, W. and Salva, S. (2015). Passive testing of production systems based on model inference. In *13. ACM/IEEE International Conference on Formal Methods and Models for Codesign, MEMOCODE 2015, Austin, TX, USA, September 21-23, 2015*, pages 138–147. IEEE.

Ernst, M. D., Cockrell, J., Griswold, W. G., and Notkin, D. (1999). Dynamically discovering likely program invariants to support program evolution. In *Proceedings of the 21st International Conference on Software Engineering*, ICSE '99, pages 213–224, New York, NY, USA. ACM.

Krka, I., Brun, Y., Popescu, D., Garcia, J., and Medvidovic, N. (2010). Using dynamic execution traces and program invariants to enhance behavioral model inference. In *Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering - Volume 2*, ICSE '10, pages 179–182, New York, NY, USA. ACM.

Lorenzoli, D., Mariani, L., and Pezzè, M. (2008). Automatic generation of software behavioral models. In *Proceedings of the 30th International Conference on Software Engineering*, ICSE '08, pages 501–510, New York, NY, USA. ACM.

Mariani, L. and Pastore, F. (2008). Automated identification of failure causes in system logs. In *Software Reliability Engineering, 2008. ISSRE 2008. 19th International Symposium on*, pages 117–126.

Mariani, L. and Pezze, M. (2007). Dynamic detection of cots component incompatibility. *IEEE Software*, 24(5):76–85.

Mariani, L., Pezz, M., and Santoro, M. (2017). Gk-tail+ an efficient approach to learn software models. *IEEE Transactions on Software Engineering*, 43(8):715–738.

Meinke, K. and Sindhu, M. (2011). Incremental learning-based testing for reactive systems. In Gogolla, M. and Wolff, B., editors, *Tests and Proofs*, volume 6706 of *Lecture Notes in Computer Science*, pages 134–151. Springer Berlin Heidelberg.

Pradel, M. and Gross, T. R. (2009). Automatic generation of object usage specifications from large method traces. In *Proceedings of the 2009 IEEE/ACM International Conference on Automated Software Engineering*, ASE '09, pages 371–382, Washington, DC, USA. IEEE Computer Society.

Salva, S., Blot, E., and Laurençot, P. (2018). Model Learning of Component-based Systems. Limos research report. http://sebastien.salva.free.fr/useruploads/files/SBL18a.pdf.

Yoon, K. P. and Hwang, C.-L. (1995). Multiple attribute decision making: An introduction (quantitative applications in the social sciences).