Periodic Task Mining in Embedded System Traces

Oleg Iegorov
University of Waterloo, Canada
oiegorov@uwaterloo.ca

Reinier Torres
University of Waterloo, Canada
rtorresl@uwaterloo.ca

Sebastian Fischmeister
University of Waterloo, Canada
sfischme@uwaterloo.ca

***Abstract*—Modern systems are growing in complexity beyond deep comprehension of developers. Increasing difficulties of keeping software projects on schedule and increasing recall rates are symptoms of this development. Consequently, developers need new methods and tools to build embedded systems, such as tools that dynamically analyze systems and recover comprehensible specifications of particular aspects.**

In this paper, we address the problem of discovering temporal behavior of real-time systems by mining periodic task sets and their temporal characteristics from system execution traces. We leverage the periodic nature of real-time systems to achieve this goal in an automatic way.

We propose PeTaMi (PEriodic TAsk MIner) – a novel approach and a tool to mine periodic tasks along with information on their periods and response time profiles from execution traces of real-time systems. PeTaMi embraces an important observation we make about operation of periodic tasks: their individual jobs are usually followed by intervals of task inactivity of a considerable duration. We evaluated PeTaMi on two case studies (unmanned aerial vehicle and a commercial car in operation) using traces containing tens of thousands of recorded execution events.

I. Introduction

In the past few decades real-time embedded systems (RTES) have seen an exponential growth in their source lines of code, as for example RTES found in cars [14] and avionics [43]. The complexity of these systems is also growing, which combined with large number of source lines of code increases the number of software related malfunctions [8]. For example, the automotive industry shows an increasing trend in car recalls due to ECU issues [25]. Multiprocessor systems, hardware acceleration, and real time operating systems (RTOS) provide support for more complex software. However, these new platforms are also hindering the real-time analysis of embedded software. This reality dictates the need to provide software developers with tools to help them understand runtime behavior of these complex systems.

Software reverse engineering aims at providing support for the comprehension of complex systems by creating suitable representations of the system in a higher level of abstraction [18]. Reverse engineering of desktop and enterprise software systems has reached maturity and has been successfully applied and commercialized for numerous software engineering problems [11]. On the other hand, surprisingly little amount of research has been done on reverse engineering of embedded software, as confirmed in [28]. Indeed, traditional reverse engineering tools are mostly based on structure recovery, and do not consider the timeliness aspect inherent to RTES [22].

Mining task sets and their temporal properties from traces collected during system operation could be an essential tool for reverse engineering of modern RTES. Indeed, most RTES are implemented as a set of tasks (recurrent programs) that run on one or many processors. In many cases these tasks are activated periodically to run a job and then wait until next activation. For periodic tasks, the activation period and the time to complete each job are critical to the proper operation of the system. Regardless of whether the temporal specification of a task set is known a priori, it is useful to learn these properties from the system itself.

In this work we address mining of periodic tasks, their periods and response times profiles, using passive learning. To this end, we propose PeTaMi (PEriodic TAsk MIner) - a novel approach to mine periodic tasks and their temporal specifications from execution traces produced by RTES under real operating conditions. In its first stage, PeTaMi performs binary classification of the system's task set into two subsets: one containing periodic tasks, and another one containing non-periodic tasks. The set of tasks deemed periodic is then used in the second stage of PeTaMi to compute periods and response times profiles of periodic tasks.

The key contributions of this paper include:

- A novel automatic method to classify task sets into periodic and non-periodic categories using timestamped event traces.
- A clustering-based approach to mine temporal specifications of periodic tasks, namely task's period and response time profile.
- A thorough evaluation using execution traces of two deployed real-time systems.

The rest of the paper is organized as the following: Section II presents a short review of the related work on mining system behavior of complex RTES. Section III formally introduces the problem addressed by PeTaMi, followed by an overview of our approach in Section IV. Section V explains how to pre-process execution traces before feeding them to PeTaMi. In Section VI we address the problem of identifying periodic task sets. We use run-to-completion and preemptive scheduling as "extreme type" scenarios for which PeTaMi can properly classify the system's task set. In Section VII we tackle the problem of mining periods and response time profiles from the set of periodic tasks. In Section VIII we evaluate PeTaMi firstly with synthetic data sets, and then discuss the results of

978-1-5090-5269-1/17 $31.00 © 2017 IEEE

using PeTaMi in two case studies: an unmanned aerial vehicle and a CAN communication bus protocol. Finally we discuss the obtained results in Section IX and conclude the paper in Section X.

II. Related Work

The problem of reverse engineering of complex systems has two independent aspects: extraction of a system model and representation of a system model.

A system model can be extracted either statically or dynamically. Methods that implement the static approach only rely on the source code to find erroneous or suspicious system behavior, without actually executing programs [20] [4] [32]. Methods that implement the dynamic approach, on the other hand, consider runtime system behavior, as recorded in an execution trace. The two methods to extract a system model from execution traces are active and passive learning. With active learning [13] [7], the target system is asked to execute under various stimuli (inputs), while passive learning considers only the given execution traces, and does not need to interact with the target system [35]. PeTaMi uses dynamic information with passive learning.

Methods then use the extracted information to represent a system model in an abstract formalism. There exists a vast literature on different formalisms including state machines [3] [39], Petri nets [42], various types of invariants [21] [19] and UML models [29] [2]. Some formalisms support timing, such as mining LTL expressions [33], CTL expressions [6], or hybrid system automata [37]. Unlike the formalisms mentioned above, PeTaMi models system behavior as a set of independent periodic tasks. This representation is more intuitive to software developers of real-time systems, as tasks are scheduling entities inside systems and used in many aspects of analysis of real-time systems.

Several works attempted to mine periodic tasks from execution traces. The authors in [24] mine periodic tasks to discover suspicious system behavior related to the missed deadlines. Their model of periodic tasks, however, is too simplistic and does not take into account the fact that a single task may have different response times at run time. The authors in [36] deal with situations where developers do not have knowledge about the representation of tasks in execution traces, and mine groups of events occurring periodically and in proximity of each other. However, their approach works only if the developer successfully chooses a time window whose value is a factor of the task's period.

III. Problem formulation

Consider a set of n tasks $\tau_1, \ldots, \tau_n$. Each task τ_i is a recurrent program that executes a sequence of jobs $j_{i,1}, \ldots, j_{i,m}$. We say that task τ is periodic, if it releases jobs with a fixed interval of time T and not periodic otherwise. The response time (RT) of τ is the time required to execute the task program from job's release time until completion. In the general case, RTs will vary due to the task's execution time profile and scheduling interference. The variability in response times for τ can be expressed as the task's response time profile (RTP).

Let τ be an event generator. Each event is the result of activities recorded by the operating system and saved into a trace file or streamed to some communication channel. Events can be related to the execution of task code (e.g., inter-process synchronization) or changes of the task state due to scheduling activity (e.g., preemption). Figure 1 shows the timeline of the sequence of events as well as the inter-arrival time (IAT) series for some periodic task τ. Events (square markers on the timeline) can potentially arrive at any time, and no assumptions are made about what they represent. For clarity, we have marked the jobs' start times with arrows on the shaded area.

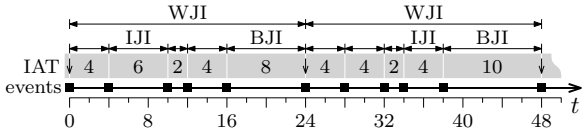


Fig. 1: Task events and IAT types.

The IATs of τ is the sequence of numbers on the shaded area of Figure 1. The values of IATs are computed as the time differences between consecutive events. There are two types of IATs: intra-job inter-arrival times (IJI) and between-jobs inter-arrival times (BJI). The whole-job inter-arrival times (WJI) denote those intervals that account for complete jobs. More precisely:

IJI: The time difference between consecutive events that occur within a job. Regarding Figure 1, this set is $\{IJI\} = \{4, 6, 2, 4, 4, 4, 2, 4, \ldots\}$.

BJI: The time difference between the last event of a job and the first event of the next job. In Figure 1 this set is $\{BJI\} = \{8, 10, \ldots\}$.

WJI: The time difference between first events of two consecutive jobs. In Figure 1, both jobs have a WJI of 24. For a periodic task $WJI \equiv T$.

Since each WJI corresponds to one and only one BJI, finding the set $\{BJI\}$ becomes the key to job discovery. In this example, there are two BJIs that clearly differentiate the jobs. However, in general case, we expect BJIs to have a large variability, thus complicating their identification.

In this paper, we argue that the set $\{BJI\}$ of periodic tasks can be discovered automatically from system traces. Therefore, the set $\{WJI\}$ can also be found. Our argument relies on a single assumption regarding temporal behavior of periodic tasks in RTES:

> *Assumption 1:* If we let a periodic task τ execute for a considerable amount of time, then the mean duration of τ's BJIs will be larger than the mean duration of τ's IJIs.

We argue that this assumption holds for industrial RTES by simulating the execution of periodic real-time tasks scheduled

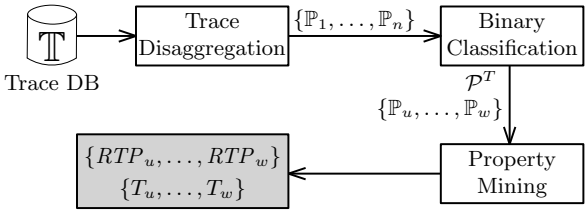


Fig. 2: PeTaMi flowchart.

with RM and EDF algorithms (see the Appendix). According to the obtained results (see Figure 13), Assumption 1 is true for RTES with up to 85% of CPU utilization. At the same time, systems with utilization above the Liu and Layland bound ($\approx 70\%$) may be considered unsafe in industrial settings (Table 1.3 on page 11 in [30]). Therefore, most systems are designed with at least 30% slack capacity in the worst case, as for example in the case study from [9]. Schedulability analysis studies focused on characterization of RM [31] and EDF [10] scheduling policies have found that it is hard to guarantee schedulability of systems with high worst-case CPU utilization. Finally, the results of case studies from Section VIII-B show that BJIs tend to be significantly larger when compared to IJIs.

Our goal is to mine periodic task sets along with the period T and response time profile RTP for every periodic task τ. We split this goal into two problems:

P1 Let $\mathbb{T}$ be a timed trace of an RTES. Classify the tasks in $\mathbb{T}$ into periodic or non-periodic categories.

P2 Let $\mathcal{P}^T = \{\tau_u, \ldots, \tau_w\}$ be the set of tasks classified to the periodic category. Find the period T_v and response time profile RTP_v for every task $\tau_v \in \mathcal{P}^T$.

IV. Overview

PeTaMi, as shown in the flowchart of Figure 2 takes as input a timestamped trace of a RTES. During the first stage, traces are disaggregated into a set of trace projections (see Definition 2). A projection $\mathbb{P}_i$ is produced for every task $\tau_i \in \mathbb{T}$ and sent to the binary task classifier. The classifier finds the set $\{BJI\}$ after extracting IATs from $\mathbb{P}$. Using $\{BJI\}$, tasks are then classified into periodic or non-periodic category, and the subset of periodic tasks $\mathcal{P}^T$ with the set of projections for every task $\tau_v \in \mathcal{P}^T$ is fed into the temporal specifications miner. The miner computes approximations of the period T_v, and the response time profile RTP$_v$ for every task $\tau_v \in \mathcal{P}^T$.

V. Trace disaggregation

We consider those systems whose tracing module produces a timed entry per traced event. Each trace entry is comprised of a specific number of attributes precisely describing an event.

Definition 1 (Trace and Trace Entry): A trace $\mathbb{T} = \{\mathcal{E}_1, \ldots \mathcal{E}_N\}$ is a chronologically ordered list of trace entries. A trace entry $\mathcal{E} = \langle t, ID \rangle$ is a tuple consisting of a time stamp value t,and a task identifier ID.

Trace entries in $\mathbb{T}$ are ordered by their timestamps. Depending on the system and time resolution, more than one entry can have the same timestamp. Those cases are not critical, since PeTaMi does not make any assumptions on the ordering of events with the same timestamps.

The task identifier ($\mathcal{E}_i.ID$) is a conjunction of attributes in the trace that uniquely identifies a task. A task can be a process, a thread, or both, but its ID must be unique. For example, in a system with processes and threads the task identifier can be $ID =< PID, TID >$.

Definition 2 (Trace Projection): A trace projection $\mathbb{P} = \{\mathcal{E}_u, \ldots, \mathcal{E}_w\}$ over task τ is a filtered and chronologically ordered list of trace entries where every $\mathcal{E}_v.ID \in \mathbb{P}$ is equal to the identifier of task τ.

Trace projections are required by both stages of PeTaMi. The first stage uses timestamps in $\mathbb{P}$ to mine $\{WJI\}$. The second stage uses timestamps to discover typical RTs.

VI. Binary classification of real-time tasks

In this section, we address the challenge of binary classification of a real-time task τ into either periodic or non-periodic category given τ's trace projection $\mathbb{P}$. PeTaMi tackles this problem by firstly mining the set of τ's between-jobs inter-arrival times $\{BJI\}$ from the set of τ's inter-arrival times $\{IAT\}$, then extracting the set of τ's whole-job inter-arrival times $\{WJI\}$ from $\{BJI\}$, and finally measuring the spread of $\{WJI\}$. If the spread is smaller than a given threshold α, PeTaMi classifies the task into the periodic category.

We consider two possible cases of task scheduling and explain how PeTaMi performs classification in each of them.

A. Run-to-completion scheduling

In the first case, PeTaMi deals with a scenario where a task's jobs are always scheduled to run to completion. In this scenario, each job appears in the trace projection as a single event, as shown in Figure 3. This makes the set of IJIs empty ($\{IJI\} = \emptyset$), so that $\{IAT\} = \{BJI\}$. In general case, $\{WJI\}$ can be extracted from $\{BJI\}$ by sorting BJIs in ascending order of their timestamps and then calculating the differences between the timestamps of the adjacent BJIs.

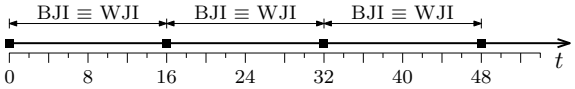


Fig. 3: IATs under run-to-completion scheduling.

WJIs of a periodic task tend to have the same value (i.e., $WJI \cong T$, the period of the task). Therefore, PeTaMi classifies a task to the periodic category, if $\{WJI\}$ has a small spread; otherwise, the task is classified to the non-periodic category. PeTaMi applies the quartile coefficient of dispersion (QCoD) statistic [45] to evaluate the spread of $\{WJI\}$:

$$QCoD = \frac{Q_3 - Q_1}{Q_3 + Q_1} * 100\%, \quad (1)$$

where Q_1 and Q_3 are the first and the third quartiles of $\{WJI\}$ accordingly. PeTaMi requires the user to provide a value for the threshold α to assess the smallness of the QCoD statistic. Setting α to 1% turned out to be a reasonable choice for real-world traces considered in Section VIII-B. On the other hand, the authors in [24] use a much more relaxed $\alpha = 10\%$ to assess the spread of recurrent events in execution traces. The strong points of the QCoD statistic are: (1) it is a dimensionless measure of spread, that is, its value does not depend on the values of WJIs; (2) it is a robust measure of spread, as its value is not sensible to outliers. Therefore, PeTaMi can use the same threshold α for tasks with different values of WJIs, and also correctly classify periodic tasks which occasionally miss their deadlines. As an example, consider sets $\{WJI_1\}$ and $\{WJI_2\}$ of two real-time tasks τ_1 and τ_2: $\{WJI_1\}$ = {32, 48, 40, 18, 53, 8, 25, 30, 49}, $\{WJI_2\}$ = { 4305, 4277, 9350, 4311, 4302, 4340, 4293, 8100, 4301 }. QCoD($\{WJI_1\}$) = 31.5, QCoD($\{WJI_2\}$) = 0.22. Although the WJIs from $\{WJI_1\}$ and $\{WJI_2\}$ have different ranges, we can easily compare their spreads with the QCoD statistic and conclude that the WJIs from $\{WJI_2\}$ have a much smaller spread than the ones from $\{WJI_1\}$, even in the presence of two clear outliers (9350 and 8100).

B. Preemptive scheduling

With preemptive scheduling, a task's execution can be interrupted by the real-time system at any moment in time. This way, a task's job can appear as a sequence of events in the execution trace. Therefore, the set of task's IATs will consist of both IJIs as well as BJIs: $\{IAT\} = \{IJI\} \cup \{BJI\}$ (see Figure 4).

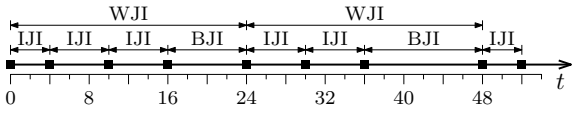


Fig. 4: IATs under preemptive scheduling.

As we explained in Section VI-A, PeTaMi requires $\{BJI\}$ to classify a given task τ. Therefore, the set $\{BJI\}$ must be extracted from the set $\{IAT\}$. The brute-force approach of extracting $\{BJI\}$ from $\{IAT\}$ would consist in considering every possible subset $\{S\} \sqsubset \{IAT\}$ to find the one $\{S\} = \{BJI\}$. Obviously, such method is computationally infeasible for realistic trace projections with thousands of IATs.

PeTaMi relies on the Assumption 1 to efficiently mine an approximate set of BJIs from IATs. According to Assumption 1, BJIs tend to be larger than IJIs. Therefore, if we denote the set of i largest IATs as $\{S_i\}$, $i \in [1:N]$, where N is the number of IATs, then we can expect that there exists k, such that $\{S_k\}$ contains BJIs with high precision and recall (i.e. with a high F-score), and whose F-score is maximal among other $\{S_j\}$, $j \neq k$. In case $\{BJI\}$ and $\{IJI\}$ are linearly separable, $\texttt{F-score}(\{S_k\}) = 1$, that is, $\{S_k\} = \{BJI\}$. In order to find k, we need to consider all $\{S_j\}$, $j \in [1, N]$, and pick the one with the highest F-score. The function `F-score`, however, is undefined as the set $\{BJI\}$ is unknown.

The following observation allows us to approximate the function `F-score`: the F-score of retrieving $\{BJI\}$ with $\{S_k\}$ is negatively correlated with the spread of the WJIs extracted from $\{S_k\}$ using the technique presented in Section VI-A. That is, the higher is $\texttt{F-score}(\{S_k\})$, the lower is $\texttt{spread}(\{WJI_{S_k}\})$. Therefore, the $\{S_k\}$ with the maximal F-score among all $\{S_j\}$, $j \in [1, N], j \neq k$, will have the minimal spread of the corresponding WJIs. To illustrate this negative correlation, consider the case where $\texttt{spread}(\{WJI_{BJI}\}) = 0$, that is, all the task's WJIs have identical values. If we add $\iota \in \{IJI\}$ to $\{BJI\}$, so that $\{BJI'\} = \{BJI\} + \iota$, then we get $\texttt{spread}(\{WJI_{BJI'}\}) > 0$, as this would imply replacing some $w \in \{WJI\}$ with w_1 and w_2 ($w = w_1 + w_2$). Trivially, $\texttt{F-score}(\{BJI'\}) < \texttt{F-score}(\{BJI\})$. Similarly, by removing a random $b \in \{BJI\}$: $\{BJI'\} = \{BJI\} - b$, we get $\texttt{spread}(\{WJI_{BJI'}\}) > 0$, and $\texttt{F-score}(\{BJI'\}) < \texttt{F-score}(\{BJI\})$. The same generally holds when $\texttt{spread}(\{WJI_{BJI}\}) > 0$.

PeTaMi mines an approximate set of BJIs $\{S_j\}$ by calculating $\texttt{spread}(\{WJI_{S_j}\})$ with the QCoD statistic (Equation 1) for all $j \in [1, N]$, and then picks the one with the smallest value of spread. If the computed value of the QCoD statistic is smaller than the threshold α, then the task is classified as a periodic one, and the set $\{S_j\}$ is returned as an approximation of $\{BJI\}$. Algorithm 1 presents the pseudocode of the classification stage of PeTaMi. Note that we do not consider $\{S_j\}$ for $j < 5$, as the QCoD statistic uses the first and the third quartiles of $\{S_j\}$, which are present in the set only if $j \geq 5$. Algorithm 1 can as well classify tasks scheduled under the run-to-completion scheduling.

Function `classifyTask` (IAT,$\alpha = 1$)
  Data: IAT: set of task's IATs sorted in descending order, α: threshold for spread
  Result: $isPeriodic$: boolean, BJI: set of task's BJIs
  q = 5
  $numBJI$ = length(IAT)$-q+1$
  foreach $i \in 1{:}numBJI$ **do**
    $BJIs[i]$ = $IAT[1{:}(i+q-1)]$
    $WJIs$ = extract from $BJIs[i]$ (see Section VI-A)
    $spread[i]$ = `QCoD` ($WJIs$) (see Equation 1)
  end
  $minSpread$ = `min` ($spread$)
  if $minSpread > \alpha$ **then**
    return ($FALSE$)
  else
    m = index of $minSpread$ in $spread$
    $finalBJI$ = $BJIs[m]$
    return ($TRUE$, $finalBJI$)

Algorithm 1: Classification of real-time tasks in PeTaMi

VII. Mining temporal specifications of periodic tasks

Once a periodic task is identified, it is crucial to report its period (T) and response time profile (RTP), as we argued in Section I. Therefore, the second stage of PeTaMi mines temporal specifications of periodic tasks from their trace projections.

Period mining can be performed as long as the set of BJIs of a periodic task is known. Indeed, T will be equal to the most common value among the WJIs extracted from the set $\{BJI\}$. In statistical terms, having a distribution of WJIs, its *mode* represents the most common value among the intervals in $\{WJI\}$: $T = \texttt{mode}(\{WJI\})$.

As we argued in Section III, a RTP of a periodic task τ must be represented as a set of τ's "typical" RTs. As we can see in Figure 1, τ's set of RTs ($\{RT\}$) can be calculated using its sets $\{WJI\}$ and $\{BJI\}$: $RT_i = WJI_i - BJI_i$. At the same time, to find the "typical" RTs we must first split the set $\{RT\}$ into groups, such that RTs from the same group tend to be more similar to each other than to RTs from other groups; "typical" RTs then will be equal to the representative RTs from each such group.

PeTaMi applies partition clustering in order to mine typical RTs of a given task τ. Partitioning clustering algorithms aim at organizing the elements of a set into several exclusive groups in an automatic (unsupervised) way [23]. Objects within a cluster are similar to one another and dissimilar from objects in other clusters. Dissimilarity of a pair of objects is characterized by the distance (d) between them. In our case, objects are RTs, and d is simply the absolute difference in value between pairs of RTs: $d = |RT_i - RT_j|$. One of the most popular partitioning clustering algorithms is the ***k-medoids*** method [26]. Its objective is to find

$$\underset{\{L_1,L_2,\cdots,L_k\}}{arg\,min} \sum_{i=1}^{k} \sum_{x \in L_i} (x - c_i)^2, \tag{2}$$

where $x = \{x_1, x_2, \cdots, x_n\}$ is the set of objects, c_i is the *medoid* [1] of the elements assigned to the cluster L_i, and k is the number of clusters. Informally, k-medoids aims to partition n objects into k ($k \leq n$) clusters $L = \{L_1, L_2, \cdots, L_k\}$ so as to minimize the sum of distances from each object in a cluster to the cluster's medoid c_i. We opted for the k-medoids method instead of the extremely popular k-means method [34], because by grouping objects around k medoids rather than around k centroids, k-medoids allows to tolerate outliers. For example, given a set $\{RT\} = \{10, 10, 10, 10, 10, 40, 40, 40, 40, 40, 100\}$ and $k = 2$, k-medoids puts $RT_{11} = 100$, an obvious outlier, into the cluster with medoid $= 40$; instead, k-means assigns RT_{11} to a separate cluster and puts RT_1,...,RT_{10} into a single cluster. This robustness of the k-medoids method is crucial to clustering RTs, as a periodic task may miss its deadline during system execution, hence, its set $\{RT\}$ may have abnormally large RTs. PeTaMi uses an efficient implementation of the k-medoids algorithm called CLARA [27].

Partitioning clustering algorithms require a manual choice of the number of clusters to group the objects into (parameter k in Equation 2). PeTaMi, therefore, must find the right value for k automatically. The state-of-the-art on the estimation of the optimal number of clusters in a dataset is vast and spans several decades [38] [17] [15]. One of the most popular methods to estimate the value of k is the ***gap statistic*** [41]. The idea behind this method is to compare the value of Equation 2 with a null reference distribution of the objects, i.e., a distribution with no obvious clustering, for a range of k values. The estimate for the optimal number of clusters k is the value for which Equation 2 falls the furthest away from the null reference distribution. A distinctive feature of the gap statistic is its ability to return $k = 1$ if a given set of objects does not form clusters. This is important for RTP mining in case RTs have only a single typical value, i.e., the task's jobs tend to take the same amount of time to complete. Algorithm 2 summarizes RTP mining stage of PeTaMi.

Function `RTPmining`(RT)
  Data: RT: a set of task's response times
  Result: RTP: an array of representative RTs
  k = `gap`(RT) (see [41])
  $clus$ = `CLARA`(RT, k) (see [27])
  foreach $i \in 1 : k$ **do**
    $RTP[i]$ = `mode`($clus[i]$)
  end
  return (RTP)

Algorithm 2: RTP mining in PeTaMi

VIII. Experiments

We evaluate the capacity of PeTaMi to correctly mine task sets and their temporal specifications from both synthetic datasets and real-world execution traces of embedded real-time systems.

A. Evaluation on synthetic data

In this section, we evaluate the accuracy of PeTaMi's RTP mining stage on synthetically generated data. We do not address the evaluation of the classification stage because, as shown in Section VI, PeTaMi correctly classifies periodic tasks as long as the Assumption 1 holds true. Therefore, we leave validation of the classification stage of PeTaMi to case studies presented in Section VIII-B.

We evaluate RTP mining using a random sample from a heterogeneous mixture of two Gaussians and one uniform distribution. Observations from the Gaussians represent task's normal RTs, while observations from the uniform distribution represent outlier RTs. Our choice of Gaussian distributions to model normal RTs is supported by the non-determinism of modern real-time systems: if a given task τ has m execution scenarios each of which requires a particular response time r_i,

[1] an object of a cluster whose average dissimilarity to all the objects in the cluster is minimal

$i \in [1, m]$, then τ's RTP will not be a set of m values but it will rather follow a multimodal distribution where each mode corresponds to one and only one r_i, $i \in [1, m]$. Indeed, τ's RTs will fluctuate around their normal values r_i. On the other hand, we expect outlier RTs to follow a uniform distribution, as their values are completely random.

The accuracy of RTP mining is evaluated based on the number of clusters mined by PeTaMi on the synthetic data. We consider that PeTaMi correctly mines a RTP if the returned number of clusters is equal to 2.

We ran 2^{IV} experimental design and analyzed it with ANOVA to find out the characteristics of mixtures of distributions that reduce the accuracy of PeTaMi's RTP mining stage. We chose the following factors and their corresponding low (-1) and high (1) levels:

1) Distance between the means of Gaussians (δ): 3 and 30.
2) Ratio between the number of observations in Gaussians (ρ): 50/50, 75/25.
3) Standard deviation of Gaussians (σ): 0.1 and 1.
4) Ratio of the number of observations in the uniform distribution to the cumulative number of observations in two Gaussians (ν): 1/100 and 10/100.

We chose 3 as the low value for δ according to [1] to guarantee that none of the pairs of σ and ρ values will make the distribution unimodal. Table I shows the reduced ANOVA table for the synthetic data described above. Figure 5 presents the effects of individual factors on the accuracy of RTP mining.

Factors	Df	SS	MS	F val	Pr(>F)	Sig
δ	1	9.00	9.000	40.500	0.000131	***
σ	1	9.00	9.000	40.500	0.000131	***
δ:σ	1	2.25	2.250	10.125	0.011149	*
δ:ν	2	3.25	1.625	7.313	0.012999	*
σ:ν	1	2.25	2.250	10.125	0.011149	*
Residuals	9	2.00	0.222			

Significance codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

TABLE I: ANOVA table for synthetic data

The first conclusion we draw from Table I is that there is a significant effect of δ on the accuracy of RTP mining. Indeed, when the distance becomes large, PeTaMi tends to create spurious clusters in order to group noisy RTs situated between the Gaussians. On the other hand, when δ becomes small, PeTaMi may put RTs from both Gaussians into the same cluster (see the top left box plot in Figure 5).

The standard deviation of Gaussians σ also has a significant effect on the accuracy of PeTaMi's RTP mining. In fact, small values of σ make the gap between the Gaussians more pronounced; hence, PeTaMi is more prone to group outlier RTs situated between the Gaussians together, and not with the RTs from Gaussians. At the same time, large values of σ increase the probability that PeTaMi will cluster RTs from the Gaussians together (see the bottom left box plot in Figure 5).

The reason of the significance of the interaction between δ and σ factors is twofold. Firstly, a combination of a small σ and a large δ increases the probability that PeTaMi creates additional clusters to group the outlier RTs between the

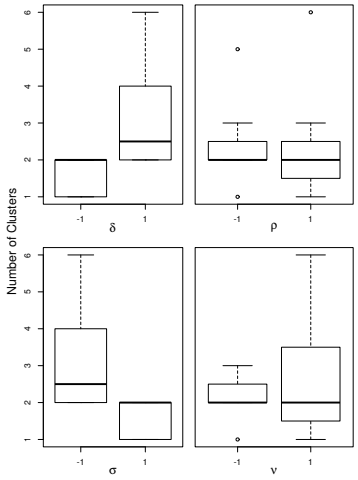


Fig. 5: Effects of the parameters of the mixture distribution on RTP mining accuracy (correct number of clusters is always 2).

Gaussians. Secondly, a combination of a large σ and a small δ will likely result in PeTaMi mixing RTs from the two Gaussians into a single cluster. The same can be said about the interactions between δ and ν, and between σ and ν: large values of ν will make PeTaMi create spurious clusters when δ is large and/or σ is small.

B. Case Studies

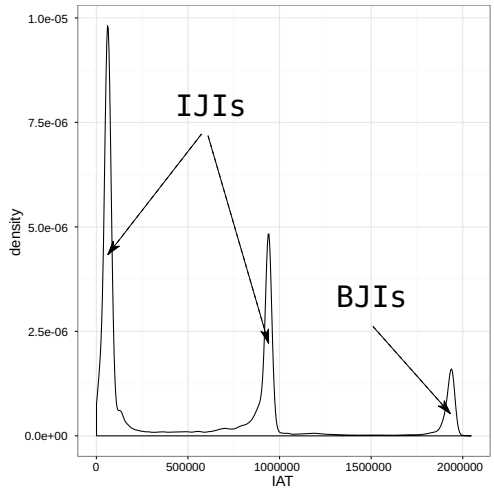
In this section, we validate both stages of PeTaMi on two case studies.

UAV case study: We use kernel event traces generated by an unmanned aerial vehicle (UAV) shown in Figure 6 running the

Fig. 6: Unmanned Aerial Vehicle (UAV)

```
time        event       pid     tid
....        ....        ....    ....
891422416   THRECEIVE   61465   4
891423916   THRUNNING   1       1
893367166   THRUNNING   61465   4
893368458   THREADY     1       1
893372750   THREADY     61465   3
893419708   THRECEIVE   61465   4
893421208   THRUNNING   61465   3
....        ....        ....    ....
```

Fig. 7: UAV Trace Snippet

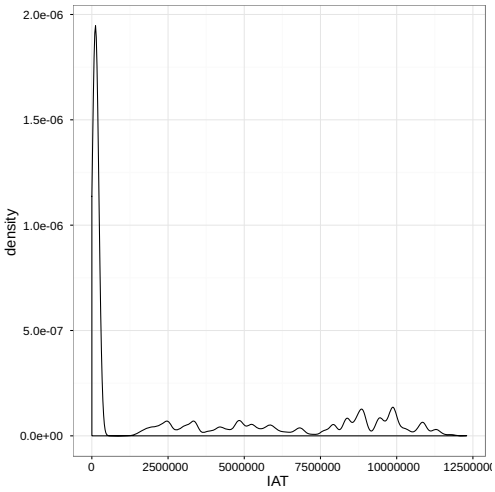


(a) Probability density function of IATs generated by a periodic task. Distributions of IJIs and BJIs as mined by PeTaMi are marked.

time	event	pid	tid	IAT	IAT type
....	....	....	....	....	....
24896653179	THRECEIVE	61465	4	1928401	BJI
24898583583	THRUNNING	61465	4	82208	IJI
24898659458	THRECEIVE	61465	4	718333	IJI
24899375583	THRUNNING	61465	4	53292	IJI
24899427208	THRECEIVE	61465	4	940458	IJI
24900366166	THRUNNING	61465	4	67418	IJI
24900432375	THRECEIVE	61465	4	947166	IJI
24901377875	THREADY	61465	4	33333	IJI
24901409833	THRUNNING	61465	4	63542	IJI
24901468833	THRECEIVE	61465	4	1900792	BJI
24903368125	THRUNNING	61465	4	80958	IJI
....	....	....	....	....	....

(b) Trace snippet showing events generated by a periodic task annotated with IATs and IAT types as mined by PeTaMi.

Fig. 8: Temporal behavior of a periodic task



(a) Probability density function of IATs generated by a non-periodic task.

time	event	pid	tid	IAT
....	....	....	....	....
24964395833	THRUNNING	20501	5	132125
24964526666	THRECEIVE	20501	5	3923251
24968448458	THRUNNING	20501	5	124500
24968571750	THRECEIVE	20501	5	11254832
24979825041	THREADY	20501	5	2085
24979841916	THRUNNING	20501	5	117609
24979954125	THRECEIVE	20501	5	4586341
....	....	....	....	....

(b) Trace snippet showing events generated by a non-periodic task annotated with IATs.

Fig. 9: Temporal behavior of a non-periodic task

real-time operating system QNX Neutrino 6.4. The UAV was developed at the University of Waterloo, received the Special Flight Operating Certificate (SFOC), and flew real mapping and payload-drop missions in Nova Scotia and Ontario [40].

The trace snippet in Figure 7 shows a subset of attributes found in the execution traces considered in this case study. The snippet is generated using the tracelogger and traceprinter utilities available in QNX Neutrino. In the experiments, a task τ has a unique ID that combines the values from the two attributes shown in the snippet as pid and tid. This way, the snippet shows various events generated by three tasks τ_1 = "61465.4", τ_2 = "1.1", τ_3 = "61465.3".

We used eight UAV traces where each trace consists of a stream of roughly 150K events generated by ten tasks. These traces contain both periodic and non-periodic tasks. Figures 8 and 9 show probability density functions (PDFs) of IATs generated by a pair of representative periodic and non-periodic tasks along with the annotated trace snippets capturing events generated by these tasks. As we can see in Figure 8a, the PDF of the IATs generated by a periodic task is represented by several well-defined peaks. This implies that the task tends to generate events with consistent IATs. Indeed, we can notice in Figure 8b that a task's job consists of a sequence of nine events: eight of those events have IATs that fall into one of the two leftmost bumps from Figure 8a, while the last job's event ("THRECEIVE") generates an IAT that falls into the rightmost bump. On the other hand, Figure 9a shows that a big number of IATs generated by a non-periodic task tend to be uniformly distributed. In fact, the non-periodic task depicted in Figure 9 is a sporadic task, and the trace snippet containing events executed by this task (Figure 9b) shows that the task generates arbitrarily large IATs.

PeTaMi's classification stage showed a perfect F-score of 1.0 on UAV traces, as all periodic and non-periodic tasks were correctly classified. The perfect F-score can be explained by the clear difference in minimal spreads ($minSpread$, see Algorithm 1) between periodic and non-periodic tasks. For example, the periodic task from Figure 8 has the following values of $minSpread$ in eight UAV traces: $\{0.11, 0.13, 0.13, 0.12, 0.18, 0.2, 0.15, 0.16\}$. At the same time, the non-periodic task from Figure 9 has $minSpread$ of $\{7.3, 6.0, 5.6, 3.0, 5.6, 4.8, 4.5, 4.4\}$. Fixing the threshold α at 1% allowed to achieve the perfect F-score.

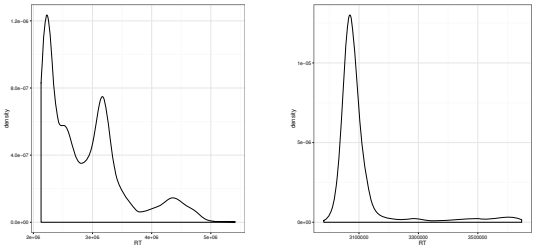


Fig. 10: Problematic PDFs of RTs in UAV case study

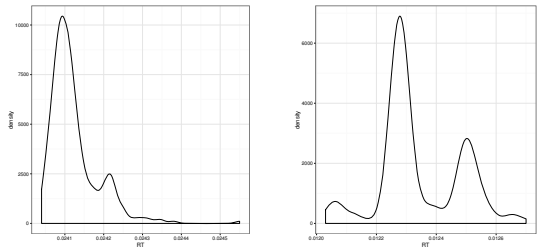


Fig. 12: Problematic PDFs of RTs in CAN case study

PeTaMi mined RTPs of periodic tasks with the accuracy of 83%: it returned a correct set of typical RTs for 20 out of 24 periodic tasks captured in eight traces. We evaluate the accuracy of RTP mining by comparing the PDF of the task's RTs with the number and values of the typical response times returned by PeTaMi: if each distinct peak in the PDF has a corresponding entry in the mined RTP, then the mining is considered as being correct. Figure 10 shows PDFs of two sets of RTs that resulted in wrong RTP mining (the other two were similar to the left-hand graph). In the first case (the left-hand graph), PeTaMi mined only one typical RT, while there are three visible peaks. This PDF is characterized by a small distance δ between the peaks combined with a large standard deviation σ of the peaks. Indeed, our findings in Section VIII-A showed that interaction between δ and σ is a significant factor in PeTaMi's accuracy. In the second case (the right-hand graph), PeTaMi returned two typical RTs, while there is only one visible peak. The reason of incorrect mining is the presence of "noisy" RTs which PeTaMi decided to put into a separate cluster, thus, returning a spurious typical response time. Again, our findings from Section VIII-A support this case, because the interaction between δ and the ratio of noise ν was found to influence the accuracy of PeTaMi.

CAN case study: We captured the traces of Controller Area Network (CAN) messages from a Toyota RAV4 during eleven driving scenarios. Each trace consists of roughly 10K CAN messages which represents 10 seconds of message capturing during the following driving scenarios: starting and stopping the engine, accelerating and decelerating between speeds of 0 and 20km/h, 0 and 40 km/h, making lane changes to the left and to the right, and driving around a ring road. Each trace contains between 15 and 20 tasks, all of which are periodic. The trace snippet shown in Figure 11 presents a subset of fields

time	id	length	bitcount	mode
....	....	....	....	....
1.9986	2C1	228000	118	Rx
2.00194	B0	192000	100	Rx
2.00209	320	144000	76	Rx
2.00229	B2	190000	99	Rx
2.00576	2C6	238000	123	Rx
2.00793	20	148000	78	Rx
....	....	....	....	....

Fig. 11: CAN Trace Snippet

characterizing each traced event (CAN message), as well as a few events as seen in captured traces. We use the "id" field of CAN messages to identify a task.

PeTaMi's classification stage showed a perfect F-score of 1.0 on CAN traces: all tasks were correctly classified as periodic. Similarly to the UAV case study, the threshold of spread $\alpha = 1\%$ was a good choice. Indeed, only a few periodic tasks in CAN traces had $minSpread > 0.1\%$, which was far smaller than 1%.

PeTaMi mined RTPs of periodic tasks with the accuracy of 74.4%: 134 out of 180 mined RTPs were correct. We evaluated the accuracy of RTP mining using the same approach applied in the UAV case study. Figure 12 shows PDFs of two representative sets of RTs for which PeTaMi returned incorrect RTP. In the first case (the left-hand graph), PeTaMi returned only one typical RT while there are two clearly visible peaks, hence, two typical RTs. In the second case (the right-hand graph), PeTaMi mined only 2 typical RTs corresponding to the two rightmost peaks. Both cases confirm our conclusion from Section VIII-A that the distance δ between the peaks and the standard deviation σ of the peaks influence the accuracy of PeTaMi's RTP mining stage.

IX. Discussion

Experimental results in Section VIII validated the effectiveness of PeTaMi in both the classification of real-time tasks and the mining of temporal specifications of periodic tasks from execution traces of embedded real-time systems.

The classification stage of PeTaMi showed perfect F-scores in case studies. This can be explained by the truthfulness of our assumption that BJIs of periodic tasks tend to have larger values than their IJIs in real-world execution traces. Indeed, a real-time periodic task must meet its deadline; if a set of periodic tasks has a feasible priority order, then an appropriate scheduler will try to make every task meet its deadline by minimizing task's RTs. As a consequence, task's BJIs are maximized. Therefore, the set of BJIs of a periodic task will tend to have larger values than its set of IJIs. We must note, however, that there exist two scenarios where PeTaMi will fail to correctly classify a real-time task:

1) The task is periodic, and its period is respected at run time, but task's BJIs and IJIs have random values. Indeed, this scenario will imply a uniform distribution of IATs so that PeTaMi will not be able to find an accurate

approximation of the set $\{BJI\}$ using Algorithm 1. Such scenario, however, can occur only for low-priority tasks in highly utilized systems.

2) The largest $n \geq 5$ IATs of a non-periodic task are evenly spaced in time. Given the inherent randomness in IATs of non-periodic tasks, such scenario is extremely unlikely to happen in real-world execution traces.

PeTaMi showed an acceptable accuracy of RTP mining in case studies. In Section VIII-A, we analyzed the influence of various characteristics of the probability density function (PDF) of the task's RTs on the accuracy of RTP mining. According to our findings, the standard deviation of the Gaussians which model the distributions of task's RTs as well as the distance between the modes of these Gaussians are the most significant factors. The characteristics of the problematic PDFs from the case studies (Figures 10 and 12) confirmed our findings. We envision to improve the accuracy of the RTP mining stage in the future. One possibility is to replace the k-medoids method by a clustering algorithm based on kernel density estimation (KDE), such as the mean shift algorithm [16] [12]. The idea behind this type of clustering algorithms is to leverage sound statistical properties of KDE to group observations around the regions of high data density. In our case, such method could be beneficial in ignoring the regions of low density (i.e., avoiding the creation of spurious clusters containing outlier RTs), while treating peaks in high-density regions more carefully (i.e., correctly detecting typical RTs having a small difference in values).

X. Conclusion

In this work, we address the challenge of reverse engineering modern real-time systems by representing their runtime behavior with a set of independent periodic tasks. To this end, we introduced PeTaMi – a novel approach to mine periodic tasks, along with their periods and response time profiles from execution traces of real-time embedded systems.

PeTaMi takes as input an execution trace and a set of trace fields encoding a task. Upon termination, PeTaMi returns a "periodic"/"non-periodic" classification label for each task, and in case the "periodic" label is assigned, PeTaMi returns the task's period and response time profile represented as a set of typical response times.

We validated the applicability of PeTaMi on two case studies and showed that it is possible to discover the period and the response time profile of a periodic task using only its inter-arrival times extracted from the execution trace.

Appendix

To support Assumption 1, we performed the following simulation using the YAO-SIM tool [2] . We generated 1000 periodic task sets for each utilization from 0.5 to 1.0 with a step 0.05, and simulated execution of the tasks on a single CPU for 10^7 time units (following a similar practice as in [44]) using RM and EDF scheduling algorithms. The number of tasks in each task set was picked randomly from the interval $[3, 10]$. The worst-case execution times of tasks were generated as random integers uniformly distributed in the interval $[1, 30]$, while the tasks' periods were computed from WCETs and utilizations using the UUniFast algorithm [5]. Figure 13 presents the results of the simulation. The horizontal line corresponds to the case where the mean IJI value of a particular task execution is equal to its mean BJI value. By comparing the simulation results with this line, we can conclude that Assumption 1 holds true for systems with CPU utilization of up to 85%.

[2]http://yaosim.sssup.it/

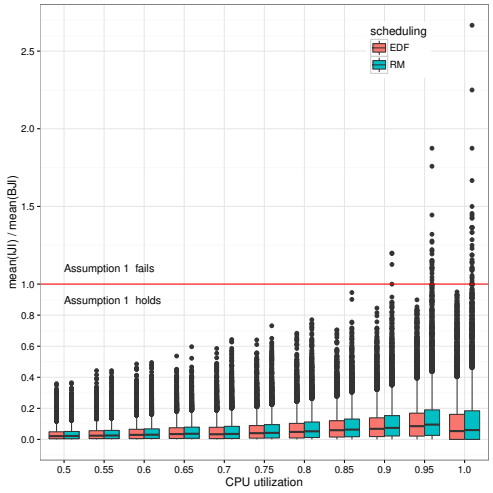


Fig. 13: Simulation results supporting Assumption 1.

References

[1] Javad Behboodian. On the modes of a mixture of two normal distributions. *Technometrics*, 12(1):131–139, 1970.

[2] Alexander Bergmayr, Hugo Bruneliere, Jordi Cabot, Jokin Garcia, Tanja Mayerhofer, and Manuel Wimmer. frex: fuml-based reverse engineering of executable behavior for software dynamic analysis. In *Proceedings of the 8th International Workshop on Modeling in Software Engineering*, pages 20–26. ACM, 2016.

[3] Ivan Beschastnikh, Jenny Abrahamson, Yuriy Brun, and Michael D Ernst. Synoptic: Studying logged behavior with inferred models. In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, pages 448–451. ACM, 2011.

[4] Armin Biere, Alessandro Cimatti, Edmund M Clarke, Ofer Strichman, and Yunshan Zhu. Bounded model checking. *Advances in computers*, 58:117–148, 2003.

[5] Enrico Bini and Giorgio C Buttazzo. Measuring the performance of schedulability tests. *Real-Time Systems*, 30(1-2):129–154, 2005.

[6] Marco Bonato, Giuseppe Di Guglielmo, Masahiro Fujita, Franco Fummi, and Graziano Pravadelli. Dynamic property mining for embedded software. In *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, pages 187–196. ACM, 2012.

[7] James F Bowring, James M Rehg, and Mary Jean Harrold. Active learning for automatic classification of software behavior. In *ACM SIGSOFT Software Engineering Notes*, volume 29, pages 195–205. ACM, 2004.

[8] Manfred Broy. Challenges in automotive software engineering. In *Proceedings of the 28th International Conference on Software Engineering*, ICSE '06, pages 33–42, New York, NY, USA, 2006. ACM.

[9] Alan Burns, Andy J Wellings, CM Bailey, and E Fyfe. The olympus attitude and orbital control system a case study in hard real-time system design and implementation. In *Ada-Europe International Conference*, pages 19–35. Springer, 1993.

[10] Giorgio C Buttazzo. Rate monotonic vs. edf: judgment day. *Real-Time Systems*, 29(1):5–26, 2005.

[11] Gerardo Canfora, Massimiliano Di Penta, and Luigi Cerulo. Achievements and challenges in software reverse engineering. *Communications of the ACM*, 54(4):142–151, 2011.

[12] Miguel A Carreira-Perpinán. A review of mean-shift algorithms for clustering. *arXiv preprint arXiv:1503.00687*, 2015.

[13] Sofia Cassel, Falk Howar, Bengt Jonsson, and Bernhard Steffen. Active learning for extended finite state machines. *Formal Aspects of Computing*, 28(2):233–263, 2016.

[14] Samarjit Chakraborty, Martin Lukasiewycz, Christian Buckl, Suhaib Fahmy, Naehyuck Chang, Sangyoung Park, Younghyun Kim, Patrick Leteinturier, and Hans Adlkofer. Embedded systems and software challenges in electric vehicles. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '12, pages 424–429, San Jose, CA, USA, 2012. EDA Consortium.

[15] Malika Charrad, Nadia Ghazzali, Véronique Boiteau, Azam Niknafs, and Maintainer Malika Charrad. Package "nbclust". *J. Stat. Soft*, 61:1–36, 2014.

[16] Yizong Cheng. Mean shift, mode seeking, and clustering. *IEEE transactions on pattern analysis and machine intelligence*, 17(8):790–799, 1995.

[17] Mark Ming-Tso Chiang and Boris Mirkin. Intelligent choice of the number of clusters in k-means clustering: an experimental study with different cluster spreads. *Journal of classification*, 27(1):3–40, 2010.

[18] Elliot J. Chikofsky and James H Cross. Reverse engineering and design recovery: A taxonomy. *IEEE software*, 7(1):13–17, 1990.

[19] Greta Cutulenco, Yogi Joshi, Apurva Narayan, and Sebastian Fischmeister. Mining timed regular expressions from system traces. In *Proceedings of the 5th International Workshop on Software Mining*, pages 3–10. ACM, 2016.

[20] Pär Emanuelsson and Ulf Nilsson. A comparative study of industrial static analysis tools. *Electronic notes in theoretical computer science*, 217:5–21, 2008.

[21] Michael D Ernst, Jeff H Perkins, Philip J Guo, Stephen McCamant, Carlos Pacheco, Matthew S Tschantz, and Chen Xiao. The daikon system for dynamic detection of likely invariants. *Science of Computer Programming*, 69(1):35–45, 2007.

[22] Brice Govin, Nicolas Anquetil, Anne Etien, Arnaud Monegier Du Sorbier, and Stéphane Ducasse. Reverse engineering tool requirements for real time embedded systems. In *SATToSE'15*, 2015.

[23] Jiawei Han, Jian Pei, and Micheline Kamber. *Data mining: concepts and techniques*. Elsevier, 2011.

[24] Oleg Iegorov, Vincent Leroy, Alexandre Termier, Jean-François Méhaut, and Miguel Santana. Data mining approach to temporal debugging of embedded streaming applications. In *Proceedings of the 12th International Conference on Embedded Software*, pages 167–176. IEEE Press, 2015.

[25] Taylor T. Johnson, Raghunath Gannamaraju, and Sebastian Fischmeister. A survey of electrical and electronic (e/e) notifications for motor vehicles. In *24th International Technical Conference on the Enhanced Safety of Vehicles (ESV)*, pages 1–15, Gothenburg, Sweden, 2015.

[26] Leonard Kaufman and Peter Rousseeuw. *Clustering by means of medoids*. North-Holland, 1987.

[27] Leonard Kaufman and Peter J Rousseeuw. *Finding groups in data: an introduction to cluster analysis*, volume 344. John Wiley & Sons, 2009.

[28] Holger M Kienle, Hausi A Müller, and Johan Kraft. *Software reverse engineering in the domain of complex embedded systems*. INTECH Open Access Publisher, 2012.

[29] Ralf Kollmann, Petri Selonen, Eleni Stroulia, Tarja Systa, and Albert Zundorf. A study on the current state of the art in tool-supported uml-based static reverse engineering. In *Reverse Engineering, 2002. Proceedings. Ninth Working Conference on*, pages 22–32. IEEE, 2002.

[30] Phillip A Laplante. *Real-Time Systems Design and Analysis (3rd Edition)*. Wiley, 2004.

[31] John Lehoczky, Lui Sha, and Ye Ding. The rate monotonic scheduling algorithm: Exact characterization and average case behavior. In *Real Time Systems Symposium, 1989., Proceedings.*, pages 166–171. IEEE, 1989.

[32] K Rustan M Leino, James B Saxe, and Raymie Stata. Checking java programs via guarded commands. In *ECOOP Workshops*, pages 110–111, 1999.

[33] Caroline Lemieux, Dennis Park, and Ivan Beschastnikh. General ltl specification mining (t). In *Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference on*, pages 81–92. IEEE, 2015.

[34] Stuart Lloyd. Least squares quantization in pcm. *IEEE transactions on information theory*, 28(2):129–137, 1982.

[35] David Lo, Hong Cheng, Jiawei Han, Siau-Cheng Khoo, and Chengnian Sun. Classification of software behaviors for failure detection: a discriminative pattern mining approach. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 557–566. ACM, 2009.

[36] Patricia López Cueva, Aurélie Bertaux, Alexandre Termier, Jean François Méhaut, and Miguel Santana. Debugging embedded multimedia application traces through periodic pattern mining. In *Proceedings of the tenth ACM international conference on Embedded software*, pages 13–22. ACM, 2012.

[37] Ramy Medhat, S Ramesh, Borzoo Bonakdarpour, and Sebastian Fischmeister. A framework for mining hybrid automata from input/output traces. In *Proceedings of the 12th International Conference on Embedded Software*, pages 177–186. IEEE Press, 2015.

[38] Glenn W Milligan and Martha C Cooper. An examination of procedures for determining the number of clusters in a data set. *Psychometrika*, 50(2):159–179, 1985.

[39] Giles Reger, Howard Barringer, and David Rydeheard. Automata-based pattern mining from imperfect traces. *ACM SIGSOFT Software Engineering Notes*, 40(1):1–8, 2015.

[40] Yassir Rizwan, Steven L Waslander, and Christopher Nielsen. Nonlinear aircraft modeling and controller design for target tracking. In *Proceedings of the 2011 American Control Conference*, pages 3191–3196. IEEE, 2011.

[41] Robert Tibshirani, Guenther Walther, and Trevor Hastie. Estimating the number of clusters in a data set via the gap statistic. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 63(2):411–423, 2001.

[42] Wil Van der Aalst, Ton Weijters, and Laura Maruster. Workflow mining: Discovering process models from event logs. *IEEE Transactions on Knowledge and Data Engineering*, 16(9):1128–1142, 2004.

[43] Christian Hagen Jeff Sorenson Steven Hurt Dan Wall. Software: The brains behind u.s. defense systems. Technical report, ATKearney, 2012.

[44] Gang Yao, Giorgio Buttazzo, and Marko Bertogna. Feasibility analysis under fixed priority scheduling with limited preemptions. *Real-Time Systems*, 47(3):198–223, 2011.

[45] Daniel Zwillinger and Stephen Kokoska. Standard probability and statistical tables and formula, 2000.