Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Definindo relações semânticas entre famílias de cenários implícitos.

Caio Batista de Melo

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

Orientadora
Prof.ª Dr.ª Genaína Nunes Rodrigues

Brasília
2017

Universidade de Brasília — UnB
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Bacharelado em Ciência da Computação

Coordenador: Prof. Dr. Rodrigo Bonifácio de Almeida

Banca examinadora composta por:

Prof.ª Dr.ª Genaína Nunes Rodrigues (Orientadora) — CIC/UnB
Prof. Dr. André Luiz Fernandes Cançado — EST/UnB
Prof. Dr. Rodrigo Bonifácio de Almeida — CIC/UnB

CIP — Catalogação Internacional na Publicação

Melo, Caio Batista de.

Definindo relações semânticas entre famílias de cenários implícitos. / Caio Batista de Melo. Brasília : UnB, 2017.
63 p. : il. ; 29,5 cm.

Monografia (Graduação) — Universidade de Brasília, Brasília, 2017.

1. engenharia de software, 2. dependabilidade, 3. confiabilidade, 4. cenários implícitos, 5. sistemas concorrentes

CDU 004.4

Endereço: Universidade de Brasília
Campus Universitário Darcy Ribeiro — Asa Norte
CEP 70910-900
Brasília–DF — Brasil

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Definindo relações semânticas entre famílias de cenários implícitos.

Caio Batista de Melo

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

Prof.ª Dr.ª Genaína Nunes Rodrigues (Orientadora)
CIC/UnB

Prof. Dr. André Luiz Fernandes Cançado
EST/UnB

Prof. Dr. Rodrigo Bonifácio de Almeida
CIC/UnB

Prof. Dr. Rodrigo Bonifácio de Almeida
Coordenador do Bacharelado em Ciência da Computação

Brasília, 15 de julho de 2017

Dedicatória

Dedico este trabalho a você que está lendo. Espero que este lhe ajude em seus próprios trabalhos e estudos, assim como muitos outros ajudaram para que este fosse concluído.

Agradecimentos

Aos meus professores que me propiciaram a oportunidade de chegar até aqui, sempre contribuindo para o meu crescimento pessoal e profissional. Aos meus amigos que me acompanharam nesta caminhada e fizeram que tudo parecesse tão rápido. E principalmente à minha família por me apoiar em todos os momentos e me colocar no caminho de grandes conquistas.

Resumo

Quando um sistema concorrente é modelado a base de cenários, muitas vezes podemos encontrar durante a execução do sistema um cenário que não havia sido previsto. Este é chamado de cenário implícito e decorre de uma modelagem errônea. O tratamento de cenários implícitos é um processo de extrema importância para a correta execução do sistema, porém demanda tempo e esforço do usuário. Logo, este trabalho visa facilitar a parte da detecção para que o usuário possa tratá-los de uma forma potencialmente mais simples e rápida. Isto é alcançado utilizando o conceito de famílias de cenários implícitos, que são grupos de cenários similares cujos cenários que os compõe ocorrem provavelmente pelos mesmos fatores. Dado que o processo de detecção de cenários implícitos é NP-difícil, isto é, não se sabe quando irá terminar, se estas famílias detectadas caracterizarem todas as variações dos cenários implícitos do sistema, o tratamento deles será facilitado, pois o usuário necessitará somente tratar tais famílias.

Palavras-chave: engenharia de software, dependabilidade, confiabilidade, cenários implícitos, sistemas concorrentes

Abstract

When a concurrent system is modeled on a scenario-basis, a often observed problem is the presence of unexpected scenarios during runtime, which are called implied scenarios and occur because of an erroneous modeling. The treatment process of implied scenarios is very important for a system to have correct behavior, however it demands time and effort from the user. Therefore, this work aims to ease the detection phase, so that the user can treat them in a potentially faster and simpler way. This is achieved by using the concept of families of implied scenarios, which are groups of similar scenarios that are likely to have the same cause. Given that detecting implied scenarios is NP-hard, that is, it is not known when it will end, if detected families characterize all variations of implied scenarios of the system, the treatment process will be easier, as the user will only need to treat said families.

Keywords: software engineering, dependability, reliability, implied scenarios, concurrent systems

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

A sociedade cotidianamente se beneficia com computadores e das enormes possibilidades que estes trazem consigo. Eles nos permitem realizar diversas tarefas que demandavam muito tempo anteriormente, mas que agora são executadas por software de uma maneira rápida. Com este crescimento na utilização de computadores e dos diferentes softwares disponíveis, é de vital importância que eles sejam confiáveis e para tal análise são necessários os conceitos de dependabilidade [1], que são utilizados justamente para verificar quão confiável é um sistema.

Uma maneira de garantir isso é pela correta modelagem do software. Quando uma modelagem a base de cenários é utilizada para um sistema concorrente, durante a sua execução são muitas vezes observados cenários que não haviam sido modelados, logo não estavam previstos. Estes cenários são chamados cenários implícitos [2] e representam um erro recorrente em sistemas concorrentes. Consequentemente, a detecção e análise de tais cenários se torna extremamente importante para a corretude do software e para que este também tenha um alto grau de confiabilidade.

Trabalhos anteriores, como por exemplo os de Lima [3], Uchitel et al. [6] e Reis [7], abordam esta situação. Uchitel et al. propôs em [6] um plugin para a ferramenta LTSA [8], chamado LTSA-MSC, que permitia a detecção de cenários implícitos, porém, era possível somente detectar e tratar um cenário de cada vez, o que demandava muito tempo do usuário.

Com isto em vista, no trabalho de Reis [7] a noção de famílias de cenários implícitos foi introduzida afim de caracterizar os grupos de cenários implícitos por semelhança de comportamento, resultando em uma otimização no processo de análise dos cenários. Desta forma, o tratamento de cenários implícitos poderia ser feito não apenas individualmente para cada cenário (como proposto originalmente por Uchitel et al. [6]), mas para toda a família à qual o cenário faz parte.

Lima [3] por sua vez propôs uma primeira implementação para a ideia de famílias

de cenários implícitos. Além disso, Lima [3] também modificou o processo de detecção de cenários implícitos do LTSA-MSC original, de modo que eles fossem encontrados de uma maneira iterativa, isto é, uma busca exaustiva era executada com intuito de coletar múltiplos cenários implícitos que seriam analisados de uma só vez, utilizando o conceito de famílias.

Entretanto, a heurística do processo de extração dessas famílias, proposta por Lima [3], era bastante limitada, uma vez que se baseava somente nas duas últimas mensagens trocadas em cada cenário encontrado. Essa heurística dificilmente escalaria para situações onde seria necessário considerar mais do que duas mensagens. Como nos casos onde houvesse a necessidade de analisar o comportamento global do cenário implícito como um todo (para fins de caracterização de falhas) e não apenas de suas duas últimas mensagens, a proposta de Lima [3] também não seria aplicável.

1.1 Problema

O processo de tratamento de cenários implícitos quando tratados um a um é caro, pois não é possível saber quantos cenários ao todo serão detectados. Ao definir famílias de cenários implícitos, é esperado que a partir de um certo ponto, todos os novos cenários implícitos detectados façam parte de uma família já definida e, consequentemente, tenha a mesma causa dos outros cenários que a compõem. Assim, o usuário conseguiria diminuir o custo do tratamento, pois poderia tratar somente os cenários implícitos que melhor representam suas respectivas famílias.

Porém, a primeira proposta para criação de família de cenários implícitos, feita por Lima [3], se mostrou bastante limitada por apenas considerar as últimas mensagens trocadas no cenário, ignorando informações possivelmente importantes sobre quão relacionadas são outros cenários implícitos.

Com isto em vista, é necessário uma proposta que leve em conta mais informações sobre os cenários implícitos detectados, enquanto também permite definir relações entre as diversas famílias de cenários implícitos, possibilitando ao usuário perceber como estes cenários implícitos se relacionam, para que o processo de tratamento seja mais eficaz.

1.2 Solução Proposta

Para tentar resolver o problema acima, algumas técnicas foram propostas neste trabalho. Primeiramente, é utilizado o processo de detecção de cenários implícitos utilizado em Lima [3], pois já são detectados automaticamente vários cenários implícitos de forma iterativa, tirando a necessidade de interação constante com o usuário.

Tendo este conjunto de cenários implícitos, vamos realizar dois agrupamentos sobre ele caso o usuário deseje. Primeiramente eles serão clusterizados [4] de forma não supervisionada, com base nas mensagens trocadas [8]. Após ser utilizada esta técnida de clusterização, agora já utilizando um conjunto de famílias de cenários implícitos, um agrupamento com supervisão do usuário é realizado, para que o conhecimento de domínio do usuário seja benéfico ao processo.

Estes resultados obtidos são, então, utilizados para gerar um relatório no qual o usuário pode ver as famílias detectadas. Conceitos de clusterização são utilizados na geração do relatório, para que as informações apresentadas ao usuário estejam melhor dispersas. Com o tratamento do cenário implícito que melhor caracteriza sua família, é possível que os outros cenários dessa mesma família sejam tratados. Desta forma, o esforço requerido na parte de tratamento seria diminuído, como já foi feito na parte de detecção.

1.3 Trabalhos Relacionados

Uchitel et al. [6] propôs um plugin, para a ferramenta LTSA [8], que permite ao usuário detectar a presença de cenários implícitos [2] em um sistema modelado a base de cenários. Este plugin é o LTSA-MSC. Para utilizá-lo, o usuário necessita modelar o sistema desejado a base cenários e representar tais cenários em diagramas de trocas de mensagens (Message Sequence Charts). Com isto, a ferramenta detectaria um cenário implícito no sistema e o mostraria ao usuário, que deveria então classificá-lo como positivo ou negativo. Assim o usuário poderia detectar os cenários implícitos, porém de uma forma demorada visto que ele deve classificar os cenários encontrados um a um.

Reis [7] introduziu a noção de famílias de cenários implícitos. A ideia é que diferentes cenários implícitos, que tenham sido causados pelas mesmas trocas de mensagens, podem ser tratados de uma mesma maneira. Desta forma, o usuário teria um esforço reduzido, pois somente necessitaria tratar um cenário implícito de cada família detectada. Além disso, como o processo de detecção de cenários implícitos é NP-difícil, não é possível saber quando o processo terminará, ou seja, quantos cenários serão detectados. Porém, se as famílias de cenários implícitos detectadas caracterizarem o sistema por inteiro, em um determinado ponto todos os novos cenários implícitos detectados fazem parte de uma família já caracterizada, desta forma o usuário conseguiria tratar todos os cenários implícitos do sistema.

Lima [3] pôs as ideias propostas por Reis [7] em prática. A ferramenta LTSA-MSC é modificada para que 100 cenários implícitos sejam detectados de uma só vez, não necessitando interagir com o usuário após cada detecção e sem classificá-los como positivos ou negativos. Estes cenários são então agrupados em famílias com base nas duas últimas

mensagens trocadas. Assim, os resultados mostrados ao usuário diminuíram drasticamente, pois apenas o primeiro cenário implícito de cada família é mostrado ao usuário. Com menos resultados a serem analisados e se as famílias estiverem bem definidas, o usuário conseguirá tratar todos os 100 cenários implícitos detectados de uma maneira mais rápida.

1.4 Organização do Trabalho

No Capítulo 2 estão dispostas as informações necessárias para o entendimento do trabalho e dos conceitos utilizados. Serão definidos os conceitos de Dependabilidade, Cenários e Clusterização.

No Capítulo 3 são apresentadas as ideias que foram inicialmente consideradas, assim como as razões para terem sido descartadas.

No Capítulo 4 a metodologia utilizada para a implementação da solução proposta é descrita de maneira detalhada.

No Capítulo 5 os resultados obtidos neste trabalho são apresentados. É mostrado como a execução da solução ocorre e também as saídas geradas por ela. Além disso, são discutidos os resultados para os sistemas utilizados como exemplo, o Sistema de caldeira [2] e Cruiser [9].

Por fim, no Capítulo 6, a conclusão sobre os resultados obtidos é apresentada, assim como os possíveis trabalhos futuros.

Também é importante ressaltar que os capítulos de 2 a 5 estão redigidos em inglês, ao contrário dos outros capítulos que são apresentados em português. Esta decisão foi tomada afim de preparar este trabalho para publicações internacionais futuramente.

Capítulo 2

Background

In this chapter, definitions and technical concepts that are used throughout this work will be laid out and explained with examples, where applicable.

2.1 Dependability

A seminal taxonomy on dependability was defined by Avizienis et al. in [1], and it is the base which will be used on this subsection and throughout this work.

The original definition of dependability involves the concept of security, which won't be used in this work. Therefore, the usual definition will be followed.

The dependability of a system must be compatible with the level of trust that it is relied upon it. With this in mind, the dependence of a system (A) in another system (B), represents how much the dependability of A is affected by the dependability of B.

Dependability is a concept that consists of five attributes, and those are:

availability - the system is available to provide correct service;

reliability - the system delivers the correct service;

safety - the system doesn't show catastrophic consequences;

integrity - the system is free of improper changes;

maintainability - the system can be changed to be kept running or have new functions added to it.

2.1.1 Threats to Dependability: Failures, Errors and Faults

In this subsection, concepts that might affect a given system's dependability are presented. These concepts may result in a behavior different than expected. They will also be used to define other concepts further on.

A system's **correct service** is delivered when this service implements the system's function, in other words, the behavior observed by the user was the expected system behavior for that specific system function.

A **service failure**, or simply **failure** (see 2.1.2) in this context, is an event that happens when the delivered service deviates from the correct one. It occurs because it doesn't comply with the system's specification, or because that specification doesn't portrait the system's function in a acceptable way. A failure is a transition from correct service to incorrect.

When a system's delivered service deviates from the correct service, we have an **error**. An error is a part of the system's states where it can have a deviation from correct service, resulting in a failure. Notice that an error is simply the possibility of a failure occurring, that is, a system can contain errors even though failures aren't presented.

An error's cause, hypothetical or real, is called a **fault**. It can be internal or external on a system, where this shows on which part of the system's total states it is. It can also be active or dormant, indicating whether or not an error is caused.

2.1.2 Failures

Failures, more specifically service failures, can be categorized following a taxonomy that's already been defined. This taxonomy is composed of four characteristics, which are detailed below:

1. Domain

   Indicates on which context the failure occurred, showing what wasn't expected. Failures are categorized according to *time* and *content*. A failure can have:

   (a) Correct content but wrong timing:

   This way, a failure can be either *early* or *late*.

   (b) Correct timing but wrong content:

   In this case it is a *content* failure.

   (c) Both content and timing are wrong:

   We can have a failure that causes the system to *halt* or deliver *erratic service*.

2. Detectability

   Indicates if the user is alerted of the failure's presence or not. When the system perceives a failure, it can choose to alert the user of its presence. If the user is alerted, the failures is *signaled*. Otherwise, it is an *unsignaled* failure.

3. Consistency

   Analyses if the failure is observed consistently by all users, or if it's perceived in an inconsistent manner, where users receive different types of incorrect service, or some of them could even get the correct one. This failure is therefore said to be *consistent* or *inconsistent*.

4. Consequence

   Regarding a failure's consequences, is necessary to define, in a qualitative manner, levels of the consequences that incorrect services would result. Two extremes are defined:

   small : the cost of a failure is similar to the benefits of correct service;

   catastrophic : the cost of a failure is extremely higher (or even incomparable) to the benefits of having correct service.

   Intermediary levels between those limits can be defined.

A breakdown of this taxonomy can be seen in Figure 2.1.

2.2 Scenarios

A scenario is a description of a system's action. It describes what the user expects from the system when interacting with it. We can model entire systems based solely on scenarios that it needs to execute, this is called a positive scenario-based model.

As an example, let's look at a sample system, the *Boiler System*. The *Boiler System* is a system that controls the temperature inside a boiler, according to the measured pressure by its sensor. It has the following components:

Actuator variates the temperature inside the boiler;

Control tells the actuator to act according to the last pressure measured;

Database stores the measured pressures;

Sensor measures the pressure inside the boiler.

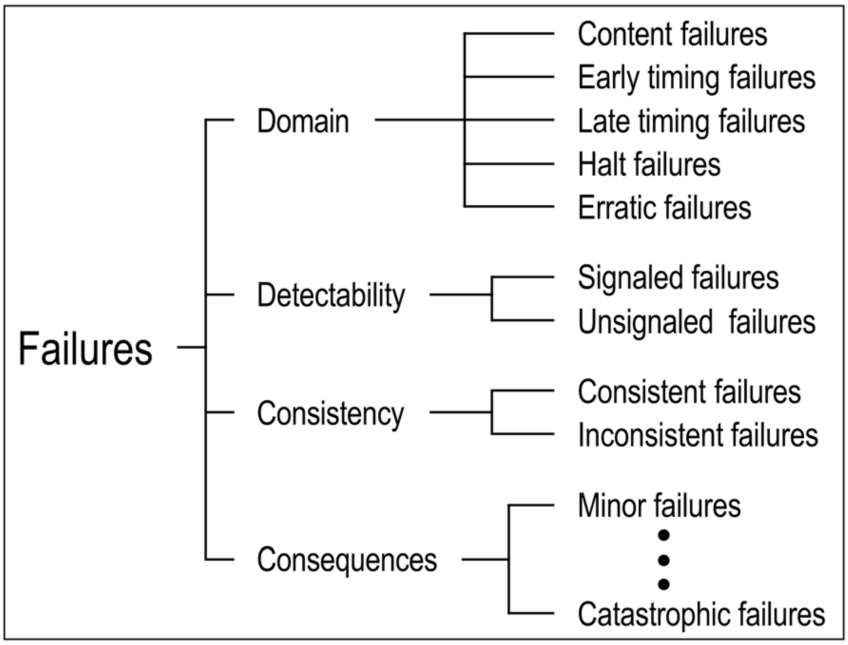


Figura 2.1: Taxonomy of a failure, taken from [1].

This system performs four scenarios, and these are all accomplished by interactions between the components. Below these scenarios are shown and the interactions between the system's components are described:

Turning on : *Control* tells *Sensor* to start monitoring the pressure;

Turning off : *Control* tells *Sensor* to stop monitoring the pressure;

Registering pressure : *Sensor* sends *Database* the current pressure so it's stored and can be queried later on;

Adjusting temperature : *Control* queries *Database* for the latest pressure and tells *Actuator* to alter the boiler's temperature accordingly.

By implementing these four scenarios, we have a system that was modeled on a scenario-based way.

2.2.1 Message Sequence Charts

A message sequence chart (MSC) is a simple way to illustrate a system's action, that is, a scenario. It explicitly shows the interactions between components, by showing each one of those as a message sent from one component to another. We can use MSCs to show the *Boiler System*'s scenarios described above as in Figure 2.2. As it can be seen, it's a really good way to exemplify the interactions that happen for a scenario be achieved.

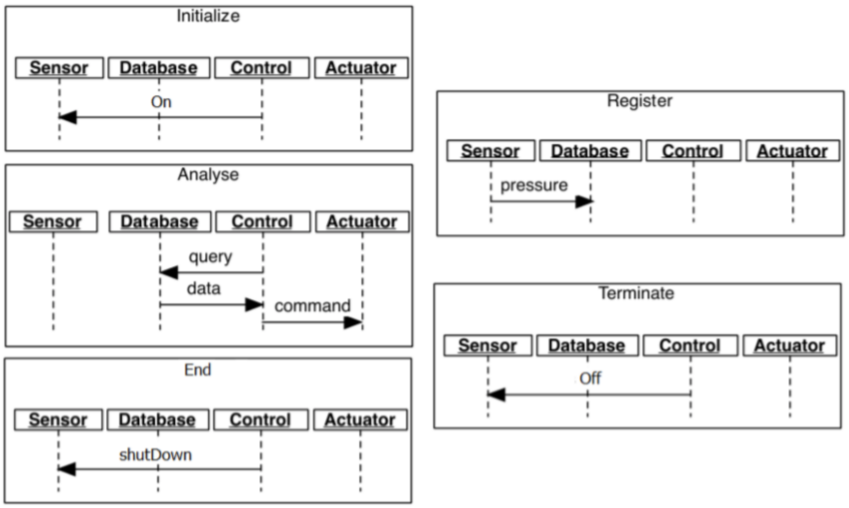


Figura 2.2: bMSCs representing the scenarios of the Boiler System. Taken from [2].

The MSCs shown in Figure 2.2 are said to be bMSCs, which stands for basic message sequence charts. A bMSC shows the interaction between instances. In the *Initialize* scenario for instance, the *Control* instance is sending the message *on* to the *Sensor* instance. A bMSC doesn't necessarily convey an order to the messages, however, in our case, there is only one bMSC with more than one message, therefore, the other ones have obviously only one possible order of execution, which is sending their only message.

For the *Analyse* bMSC however, there are three messages and that could lead to more possible orders of execution. In this case, it is important to note that an instance of a bMSC has to follow the order on which the messages are sent or received. For instance, the *Database* instance can only send the *data* message after the *query* message is received. Hence, this scenario only has one possible order as well.

An extension of bMSCs are hMSCs, which stands for high-level MSCs. The simplest hMSC possible would equal to a bMSC. However, it is possible to have hMSCs inside a hMSC, where the inner ones are used to abstract more complex messages sequences, and the outer one is called a compound hMSC. These compound hMSCs can show the order of execution of hMSCs, that is, it shows when which scenarios should happen, in a way that it is visually easily understood. These can be used to show the possible paths of execution of a system. As an example, in Figure 2.3 the hMSC for the *Boiler System* is shown. It is possible to observe that the scenarios are ordered in a way that the system delivers correct service.

2.2.2 Implied Scenarios

An implied scenario is a scenario that wasn't included in the system's modeling, but it occurs when the system runs. It is a result from implementing actions that are global to

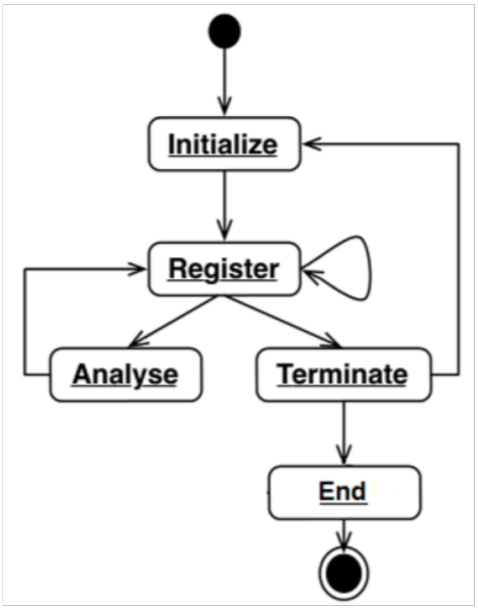


Figura 2.3: hMSC of the Boiler System. Taken from [2].

the system, in a local level to the component that executes it. Because of this implementation, a component might not have enough information locally to decide whether or not the action should be prevented, therefore it is always performed.

An implied scenario can be classified as positive or negative. A positive implied scenario is one that although it wasn't included in the system's specification and its behavior wasn't expected, it has a desired behavior. In this case, the system's specification is usually extended with this new scenario. A negative implied scenario is a scenario that wasn't expected and its observed behavior is harmful to the system's execution, that is, the system is not performing the correct service, or in other words, a failure. In this work, we'll not make this distinction on implied scenarios, all of them will be treated as a failure.

Because of the nature of concurrent systems, implied scenarios may not happen in every system run, as messages aren't synchronized and traces of execution (order of the messages on the MSC) could be different, even though the same course of action is sought.

As an example, in Figure 2.4 an implied scenario in the *Boiler System* is presented. The unexpected part, that is, the cause of the implied scenario, is that *Control* tries to execute the *Adjusting temperature* scenario before a pressure from the current system run is registered, therefore, it will tell the *Actuator* to variate the temperature according to a pressure that doesn't represent the system's current state.

We can use the failure's taxonomy laid out in 2.1.2 to classify implied scenarios. According to those definitions, it is classified as follows on the different characteristics:

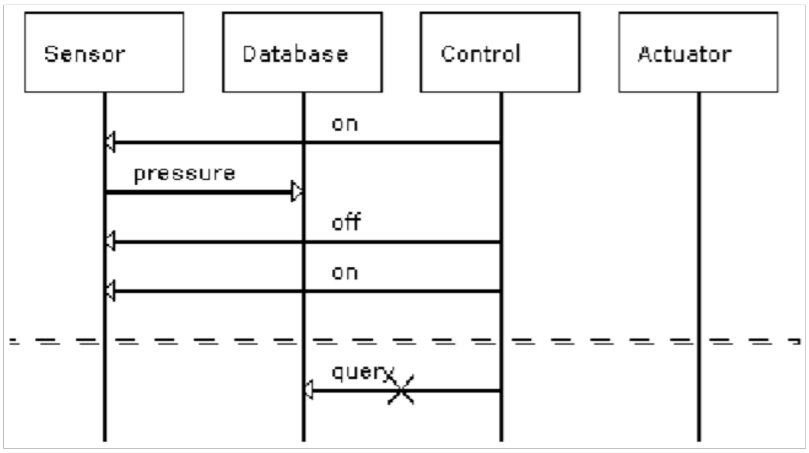


Figura 2.4: An implied scenario from the boiler system. Taken from [3].

- Domain: depends on the case. Different implied scenarios from a same system can have different domain classifications;
- Detectability: it is detectable, as the user observes unexpected behavior from the system;
- Consistency: it's *inconsistent*, because, given the non deterministic nature of concurrent systems, not all users will observe the same trace of execution;
- Consequence: depends on the system.

The tool LTSA-MSC [6] allows users to detect implied scenarios in an automated way. It is a plugin for the tool LTSA [8], which stands for Labelled Transition System Analyser. In [3], the LTSA-MSC functionality was extended, as in the original tool, the user had to provide feedback for each implied scenario detected. Instead, the tool now finds the first 100 implied scenarios in the system's model provided.

2.2.2.1 Families of Implied Scenarios

This concept was first introduced in Reis [7] as a way to reduce the time necessary to treat implied scenarios. It is a way to group distinct implied scenarios so that similar ones don't have to be treated more than once. It was defined in [7] as follows:

Definition 1. *Given a trace of execution which characterizes an exchange of messages in a negative implied scenario, a family of an implied scenarios is composed by all finite traces that contain the message exchange which characterizes the implied scenario.*

The idea behind it is that we can group implied scenarios into families, because they have the same cause. Hence, theoretically, it'd be possible to treat all scenarios that are part of a family by treating only one of them.

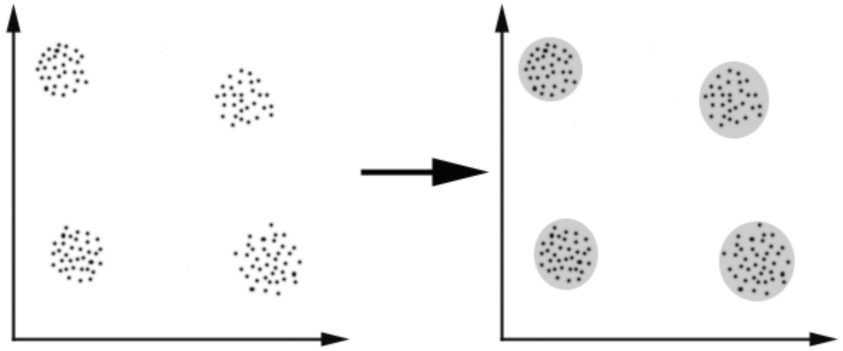


Figura 2.5: Example of a clustering technique, taken from [4].

The first try to implement it was done by Lima [3], who determined the families by grouping implied scenarios whose last two messages were the same. In this work a new approach to this problem is introduced and a methodology is proposed.

2.3 Clustering

Clustering is a data-mining technique used to group datapoints in a dataset. There are various algorithms to perform this technique, such as K-means [10] and Expectation Maximization [11], which will be used on the proposed methodology.

It is a method to group elements in a unsupervised way, and nothing more. After obtaining the separate groups, the user still has to analyze the results and figure out why those elements were clustered together. A simple definition of the clustering process is given in Matteucci [4]: "the process of organizing objects into groups whose members are similar in some way".

A good clustering result is such that the cluster's elements are very similar to each other, and very dissimilar to other clusters' elements. This way a good separation between clusters is observed and the elements within a cluster are clearly similar.

In Figure 2.5 a simple clustering process is shown, where on the left side of the image the elements are all separate, and on the right side, after clustering, four clusters can be clearly seen.

This section's remaining subsections will go into the specific clustering characteristics (i.e. metrics) and methods (i.e. K-means, Expectation Maximization, and Hierarchical) that were used on the proposed methodology of this work.

2.3.1 Metrics

Here the metrics that are used to compare different clusters, in this project, are defined. They can either indicate a characteristic of how well it describes its own elements (2.3.1.1)

or how distinct it is to another cluster (2.3.1.2).

2.3.1.1 Intra-Cluster Distance

It is a metric used to indicate how similar the elements of a cluster are. It is calculated as the average of the distance from each element to the cluster's centroid (i.e. the central point of the cluster). Since a cluster should include similar elements, it's desirable that this distance is low, as that indicates that the elements are very similar, that is, not very distant from each other.

The formula is as follows:

$$IntraCD(cluster) = \frac{\sum_{i=1}^{n} dist(e_i, centroid)}{n}$$

where:

IntraCD is the intra-cluster distance;

cluster is the cluster being used;

dist is a distance function (e.g. euclidean distance);

e_i is the i-th element of the cluster;

centroid is the center element of the cluster (i.e. the mean of all cluster's elements);

n is the number of elements of the cluster.

It is also possible to use this metric to calculate how good the clustering results are. To accomplish this, it is necessary to apply the above formula to all the clusters obtained, sum them, and then average those results. Again, it is desirable that this result is low.

Therefore, the formula would be:

$$AvgIntraCD = \frac{\sum_{i=1}^{n} IntraCD(cluster_i)}{n}$$

where:

AvgIntraCD is the average intra-cluster distance for all the clusters;

n is the number of clusters in the result set;

IntraCD is the function defined above.

2.3.1.2 Inter-Cluster Distance

The inter-cluster distance is used to indicate how far clusters are apart. It is calculated as the distance between two clusters' centroids. As we want clusters to be well separated, it is desirable that this distance has a high value.

The formula is as follow:

$$InterCD(A, B) = dist(centroid_A, centroid_B)$$

where:

InterCD is the inter-cluster distance;

A, B are the clusters being used;

dist is a distance function (e.g. euclidean distance);

$centroid_A$ is the centroid of cluster A;

$centroid_B$ is the centroid of cluster B.

Just like the intra-cluster distance, it can also be used to characterize the whole result set. In this case, it is necessary to calculate the inter-cluster distance between a cluster and every other cluster, sum those, and then average them. Therefore, we have the following formula:

$$AvgInterCD = \frac{\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} InterCD(cluter_i, cluter_j)}{n(n-1)}$$

where:

AvgInterCD is the average inter-cluster distance for the result set;

n is the number of clusters in the result set;

$cluster_i$, $cluster_j$ are clusters from the result set.

2.3.1.3 Beta-CV

The beta-cv is a coefficient that can show how many clusters would be a good fit for the dataset. It is calculated as the proportion of the average intra-cluster distance by the average inter-cluster distance.

The ideal value would be zero, where every cluster has an intra-cluster distance of zero, showing that all the elements are equal, and a high inter-cluster distance, which shows that the clusters are well defined.

However, it is not desirable to increase the number of clusters until this value is reached, as more clusters could be created, while no new knowledge is gained. The idea is that at one point the changes in inter-cluster and intra-cluster distance will be proportional, which would make the value of the beta-cv constant.

Therefore, to find the ideal number of clusters for a given dataset, one should plot the calculated beta-cv for each number of clusters, and at the point where this values stabilizes, that is, where it changes very little, that is the point where the best ratio of intra-cluster distance to inter-cluster distance is found, and consequently, the best number of clusters is found.

2.3.2 K-means

K-means is one of the most popular clustering algorithms [10]. It is a simple unsupervised method to analyze data [4]. It uses centroids to define the clusters' elements, and then uses those elements to define the clusters' centroids.

Before the algorithm starts it is necessary to define the number of clusters to be found, and the initial centroids to those clusters. Usually, these starting centroids are randomized.

After the initial centroids have been chosen, they are used to assign each element to a cluster, and after all elements have been assigned, new centroids are calculated. Then, the process is repeated until there is no change to the centroids or some other stop condition is reached (e.g. maximum number of iterations). This algorithm is shown in Algorithm 1.

Algorithm 1 K-means

```
1: procedure K-MEANS(dataset, n)          ▷ n is the number of clusters
2:     centroids[n] ← randomPositions      ▷ random starting centroids
3:     while centroids ≠ oldCentroids do   ▷ or another stop condition
4:         oldCentroids ← centroids
5:         assign each element in dataset to cluster with closest centroid
6:         updates centroids
```

When the algorithm stops, we have the final clusters defined. Given the random nature of this algorithm, sometime it's necessary to execute it through the dataset more than once, as it might not have calculated the best results with the random starting centroids.

2.3.3 Expectation-Maximization

The Expectation-Maximization (EM) algorithm is a clustering algorithm proposed by Dempster et al. in [11]. The EM algorithm is an iterative algorithm, that is, just like

K-means (2.3.2), it consists of repeating the same steps until the result obtained is satisfactory.

Each iteration of the EM algorithm consists of two processes: E-step and M-step [12], where E and M stand for Expectation and Maximization, respectively. During the E-step, hidden data (e.g. clusters' centroids) is estimated by using the observed data and current parameters. After, during the M-step, the likelihood of those estimations is maximized, by assuming that the estimated data was observed and updating the model's parameters (e.g. cluster assignments). In this way, it's possible to see that K-means and EM are very similar, and, in fact, according to Piech [10], K-means is simply a particular application of EM.

However, implementations for K-means and EM differ in a vital point in WEKA [13]. Where EM can find out the appropriate number of clusters for a given dataset, K-means is not capable of that, as needs to know how many clusters are there beforehand. Because of this, and given that the methodology proposed in this work intends to analyze multiple systems, a predefined number of clusters would not suffice and instead of looking other ways to calculate the correct number, EM was the chosen technique.

The likelihood function indicates the probability of each element belonging to each cluster [13], instead of assigning an element to a single cluster (like K-means). The metric used to check an improve in the results is the log likelihood, which can be described by the formula below [12]:

$$L(\theta) = \ln P(X|\theta)$$

The EM algorithm is an interactive procedure for maximizing $\mathrm{L}(\theta)$ [12], which will be accomplished by maximizing the likelihood ($P(X|\theta)$), given the linear nature of $\ln(x)$.

The algorithm stops when there is not an increase of log likelihood during the maximization process, indicating that an optimal result has been reached for that run, that is, the likelihood function used is at its maximum value. However, it's not certain that this is a global maximum, and, therefore, it is helpful to run the algorithm multiple times, variating the starting parameters [14].

2.3.4 Hierarchical Clustering

Hierarchical clustering is an agglomerative clustering method [4], that is, it's a method where all the elements start as a cluster of 1 element, and these clusters are then grouped one by one, until we have only one cluster that consists of all elements.

This algorithm has the benefit of being able to output how close the elements are. Because as they are being grouped one by one, it's possible to see the order they are grouped and, therefore, which ones are more similar.

Given a dataset on N elements, the first step is to have N clusters, where each cluster has only one element. Then, a matrix (A) of size $N * N$ is built, where each element A_{ij} is the similarity between clusters i and j. This matrix is called, Similarity Matrix (SM). Similarity can be calculated in various ways (e.g. euclidean distance between centroids).

With the SM built, the most similar (i.e. the least far apart) pair is found and merged into one cluster, the new similarities between this new cluster and the remaining ones is calculated and the SM updated. This process is repeated until there is only one cluster left. This algorithm can be seen in Algorithm 2.

Algorithm 2 Hierarchical Clustering

```
1: procedure HIERARCHICAL(dataset)
2:     assign each element to its own cluster
3:     update SM with current clusters
4:     find and merge the most similar pair of clusters
5:     if n > 1 then go to 3                    ▷ n is the number of clusters
```

Let's take a look at an example. Given a dataset of four elements, *A, B, C, and D*, with the respective coordinates: [0,0], [0,2], [3,4], [5,6]. And using the minimum euclidean distance between elements (this is known as single linkage) to calculate the similarity matrix. The starting SM is shown in Table 2.1(a).

Tabela 2.1: Example's Similarity Matrices

(a) Start SM

	A	B	C	D
A	0,0	2,0	5,0	7,8
B	2,0	0,0	3,6	6,4
C	5,0	3,6	0,0	2,8
D	7,8	6,4	2,8	0,0

(b) SM after first merge

	{A, B}	C	D
{A,B}	0,0	3,6	6,4
C	3,6	0,0	2,8
D	6,4	2,8	0,0

(c) SM after second merge

	{A, B}	{C, D}
{A, B}	0,0	3,6
{C, D}	3,6	0,0

The two most similar clusters are A and B, as they have a distance of 2,0. They are then grouped and the new SM is calculated, with only three clusters and the minimum distance kept between the elements of the new cluster and the other clusters. This updated SM can is shown in Table 2.1(b). Then this process is repeated twice, after the first time the updated SM is the one seen in Table 2.1(c) and after the second one there is only one cluster left, which shows that the process of clustering is complete.

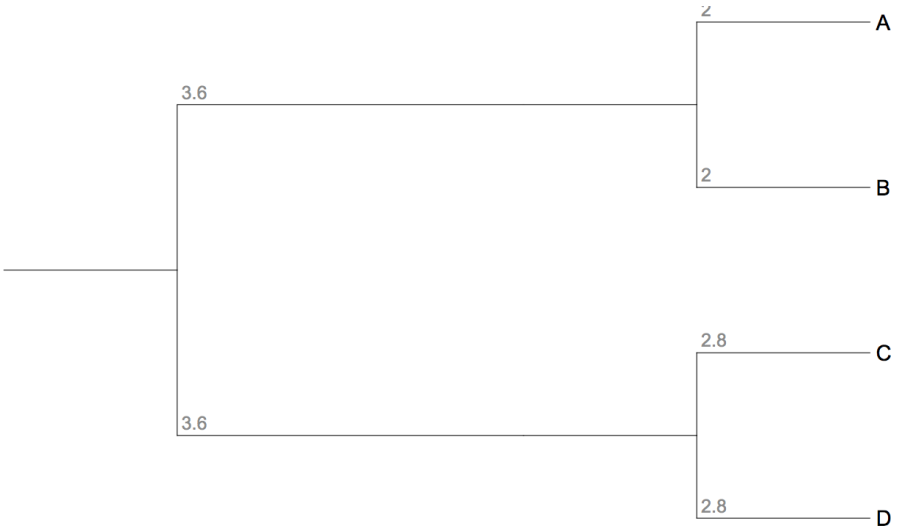


Figura 2.6: Example of a dendrogram that shows a hierarchical clustering. Drawn with [5].

When the process of grouping clusters is finished, a dendrogram can be drawn to show how similar the elements are, and the dendrogram for the given example is seen in Figure 2.6.

Capítulo 3

Ideias Iniciais

In this chapter, the initial proposals are presented. Although they were rejected, they played an important role into creating the proposed methodology, as they showed what could be done and what could not. Therefore, it is important to lay them out, so the reasoning behind the proposed idea is understood.

This work is based on the ideas by Reis [7], who first introduced the concept of families of implied scenarios, and Lima [3], which made the first try to implement that concept.

The implementation by Lima [3] grouped implied scenarios by taking into account the last two messages on the scenarios' traces, that is, the order of messages that were passed. It uses a solution called LTSA (Labelled Transition System Analyser) [8], which works in a plugin-based manner. The solution by Lima [3] was implemented by extending LTSA's plugin to analyze MSCs (LTSA-MSC [6]), and found implied scenarios in an automated way while grouping the ones that had the same messages in the last two positions of the trace. However, as it is shown in Section 3.1, it might not be the best approach to the problem, hence the proposal of a new methodology.

3.1 Increasing the Number of Relevant Messages

Through analyzing the results presented by Lima [3], specially when looking at the implied scenarios families identified on the *Cruise System* study case, there are families that share many messages, but not the last two ones. We extended the analysis to the five last messages, and did a breakdown of the percentage of the messages for each family (still using the grouping method in [3]).

An example as how two distinct families are, in fact, very similar is shown in Table 3.1. In that table, a breakdown of the last 5 messages (with the last one being n) is presented, with a percentage included, and this percentage represents how many elements

out of the entire family had that message in that position. It was expected that the last two messages showed 100%, as that's the criterion that makes them a family.

However, as it is shown, two distinct families (*Case 1*) were found to share 4 out of the 5 last messages, with the only exception being the very last one. Even more, it is possible to see that both family's core, that is, the last two messages (as defined in [3]), indicates that the user is turning the *Cruiser System* on, and right after it changing the speed manually, either by accelerating or breaking. Also, notice that although families #6 and #9 (*Case 2*) are classified as distinct families, their core is made out of the same two messages, but are not in the same order.

Tabela 3.1: Breakdown of last 5 messages with percentages

	Family	Position				
		n-4	n-3	n-2	n-1	n
Case 1	#4	on (100%)	recordSpeed (100%)	speed (100%)	enableControl (100%)	brake (100%)
	#5	on (100%)	recordSpeed (100%)	speed (100%)	enableControl (100%)	accelerator (100%)
Case 2	#6	on (45%)	speed (54%)	enableControl (81%)	speed (100%)	brake (100%)
	#9	on (100%)	recordSpeed (100%)	enableControl (100%)	brake (100%)	speed (100%)

The first idea that came up after analyzing these results was to create a metric to take into account more than two messages, as that might suffice to remedy *Case 1*, however, there wouldn't be a way to validate this metric. Therefore, given the non-deterministic nature of concurrent systems, and assuming that these messages are asynchronous (which is a possible explanation for *Case 2*), the next proposal was made (3.2).

3.2 Using K-means

After the conclusions made in 3.1, a new approach was made by calculating the frequency of messages in each implied scenario's trace, and then use these results in WEKA [13] to cluster them with the K-means algorithm.

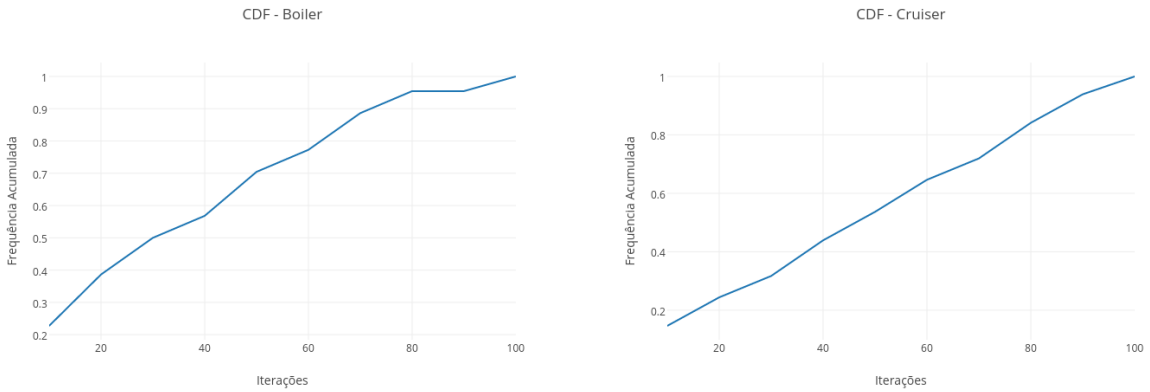
Given the randomness of K-means, tests were run to predict how many times the algorithm would have to be ran for the optimal result. These tests were done by following the steps below:

1. Detect n (provided by user) traces of implied scenarios using LTSA-MSC;
2. Convert the ordered traces to a CSV with the messages' frequency;
3. Set number of clusters to 1;
4. Run 100 times the K-means algorithm and store which run it presented best results;
5. Run Step #4 100 times (totaling 10,000 runs of the algorithm);

6. Increase the number of clusters by 1;

7. If number of clusters is less then 100 then go to Step #4.

With the results from the above test, applied to both *Boiler* and *Cruise* systems, the cumulative distribution function (CDF) of which run on Step #4 had the best result suggests that if more runs were performed, better results would be obtained. These plots are shown in Figure 3.1.



(a) CDF for the Boiler System.

(b) CDF for the Cruiser System.

Figura 3.1: CDFs for the two example systems.

Besides that, to figure out the best number of clusters for the dataset would require to check when the proportion of Intra-Cluster and Inter-Cluster distances stabilizes. However, it is not a simple task to find the point where it starts to stabilize (that is the stopping point). Lastly, this alone wouldn't solve the problem saw in *Case 1* Table 3.1, where messages are different but represent the same idea (changing from automatic speed control to manual), as this would require a system's domain. This takes us to the proposed methodology.

Capítulo 4

Metodologia Proposta

In this chapter the methodology proposed will be laid out. First, in Section 4.1, a brief overview of the whole process will be laid out, and in the subsections that follows, each step of the process will have a more in-depth explanation.

4.1 Overview

This methodology tries to overcome the difficulties observed on the initial proposals and by Lima [3]. The iterative detection process of Lima [3] is kept with a slight modification, as the user has now the option to choose how many implied scenarios should be collected.

With the desired number of implied scenarios already detected, an unsupervised clustering is made using the EM algorithm [11]. This is expected to mitigate the problem with K-means where the number of clusters must be determined beforehand. In order to treat the problem observed in the K-means, where we saw that more iterations would generate better results, the EM algorithm is run 1000 times.

Finally, in order to solve the problem of distinct detected families of implied scenarios, that are however very similar when the system knowledge is taken into account, a supervised clustering is done to benefit of the knowledge of the user.

With all this done, the results are then exported in multiple formats, so the user has various options to analyze them and opt for the best way of treatment.

The methodology steps are as follows:

1. get hMSC model into LTSA-MSC;
2. asks the user how many (n) implied scenarios should be collected;
3. collect implied scenarios based on n runs of the implied scenario detection procedure in LTSA-MSC;

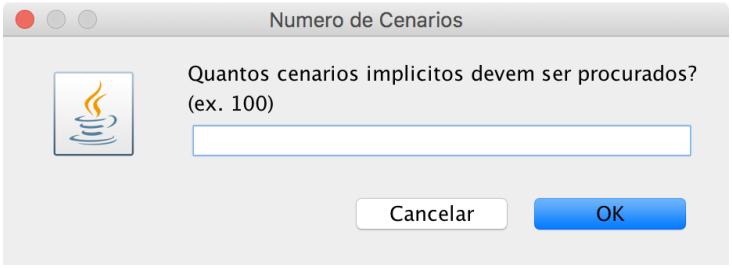


Figura 4.1: User interaction to define the number of implied scenarios to be collected.

4. convert the trace of a scenario into frequencies of messages;
5. unsupervised clustering of those scenarios into families using EM;
6. supervised clustering of families based on the domain knowledge of the user;
7. export final results.

Steps 1 and 2 are similar to Lima [3], with the exception that the implied scenarios found aren't immediately put into a family, as already stated above. Another change is that the user is now asked to see how many implied scenarios he wants to look for (as seen in Figure 4.1), instead of using a fixed value (i.e. 100 implied scenarios). These two steps, alongside step 3, are specified in Section 4.2, as this step analyzes the system's modeling, so that data needed for next steps is obtained.

In step 5, implied scenarios are split into different families in an unsupervised way. A more in-depth look at this process is shown in Section 4.3. In step 6 the families detected previously will be subjected to user's supervision, and might be grouped together, which will be addressed in Section 4.4. Finally, the results are exported in order to allow the user to act upon them (Section 4.5).

4.2 Gathering Data

While finding implied scenarios in [3], the traces of implied scenarios are obtained as a series of strings. Therefore, these traces (vectors of strings) are grouped into a vector of traces, in order to be analyzed later. Two classes were created for this part, and their names and members are shown in Listings 4.1 and 4.2.

```
public class Pair implements Comparable<Pair> {
    public String message;
    //stores the message's frequence in this scenario
    public double frequence;

    //prototype of the only method in this class
```

```
    public int compareTo(Pair);
}
```

Listing 4.1: Pair: associates a message with its frequence.

```
public class TraceInfo {
    //stores once every message that appeared at least once
    List<String> messages;
    //a list of pairs for each scenario
    Vector<List<Pair>> tracesInfo;
    //stores ordered messages for each scenario
    Vector<Vector<String>> tracesMessages;

    //prototype of the constructor
    public TraceInfo(Vector<Vector<String>>);

    //prototypes for the methods
    public void print();
    public void print(String);
    public void print(String, String);
    public void getInfo();
    private List<Pair> merge(List<Pair>, List<String>);
}
```

Listing 4.2: TraceInfo: used to transform traces into pairs.

The *Pair* class is only used to associate a message with its frequence. It is used in the *TraceInfo* class. The latter is used to extract the list of messages that show up in the traces. It implements a method called *getInfo()* to convert implied scenarios' traces (these were passed to the class' constructor and then stored in *tracesMessages*) into pairs of messages and frequencies. This is explained as a pseudocode in Algorithm 3.

Algorithm 3 getInfo

```
 1: procedure GETINFO
 2:     initialize members                          ▷ messages and tracesInfo
 3:     for each trace ∈ traces do
 4:         pairs ← ∅                               ▷ it's a list of pairs, like messages
 5:         for each message ∈ trace do
 6:             if message ∉ messages then
 7:                 add message in messages
 8:             if message ∈ pairs then
 9:                 new pair of message and frequence = 1 is added to pairs
10:             else
11:                 increase frequence of the pair that contains message by 1
12:         append pairs in tracesInfo
```

After this method is run, *messages* will contain all messages that appeared at least once in any scenario, and *tracesInfo* will have pairs (message and frequence) for each scenario. These lists are then ordered, so messages show up in the same order in *messages* and in every list of pairs (i.e. each element of *tracesInfo*). Each implied scenario's list is then extended to contain every message in *messages*, with messages that were not present in the trace being added with a frequence of 0.

Finally, after all this is done, each element of *tracesInfo* will contain every message and in the same order, varying only their frequences, corresponding to each implied scenario found previously. There results are then exported, in CSV and ARFF (WEKA [13] own format) formats, as well as text file with the original traces, that is, order of messages for each scenario. An example of this conversion is shown in Figure 4.2.

```
ID: [mensagens ordenadas]

0: [on, pressure, off, on, query]
1: [on, pressure, query, data, command, off]
2: [on, pressure, pressure, off, on, query]
3: [on, pressure, pressure, query, data, command, off]
```

(a) Original traces of execution.

id	command	data	off	on	pressure	query
0	0	0	1	2	1	1
1	1	1	1	1	1	1
2	0	0	1	2	2	1
3	1	1	1	1	2	1

(b) Frequencies of messages table.

Figura 4.2: Example of the conversion done.

4.3 Unsupervised Clustering

After the previous step is done, a pop-up window (Figure 5.3) informs the user that the process of finding implied scenarios is done, and providing with the time spent for finding them, and exporting its results. Another window then pops-up (Figure 4.3), giving the user the option to either continue with the grouping process (i.e. creating families of implied scenarios).

If the user opts to continue with the process, the data that was exported before is then clustered using the EM algorithm (via WEKA [13] tool). This algorithm was chosen as it finds the best number of clusters for the dataset, a problem that was identified if K-means were to be used. However, this might find a local number of clusters and not global. In order to lessen this problem, the algorithm is run 1000 times.

Two classes were created to manage the creation of families, and their names and members are shown in Listings 4.3 and 4.4.

```
public class ScenariosCluster implements Comparable<ScenariosCluster> {
    //cluster's centroid
```

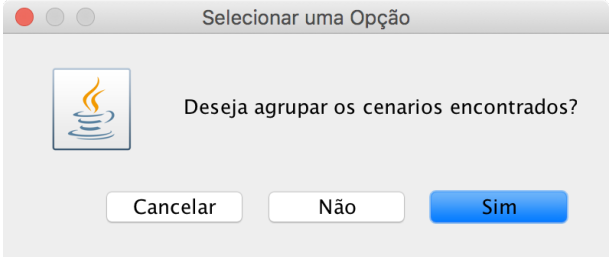


Figura 4.3: Interaction with the user to proceed with the clustering.

```
private Instance centroid;
//cluster's elements in WEKA's format
private Instances datapoints;
//indexes of the traces that are a part of this family
private Vector<Integer> index;
//traces of the scenarios that are in this family
private Vector<String> traces;
//each family has an ID that's stored here
private final int clusterNumber;

//prototypes of the constructors
public ScenariosCluster(Instances, Vector<Integer>, Vector<String>,
    boolean);
public ScenariosCluster(Instances, Vector<Integer>, Vector<String>);

//prototypes of the methods
public double calculateIntraClusterDistance();
public double calculateInterClusterDistance(ScenariosCluster);
public ScenariosCluster merge(ScenariosCluster);
public static int getTotalNumber();
public int getCount();
public Instances getDatapoints();
public Instance getCentroid();
public Vector<Integer> getIndices();
public int getClosestIndex();
public String getClosestTrace();
public Instance getClosestInstance();
public int getClusterNumber();
public String getString();
public String print();
public int compareTo(ScenariosCluster);
private void calculateCentroid();
private void orderTraces();
private void sort(Vector<Double>);
private void checkHeaders(ScenariosCluster);
```

```
}
```

Listing 4.3: ScenariosCluster: contains the information of one family.

```
public class Clusterer {
    private int repetitionNumber; //number of repetitions
    private Instances data; //data exported previously
    private Vector<ScenariosCluster> clusters; //families
    private Vector<String> traces; //detected implied scenarios
    private boolean error; //indicates an error in the process

    //prototypes of the constructors
    public Clusterer();
    public Clusterer(String);
    public Clusterer(String, String);
    public Clusterer(String, String, int);

    //prototypes of the methods
    public boolean checkError();
    public boolean clusterize();
    public void supervisedClustering();
    public void export();
    public void export(String);
    private void exportData(String);
    private ClusterDendrogram createDendrogram();
    private void exportDendrogram(String);
    private Vector<ClusterPair> pairClusters();
}
```

Listing 4.4: Clusterer: used to group implied scenarios into families.

The class *ScenariosCluster* is used to store each family's data. It's constructor receives the family's data (i.e. traces, WEKA data, and indexes for the family's implied scenarios), stores it, and then calculates its centroid, as the average of all the WEKA data it received. It also implements methods to calculate its Intra-Cluster distance, Inter-Cluster distance between itself and another family passed as parameter (calculated by the euclidean distance between centroids), and to merge two distinct families into one.

The second class implements the process of grouping implied scenarios into families. Its constructor can receive as parameters the names for the files previously exported (text and ARFF files) in 4.2, and the number of repetitions. All this parameters, however, have default values, so they aren't necessary. The process of creating implied scenario's families, that is, clustering implied scenario's, is accomplished by following Algorithm 4, which is pseudocode for the *clusterize* method.

Algorithm 4 clusterize

```
1: procedure CLUSTERIZE                                   ▷ most variables are class members
2:     remove ID from dataset                             ▷ indicates on which iteration it was found
3:     i ← 0, final ← null, max ← −∞
4:     while i < repetitionNumber do
5:         create new cluster EM
6:         run EM through dataset                                              ▷ without IDs
7:         if likelihood(EM) > max then                    ▷ check if likelihood increased
8:             final ← EM, max ← likelihood(EM)            ▷ keep current results
9:         i++
10:    clusters ← ∅                                        ▷ initialize class member
11:    assignments ← indices ← traces ← vectors of empty lists   ▷ initialize local variables
12:    for each scenario ∈ dataset do
13:        i ← assigned family by EM to scenario
14:        add scenario's datapoint to assignments[i]
15:        add scenario's trace to traces[i]
16:        add scenario's ID to indices[i]
17:    for each cluster ∈ EM do
18:        add to clusters new ScenariosCluster instance with assignments, traces and indices for current cluster
```

The first thing done is remove the IDs of implied scenarios, as they only indicate on which iteration each scenario was found (from 1 to n) and do not help the clustering process. The EM algorithm is applied multiple times (defined by *repetitionNumber*, which was set as 1000), as suggested by Do et al. [14] because it might have calculated a local maximum for likelihood. After repeating it the amount of times needed, the EM result with higher likelihood among the ones with most probable number of clusters is kept, as it presents the best result found by EM algorithm. Each scenario is then put into it's correct family's lists of attributes (i.e. indices, traces, assignment), and one *ScenariosCluster* instance is created for each family found by EM, with its respective attributes.

After this process is finished, a pop-up window (Figure 5.5) is shown to the user, presenting the number of families (i.e. clusters) that were found. It is then asked if the user wishes to proceed with supervised grouping (Section 4.4). This is a solution to treat *Case 1* of Table 3.1, where two messages are different but might mean the same system behavior (in that case, going from automatic adjustment of speed to manual), but could be analyzed by someone with knowledge about the system.

4.4 Supervised Clustering

The user's input will be used at this point, in order to make use of his system's knowledge. Two classes were created to execute this part of the solution:

ClusterPair : pairs two families, and stores their inter-cluster distance;

ClusteringFrame : implements the window used to interact with the user;

ClusterDendrogram : draws hierarchical clusters' dendrograms with [5].

The method where this is implemented is named *supervisedClustering* and is a method from the *Clusterer* class (seen in Listing 4.4). The pseudocode for this method is shown in Algorithm 5.

Algorithm 5 supervised

```
1:  procedure SUPERVISED
2:      new ClusteringFrame is created;
3:      pairs ← ∅, i ← 0
4:      while i < number of clusters - 1 do
5:          j ← i + 1
6:          while j < number of cluster do
7:              add to pairs a pair between clusters i and j
8:              j + +
9:          i + +
10:     order pairs in crescent order according to inter-cluster distance
11:     for each pair ∈ pairs do
12:         show user pair's information
13:         if user wants to group them then
14:             group them, update the list and go to 3
15:         if user wants to exit then
16:             break this loop
```

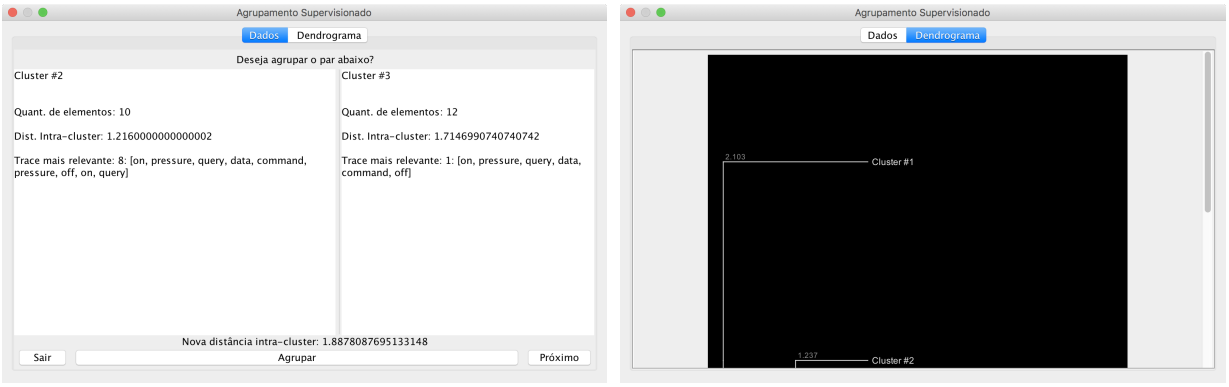
The information shown in the *ClusteringFrame* is, for each cluster:

- cluster number;
- number of elements;
- intra-cluster distance;
- most relevant trace.

Besides that, it is also shown what would be the new intra-cluster distance, if the user opts to group those families. The *most relevant trace* is the trace of the implied scenario

that is the one which is closest to the centroid of its family. Since each family centroid is calculated by the average of frequencies of its elements, it does not have a trace of execution to display to the user. Therefore, the trace of the closest implied scenario is shown, and that is referred as the *most relevant trace*.

An example of the window is shown in Figure 4.4a. It is noticeable that it presents the user with two tabs: one that displays data (i.e. the information discussed above) and another one that has a dendrogram drawn displayed. This dendrogram shows the results of a hierarchical clustering using the families that are left. This dendrogram provides the user with information on how close the families shown are, relative to how close other families are. For instance, if a family is very close to each other, while all the other families show a high inter-cluster distance, those close families could be grouped. This tab of the windows is shown in Figure 4.4b.



(a) Tab with data.

(b) Tab with the dendrogram.

Figura 4.4: Both tabs of the window used for supervised clustering.

It's also important to notice that in Algorithm 5, the clusters' pairs are ordered according to their inter-cluster distance, this way the pairs that are more closely related, appear first to the user, increasing the chances that they actually are similar. The process ends when one out of the following three occurs:

1. the user either chooses to exit (either by closing the window or pressing the exit button);
2. the user exhausts his options of aggregation, without opting to merge any of those;
3. there is only one family left.

After this process is finished, the results are exported (see 4.5).

4.5 Exporting Results

The first results of detected implied scenarios had already been presented in Section 4.2. This consists of CSV and ARFF files that contain the frequences of each message for the implied scenarios found, as well as a text file, which shows the traces of execution for those same scenarios. So, even if the user chose to skip the grouping process, these results have already been exported.

If the user opted to continue with the grouping process, three other files will be exported. First, the dendrogram drawn with [5] will be exported in two formats, PDF and JPG, using [15]. These are so that the user can perform further analysis, if desired, according to the implied scenarios' families. This file is seen in Figure 5.7.

Finally, the last file that will be exported is a text file. It shows how many families were left after the process finished, and also contains each family's information. This comprises their assigned number, number of implied scenarios that are a part of it, its intra-cluster distance, and the traces of its implied scenarios.

These families are ordered in a crescent manner, according to their intra-cluster distance, this way, the families that present the lowest intra-cluster distance will be listed first. This is done because the intra-cluster distance shows how similar a cluster is (the less distant the better), and therefore the families that are more similar have a better chance that all its implied scenarios had the same cause, and potentially all of them would be treated if only one of them was.

The traces of the implied scenarios of a family are also ordered. They are also ordered in a crescent manner, but it is done according to the euclidean distance between each scenario and the family's centroid. By doing this, the *most relevant trace* would be the first to be listed. This is a good information for the user, since this implied scenario is the closest to being the centroid, it'd be the point that better describes the whole family. Hence, the implied scenarios listed earlier have a better chance of being representative of their family, and consequently describe what caused it, and how to treat it. An example of this text file is shown in Figure 5.6.

With these results in hand, the user will be able to treat all the found implied scenarios in a faster way, as only the *most relevant trace* for each family would have to be treated.

Capítulo 5

Resultados

In order to analyze the methodology proposed in this work, two case studies were carried. The first one will analyze the *Boiler System*, which has been seen previously. The second one analyzes the *Cruiser System*, which has been brought up briefly in the previous section, and will have its model laid out in 5.2. The results obtained by the proposed methodology will be compared to the ones obtained with the ones by Lima [3] and in the original LTSA-MSC [6].

However, these results won't be used to make a quantitative analysis, but instead to show how the new results are easier to comprehend in a qualitative way. They will be used to show that this new extension to LTSA-MSC's functionality will provide simpler results than the original LTSA-MSC, while keeping the all information in case the user needs it, something that was not done by Lima [3].

Therefore, it's not the goal of this work to categorize implied scenarios found as good or bad, nor discuss how good the systems used are representative of the real world. It's also not necessary to discuss the validity of the test cases that were used to detect the implied scenarios, as they've been already discussed in other works (e.g. Uchitel et al. [6]) and are not part of this project's scope.

In order to share similarities with the results in Lima [3], the used number of implied scenarios to be found was 100. However, only the unsupervised part of the proposed methodology was applied, as this methodology should be generic enough to work with systems that the user doesn't have much knowledge about, and if the user does have a good knowledge of the system the supervised part can be seen as an extra factor of improvement.

It is also important to notice that given the concurrent nature of the systems analyzed, the results from the *Gathering Data* phase might differ from one run to another [7, 16]. Because of this, the results from each run were somewhat different from one another, unlike Lima [3]. That possibly happened because this proposed methodology, by using the

EM algorithm, will output different results for different datasets, while the methodology proposed by Lima in [3] will use the last messages of an execution trace, and doesn't take into account the other implied scenarios found. The differences found on each case study are explained on its respective subsection.

Finally, the computer used to perform these experiments was not dedicated, and its specifications are listed below:

Operating System : macOS Sierra 10.12.4

Processor : Intel(R) Core(TM) i7-6820HQ @ 2.7 GHz

RAM : 16 GB

5.1 Case Study: Boiler System

The first case study is a simple system that was seen previously in this work: the *Boiler System*. This system has four components and controls the temperature inside a boiler, according to pressure measured inside. An important goal on this system is the need to maintain the water level in a range of values, otherwise the system might suffer severe damage [6].

5.1.1 Step-by-Step Execution

Here the steps in 4.1 will be applied, one by one, considering the *Boiler System*. The figures shown were taken from one single sample run, and don't reflect the results on the analysis.

1. **Modeling in LTSA-MSC**:

   This system's modeling has already been presented in 2.3 and will be inserted in the LTSA-MSC tool. It uses the MSC format, which has already been described in this work. Figure 5.1 shows the model already opened in the tool. To go to the next step, the user must click the highlighted button.

2. **Detection Phase**:

   After the user chooses to proceed by clicking the button, a window will prompt him to enter how many implied scenarios should be searched (Figure 5.2a) and after the user enters a valid number (i.e. a positive integer) a window pops-up to show that the process is started (Figure 5.2b). The process of detecting implied scenarios will execute on the background, without any need for user interaction.

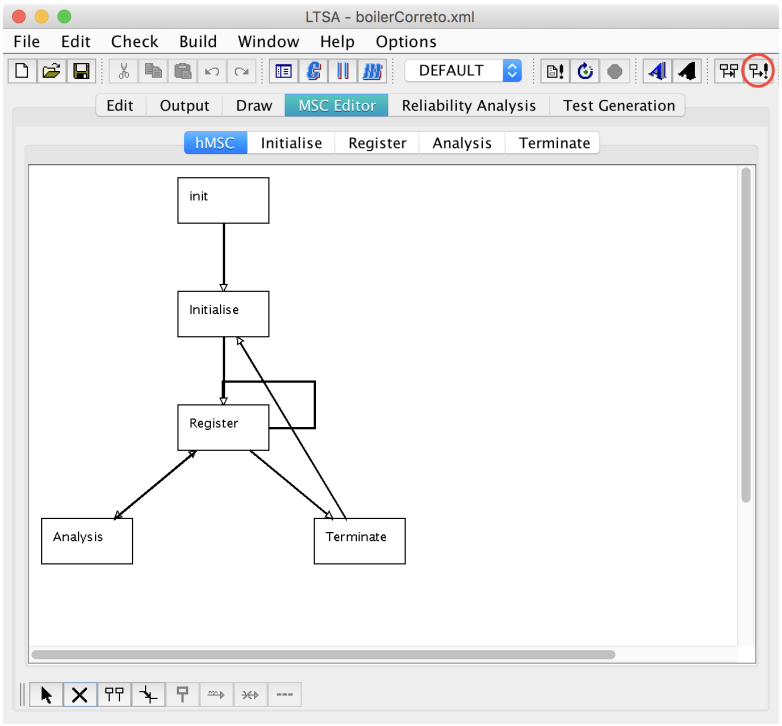
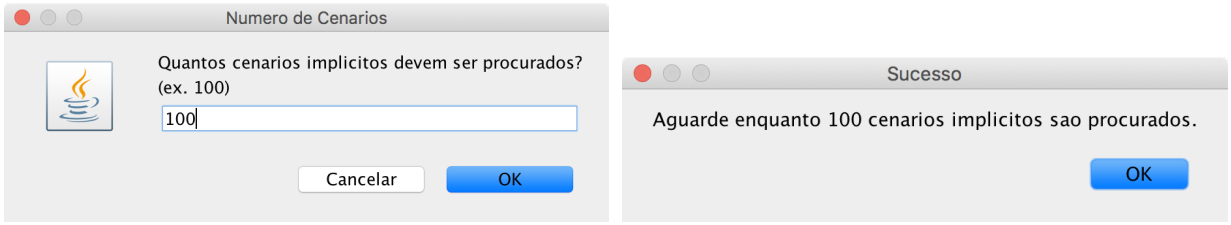


Figura 5.1: hMSC of the Boiler System opened in LTSA-MSC.



(a) Defining number of scenarios to be collected. (b) Notification of the start.

Figura 5.2: Initial interactions with the user.

3. **Prepare data for grouping**:

   With the implied scenarios detected, the first step is to convert their traces of executions into pairs of messages and frequencies. This conversion is done for each implied scenario. When this conversion is done, this first results are exported, the traces of execution and the message-frequency pairs, as the user might want to use them in a different manner and not proceed with the grouping in the tool. After these files are exported, the user is presented to yet another pop-up window, this time to let him know that the process has finished and show how much time it took to detect the scenarios, and also to export this first data. As seen in Figure 5.3, the time to convert and export is a lot lower than the detection time, therefore, this time will be disregarded in the analysis done below.

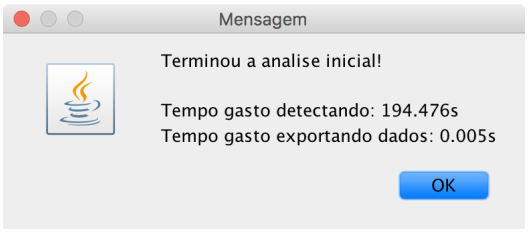


Figura 5.3: Times measured to detect and convert the implied scenarios, for the Boiler System.

4. **Group similar implied scenarios into families (unsupervised)**:

   With those files in hand, it is possible to group them into families. A window will ask the user if this grouping should be done (Figure 5.4), and if an affirmative answer is obtained, the process will be executed. When it's finished, the window (Figure 5.5 shows how many families were detected and asks if the next step should be carried. In this particular run, 8 families were detected.

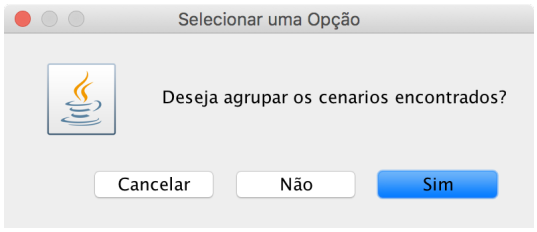


Figura 5.4: Asking the user if the implied scenarios found should be grouped.

5. **Group similar families together (supervised)**:

   This step was skipped, in order to make it a fair comparison to Lima [3], as in their work the user doesn't provide any input regarding the system. Therefore, the answer given on Figure 5.5 was *No*.

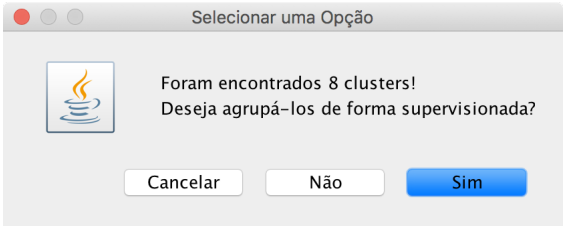


Figura 5.5: Showing how many families were found, and asking if should continue with supervised grouping, for the Boiler System.

6. **Export results**:

   If at least the unsupervised grouping was carried on, there are new results to report to the user. These will be exported as explained in the methodology, and are seen

in Figures 5.6 and 5.7. After they're exported, the window seen in Figure 5.8 is shown to let the user know that the whole process has finished.

```
Resultado da clusterizacao:

Foram encontrados 8 clusters.

---- ---- ---- ----

Cluster #1:

Quantidade de elementos: 8
Distancia intra-cluster: 0.1171875

Elementos:
39: [on, pressure, query, data, command, pressure, off, on, pressure, off, on, query]
67: [on, pressure, query, data, command, pressure, off, on, pressure, pressure, off, on, query]
68: [on, pressure, query, data, command, pressure, pressure, off, on, pressure, off, on, query]
72: [on, pressure, off, on, pressure, query, data, command, pressure, pressure, off, on, query]
73: [on, pressure, off, on, pressure, pressure, query, data, command, pressure, off, on, query]
79: [on, pressure, pressure, query, data, command, pressure, off, on, pressure, off, on, query]
84: [on, pressure, pressure, off, on, pressure, query, data, command, pressure, off, on, query]
43: [on, pressure, off, on, pressure, query, data, command, pressure, off, on, query]


Cluster #3:

Quantidade de elementos: 10
Distancia intra-cluster: 0.225

Elementos:
24: [on, pressure, off, on, pressure, off, on, pressure, off, on, query]
44: [on, pressure, off, on, pressure, off, on, pressure, pressure, off, on, query]
45: [on, pressure, off, on, pressure, pressure, off, on, pressure, off, on, query]
49: [on, pressure, pressure, off, on, pressure, off, on, pressure, off, on, query]
65: [on, pressure, off, on, pressure, off, on, pressure, pressure, pressure, off, on, query]
75: [on, pressure, off, on, pressure, pressure, off, on, pressure, pressure, off, on, query]
76: [on, pressure, off, on, pressure, pressure, pressure, off, on, pressure, off, on, query]
82: [on, pressure, pressure, off, on, pressure, off, on, pressure, pressure, off, on, query]
85: [on, pressure, pressure, off, on, pressure, pressure, off, on, pressure, off, on, query]
88: [on, pressure, pressure, pressure, off, on, pressure, off, on, pressure, off, on, query]


Cluster #6:

Quantidade de elementos: 13
Distancia intra-cluster: 1.0782885753299956

Elementos:
14: [on, pressure, query, data, command, pressure, query, data, command, off]
40: [on, pressure, query, data, command, pressure, pressure, pressure, query, data, command, off]
48: [on, pressure, pressure, query, data, command, pressure, pressure, query, data, command, off]
```

Figura 5.6: Part of report exported that shows each families' elements, for the Boiler System.

5.1.2 Comparing Different Clustering Results

As the unsupervised clustering done by the EM algorithm has a probabilistic result, the algorithm was applied one thousand with the same 100 implied scenarios detected. The breakdown of the number of families detected is shown in Figure 5.9. All these 1000 results took 72 seconds to be collected, which equates to 72 ms for each run of the algorithm. That shows that this replication of the algorithm can be scaled for larger systems.

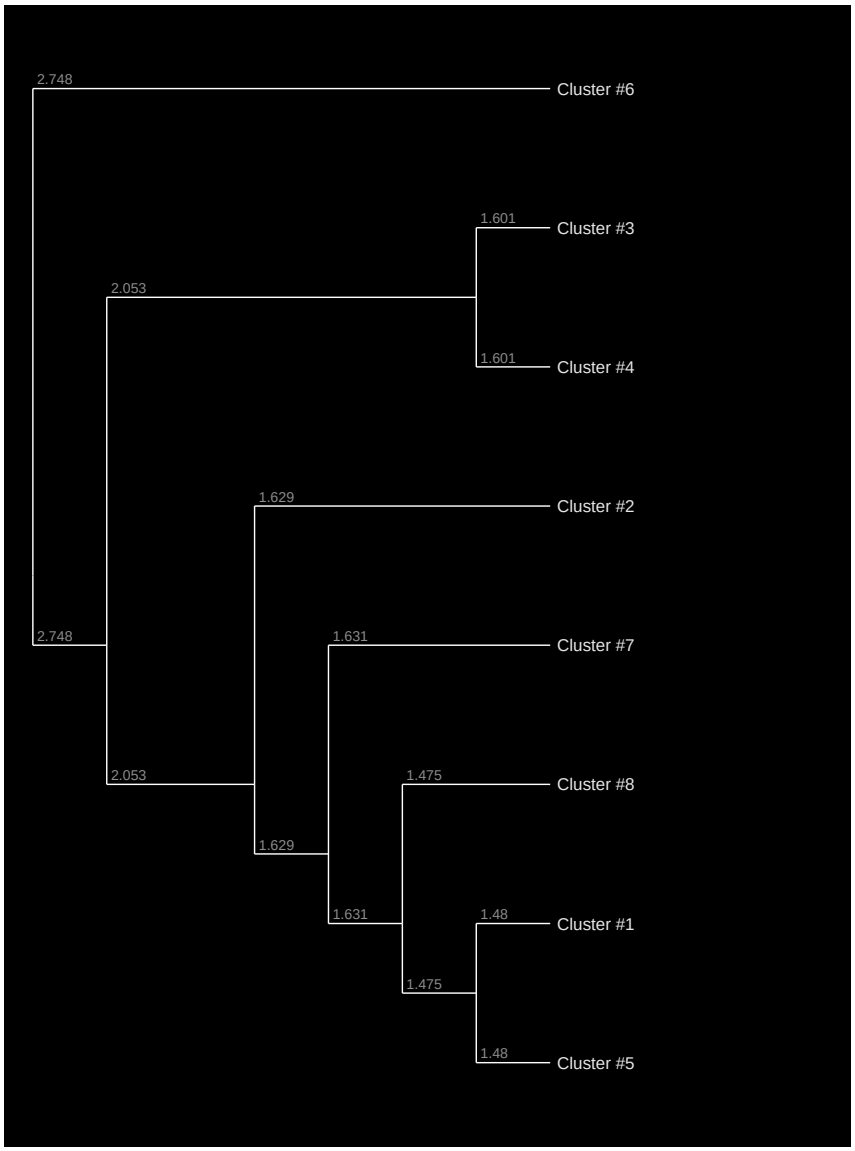


Figura 5.7: Hierarchical Clustering that shows how similar are the families, for the Boiler System.

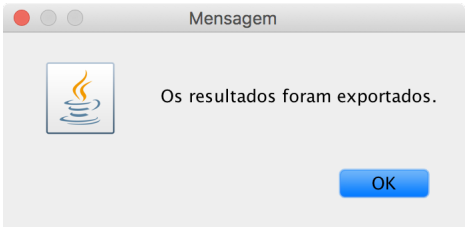


Figura 5.8: Saying that it's finished and results have been exported.

It is noticed that in more than half of those 1000 times, two clusters were detected by the EM algorithm, which suggests that two families of implied scenarios would be the better fit for the implied scenarios that were analyzed.

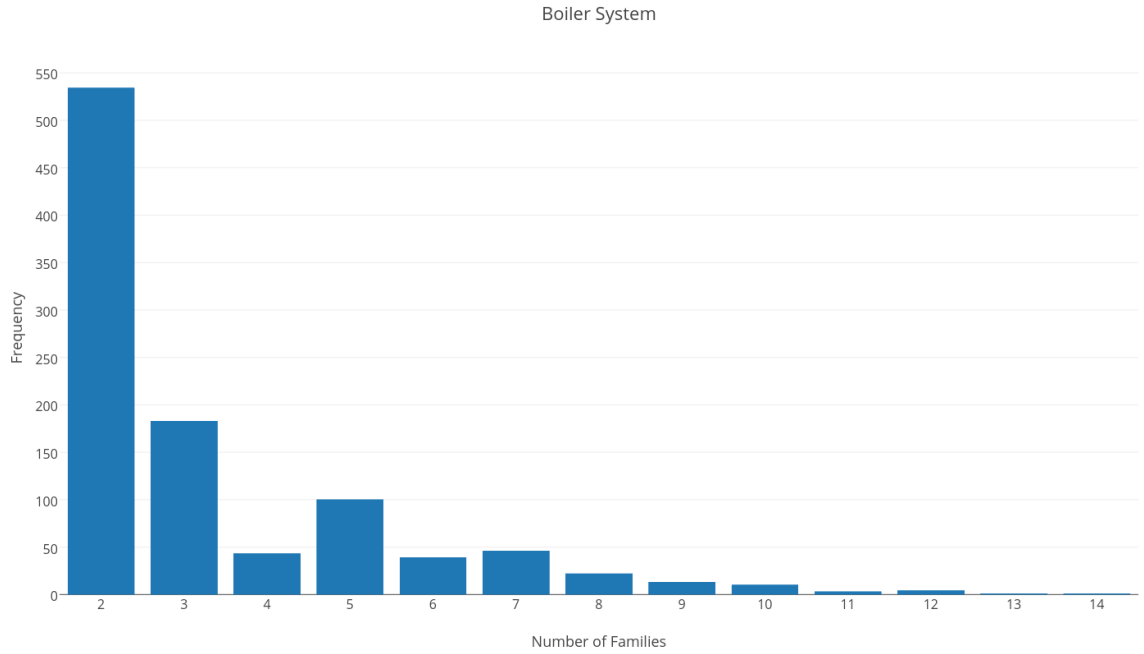


Figura 5.9: Frequency of number of families detected.

However, when the average log likelihood, intra-cluster distance, and inter-cluster distance were taken into account for those results, they both show that the user would have better results with more families detected. In Figure 5.10, the average log likelihood is shown, and the error bars represent the standard deviation, while Figure 5.11 shows the average intra-cluster distance, inter-cluster distance, and beta-cv.

Both the average likelihood and the inter-cluster distance increase as the number of families goes up, while the average intra-cluster distance decreases. Therefore, all three metrics show that as more families are detected, more distinct clusters are formed (as the likelihood and inter-cluster distance are higher) and the elements of each cluster are more similar to each other (as the intra-cluster distance is lower).

When looking at the highest average log likelihood, twelve families seem to be the best number for the dataset. However, twelve families were only detected in 5 out of 1000 runs, only 0,5% out of the total.

Similarly, both intra-cluster and inter-cluster distances have their best results for a number of families with very low frequency. The inter-cluster distance has its highest value at eleven clusters, which has a frequency of only 3 runs (0,3% of the total). And the intra-cluster distance has its lowest value at fourteen families detectec, which only happened once (0,1% of the runs).

By analyzing the graph, the point where the beta-cv seems to stabilize is at seven families detected. Even though its frequency is higher than the ones seen above, it is still very low at only 46 runs (4,6% of the total).

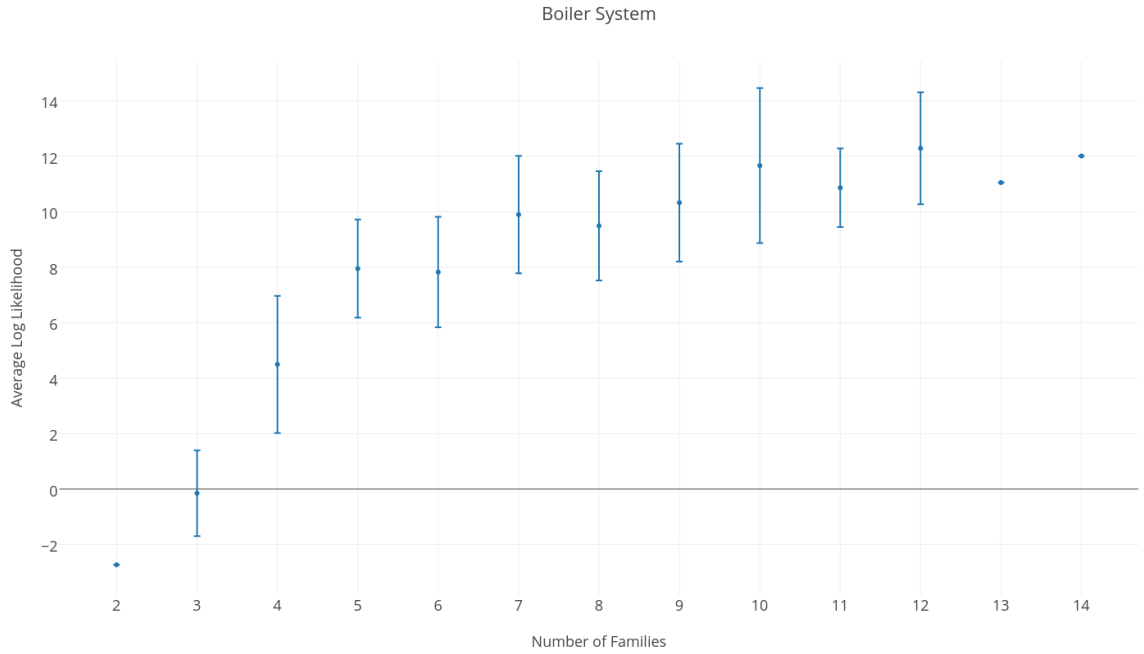


Figura 5.10: Average log likelihood and standard deviation by number of families.

Therefore, although the metrics above say otherwise, because the frequency is considerably higher for one single number of families (i.e. two families), our option to pick the best clustering among the ones that detected two families holds up correct. That is, the clustering provided by the EM algorithm that showed the highest log likelihood, while detecting two clusters.

5.2 Case Study: Cruiser System

The second system used to analyze the methodology is the *Cruiser System*. It's a car's cruise control system, that is, it's used to keep a car's speed constant. It is considerably more complex than the *Boiler System*, containing more components and messages in its scenarios, and that will allow to check how well the proposed methodology scales. As expected, the measure times for this system are higher than the ones measured with the *Boiler System*.

This system maintains the car in a constant speed by measuring the current speed and adjusting the car's throttle, so it gets to the desired speed. This system is described and detailed by Magee et al. [9].

The system has three buttons: on, off, and resume. When 'on' is pressed, the current speed is registered and then maintained by throttle adjustments. Similarly, when 'resume' is pressed, the same thing happens, with the exception that the speed maintained is the

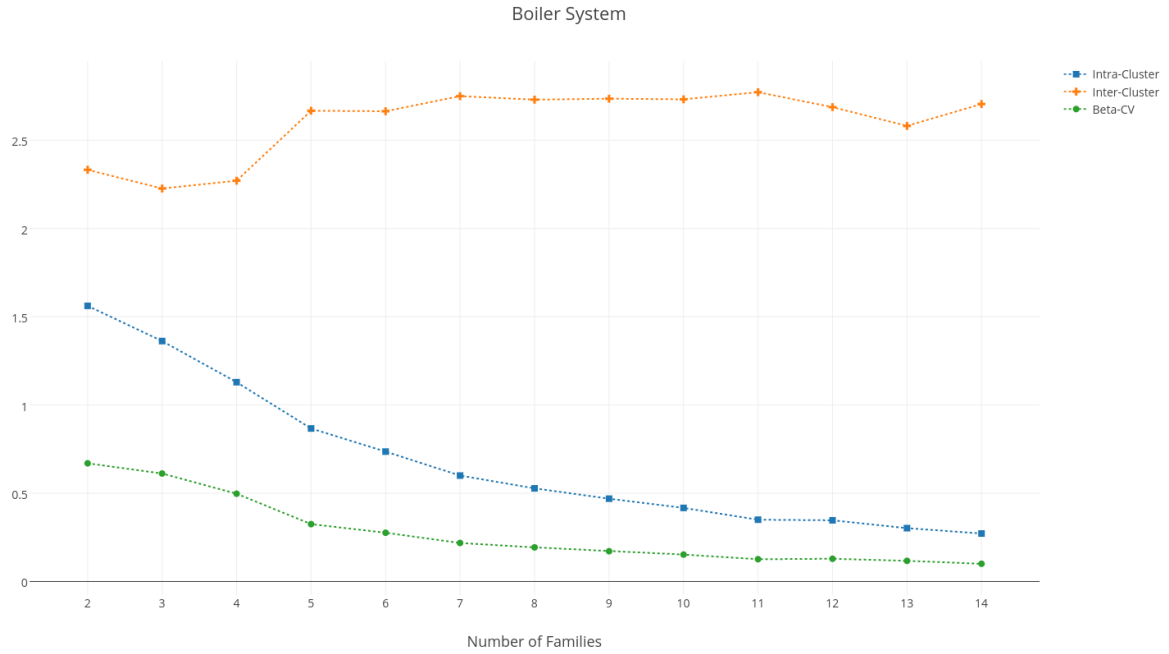


Figura 5.11: Average intra-cluster distance, inter-cluster distance and beta-cv.

last registered one, not the current one. The system is turned off when the driver either presses the 'off' button, accelerates or breaks.

The modeling for this system is shown in Figures 5.12 and 5.13. On the former each individual scenario is depicted, while on the latter, the interaction between them is showed, in a high-level MSC.

5.2.1 Step-by-Step Execution

Here the steps in 4.1 will be applied, one by one, considering the *Cruiser System*. The figures shown were taken from one single sample run, and don't reflect the results on the analysis. The process is very similar to the one presented on the previous Case Study, with the differences being the results.

1. **Modeling in LTSA-MSC**:

   This system's modeling has already been presented and Figure 5.14 shows the model already opened in the tool. To go to the next step, the user must click the highlighted button.

2. **Detection Phase**:

   After the user chooses to proceed by clicking the button, a window will prompt him to enter how many implied scenarios should be searched (left window in Figure 5.2a) and after the user enters a valid number (i.e. a positive integer) a window pops-up

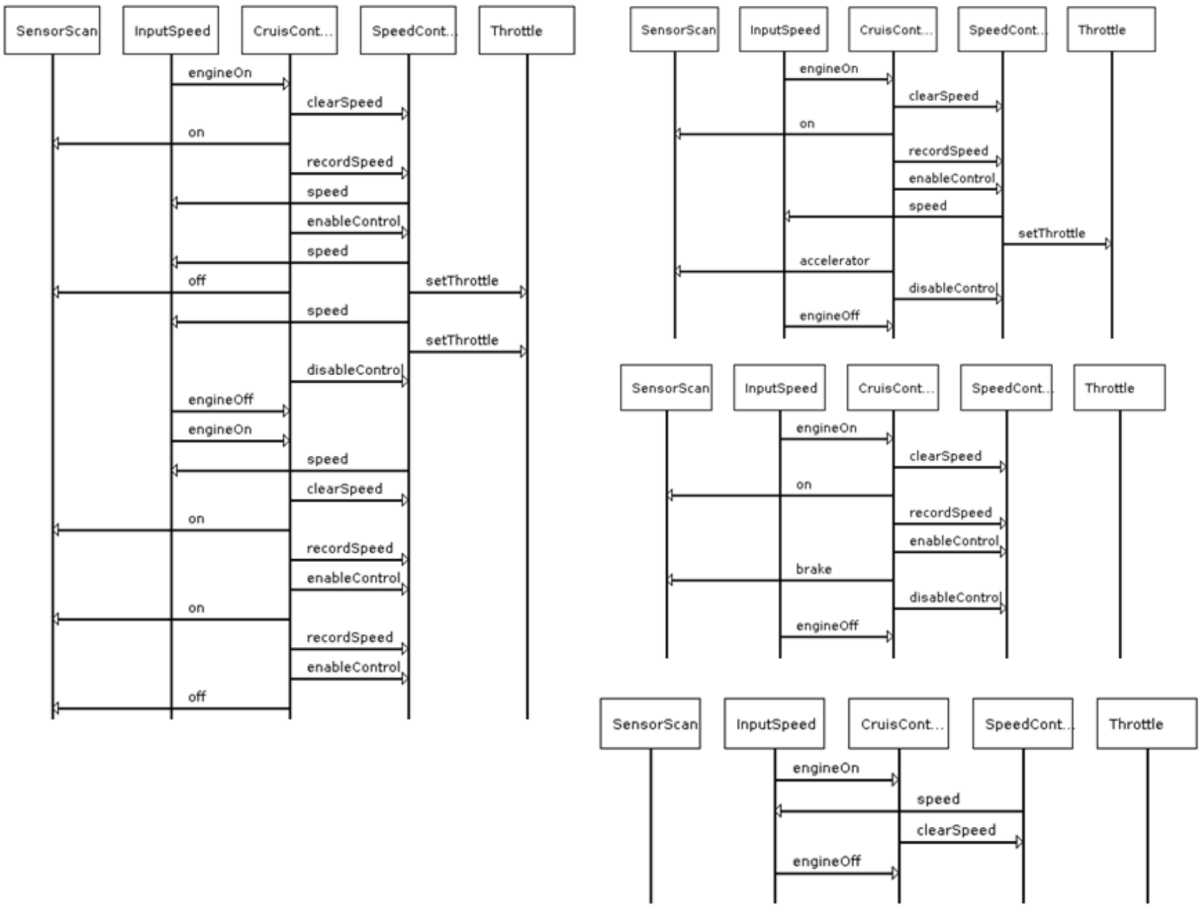


Figura 5.12: MSCs of the *Cruiser System*'s scenarios. Clockwise, starting from the left: Scen1, Scen2, Scen3, and Scen4. Taken from [3].

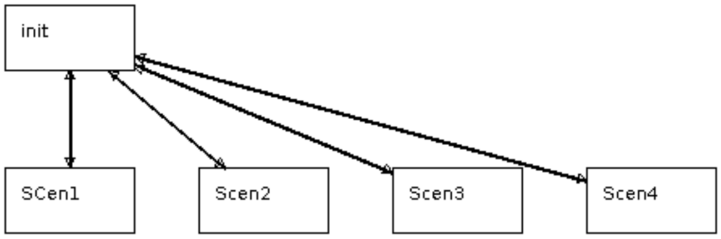


Figura 5.13: hMSC for the *Cruiser System*. Taken from [3].

to show that the process is started (right window in Figure 5.2b). The process of detecting implied scenarios will execute on the background, without any need for user interaction.

3. **Prepare data for grouping**:

   With the implied scenarios detected, the first step is to convert their traces of executions into pairs of messages and frequencies. This conversion is done for each implied scenario. When this conversion is done, this first results are exported, the

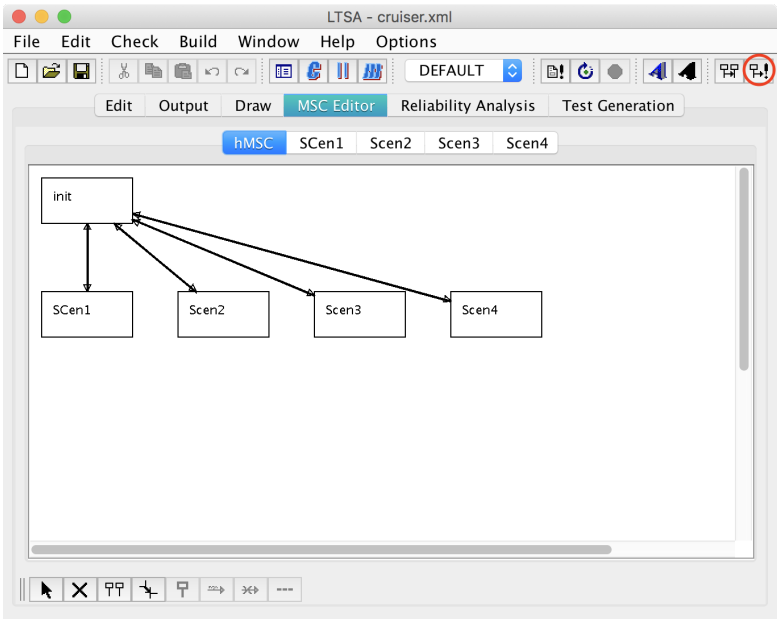


Figura 5.14: hMSC of the Cruiser System opened in LTSA-MSC.

traces of execution and the message-frequency pairs, as the user might want to use them in a different manner and not proceed with the grouping in the tool. After this files are exported, the user is presented with yet another pop-up window, this time to let him know that the process has finished and show how much time it took to detect the scenarios, and also to export this first data. As seen in Figure 5.15, the time to convert and export is a lot lower than the detection time, therefore, this time will be disregarded in the analysis done below.

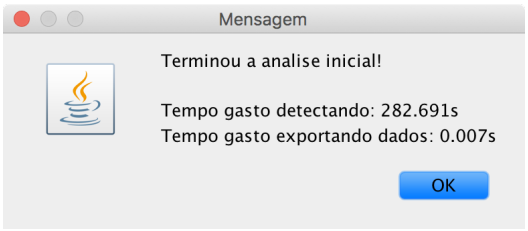


Figura 5.15: Times measured to detect and convert the implied scenarios for the Cruiser System.

4. **Group similar implied scenarios into families (unsupervised)**:

   With those files in hand, it is possible to group them into families. A window will ask the user if this grouping should be done (Figure 5.4), and if an affirmative answer is obtained, the process will be executed. When it's finished, the window (Figure 5.16 shows how many families were detected and asks if the next step should be carried. In this particular run, 8 families were detected.

5. **Group similar families together (supervised)**:

   This step was skipped, in order to make it a fair comparison to Lima [3], as in their work the user doesn't provide any input regarding the system. Therefore, the answer given on Figure 5.16 was *No*.

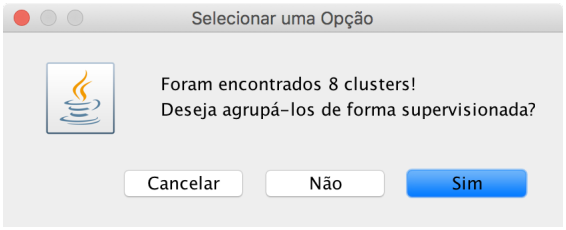


Figura 5.16: Showing how many families were found, and asking if should continue with supervised grouping, for the Cruiser System.

6. **Export results**:

   If at least the unsupervised grouping was carried on, there are new results to report to the user. These will be exported as explained in the methodology, and are seen in Figures 5.17 and 5.18. After they're exported, the window seen in Figure 5.8 is shown to let the user know that the whole process has finished.

```
Resultado da clusterizacao:

Foram encontrados 8 clusters.

---- ---- ---- ----

Cluster #7:

Quantidade de elementos: 1
Distancia intra-cluster: 0.0

Elementos:
31: [engineOn, clearSpeed, on, recordSpeed, speed, enableControl, speed, setThrottle, speed, setThrottle, brake]


Cluster #1:

Quantidade de elementos: 4
Distancia intra-cluster: 1.015625

Elementos:
34: [engineOn, speed, clearSpeed, engineOff, engineOn, speed, clearSpeed, engineOff, engineOn, speed, clearSpeed, speed]
71: [engineOn, speed, clearSpeed, engineOff, engineOn, speed, clearSpeed, engineOff, engineOn, speed, clearSpeed, engineOff, engineOn, speed, clearSpeed, on]
70: [engineOn, speed, clearSpeed, engineOff, engineOn, speed, clearSpeed, engineOff, engineOn, speed, clearSpeed, engineOff, engineOn, speed, clearSpeed, speed]
53: [engineOn, speed, clearSpeed, engineOff, engineOn, speed, clearSpeed, engineOff, engineOn, speed, clearSpeed, engineOff, engineOn, clearSpeed, engineOff]


Cluster #2:

Quantidade de elementos: 8
Distancia intra-cluster: 1.6015625

Elementos:
24: [engineOn, speed, clearSpeed, engineOff, engineOn, speed, clearSpeed, engineOff, engineOn, clearSpeed, engineOff]
54: [engineOn, speed, clearSpeed, engineOff, engineOn, speed, clearSpeed, engineOff, engineOn, clearSpeed, on, recordSpeed, speed, enableControl, accelerator]
59: [engineOn, speed, clearSpeed, engineOff, engineOn, speed, clearSpeed, engineOff, engineOn, clearSpeed, on, recordSpeed, enableControl, accelerator, disableControl]
```

Figura 5.17: Part of report exported that shows each families' elements, for the Cruiser System.

5.2.2 Analysis

As expected, the time went up when compared to the *Boiler System*'s sample run. However, it wasn't a huge increase and the measure times in 5.2.3 show that it took around 5 minutes. Therefore, it was observed, like in Lima [3], that the detection methodology proposed can be applied to more complex systems.

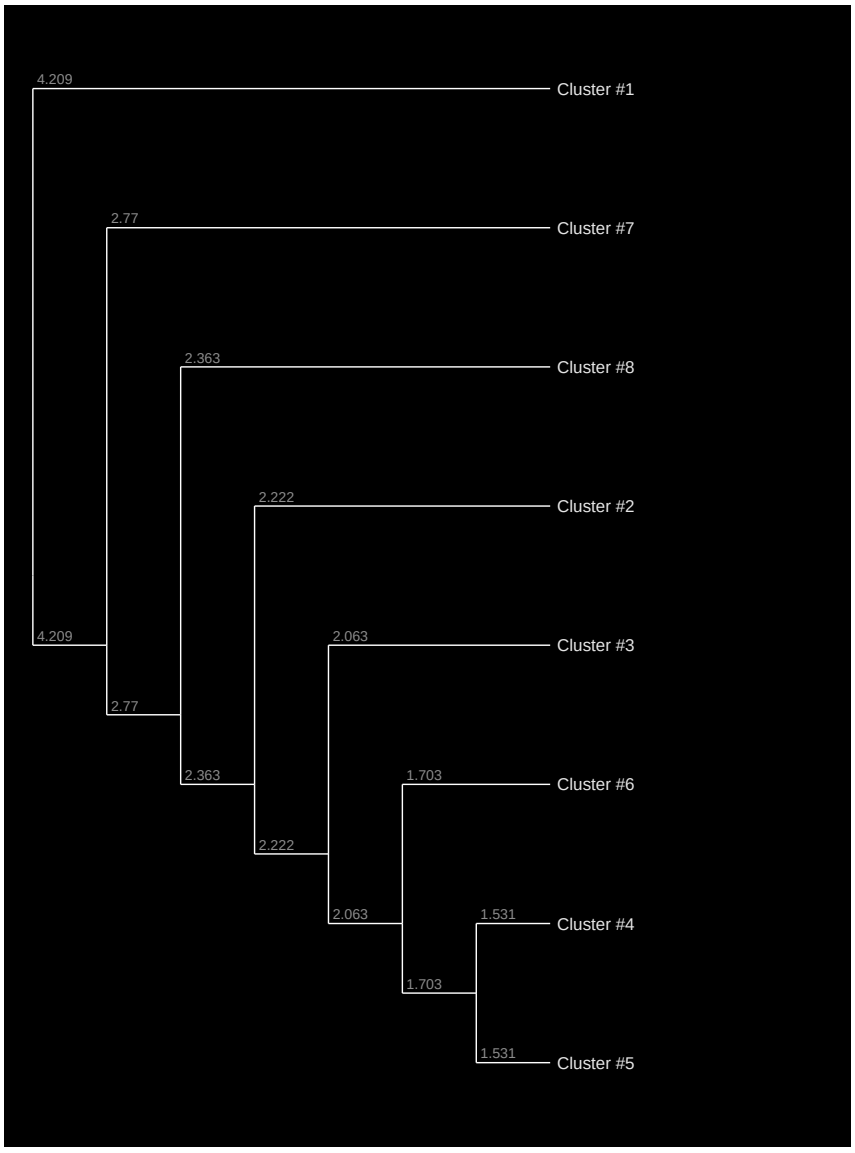


Figura 5.18: Hierarchical Clustering that shows how similar are the families, for the Cruiser System.

Again, eight families of implied scenarios were found on this sample run. However, this time that doesn't surpass the number of families detected in Lima [3], where 15 families were encountered in every run. Hence, this illustrates that the results obtained using both solutions are very different, unlike what might have been thought when seeing the sample run for the *Boiler System*. This happens because the supervised grouping phase only allows the user to group different families, and not split them into smaller ones, as it'd be necessary to achieve a higher number of families.

As seen in Figure 5.17, Cluster #2's visible elements, that is, the three elements that show up on the cropped report, have different messages on the last two positions of their traces. This reinforces the knowledge that the proposed methodology doesn't give more

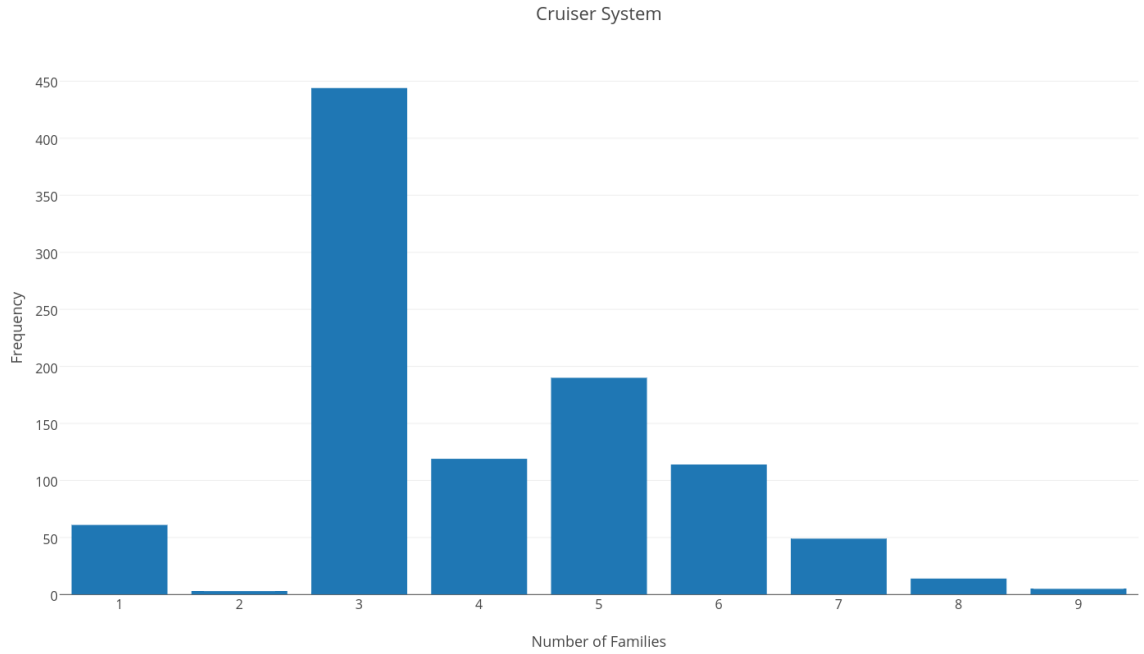


Figura 5.19: Breakdown of frequencies for each number of families detected.

importance to the last messages of the trace, it considers every message equally.

Lastly, Clusters #1 and #7 have very few elements, and consequently very low intra-cluster distance, as seen in Figure 5.17. This shows that these elements are very far away from the other ones in the dataset, and that can be observed in Figure 5.18, as both families are the last ones to be included in the hierarchical clustering, which indicates that they are the furthest ones apart. By having elements fit into their own clusters, show that if a couple of implied scenarios that have a lot of repetition, that is, repeated the same scenarios (or messages) a lot of times, would be grouped together only because they have very long traces, even if they have different causes.

5.2.3 Comparing Different Clustering Results

The same experiments that were run with the *Boiler System* were done again, that is, the EM algorithm was applied to 100 implied scenarios one thousand times. The frequency for each number of families is shown in Figure 5.19.

Just as it was observed previously, there is a specific number of families that far surpasses the other cases. In this case, three families of implied scenarios seem to be the best fit for the dataset. However, differently from what was seen in the previous case study, the other metrics back up the frequency, as they all point to the same number of families.

In Figure 5.20, the average log likelihood is shown for the different number of families defined. It is noticeable that when a new cluster is added going from 2 families to 3 families defined, there is a significant increase to the log likelihood. Although there is still an increase when more clusters are added, in no other point an increase as sharp as this one is noticed. This shows that when this third cluster is added to the dataset, it results in a very good separation of the elements, and even if better results are observed with more clusters, their resulting models would be a lot more complex to not a much higher gain.

When analyzing the average intra-cluster distance, average inter-cluster distance, and average beta-cv (seen in Figure 5.21) a significant change is observed when the number of families is changed from 2 to 3. The average inter-cluster distance spikes when this new cluster is added, which shows that the families are a lot more well defined, as they are less similar to each other.

Similarly, the average intra-cluster distance decreases with a very high slope at the same point. This reinforces the idea that with this third cluster added, the families are better defined, as the elements of a family are more similar to each other, that is, less distant from one another.

Lastly, the beta-cv looks to stabilize at the same point, as its value doesn't fluctuate as much in any other point. All these metrics show that 3 families is the better result to be kept, because even if a higher inter-cluster distance and lower intra-cluster distance is observed for 8 families, the difference is not as drastic as changing from 2 to 3, and defining 8 families would result in a much more complex model, which probably wouldn't make the gain in the distances worth it.

Therefore, opting to choose three families looks like a very strong choice, as all the metrics point to this same result, and not only the frequency as was the case for the Boiler System.

Finally, the time taken for all this executions combined was 158s, an average of 158ms for each one. This shows that this repetition is scalable to more complex systems, as it did not take too much time to run it considering the Cruiser System.

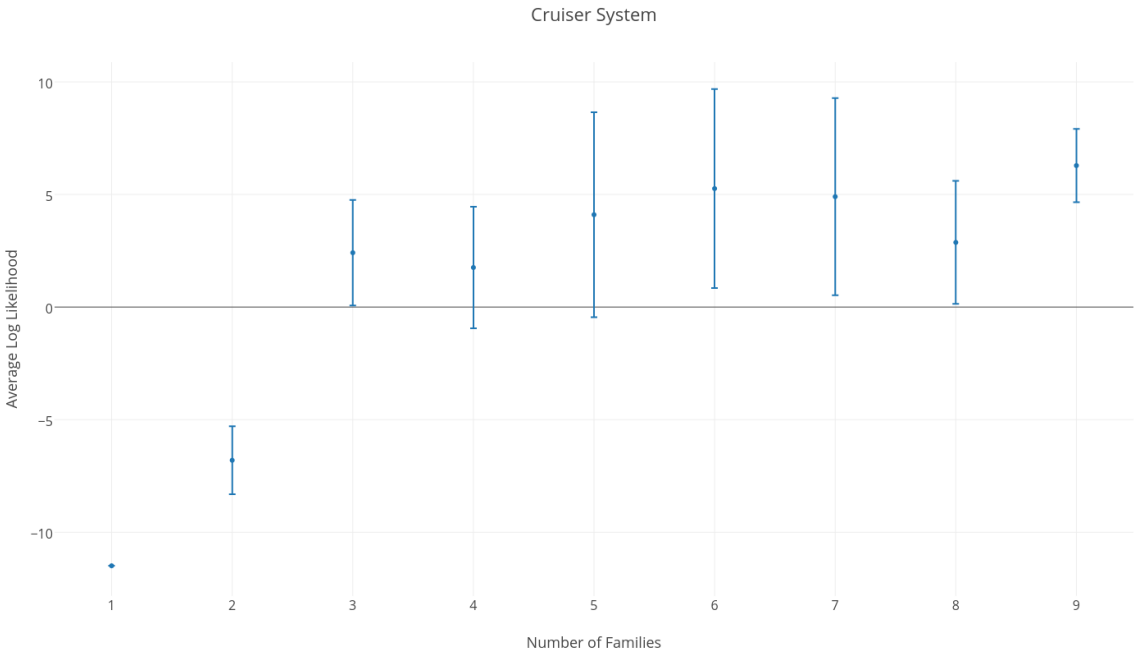


Figura 5.20: Average log likelihood and standard deviation by number of families.

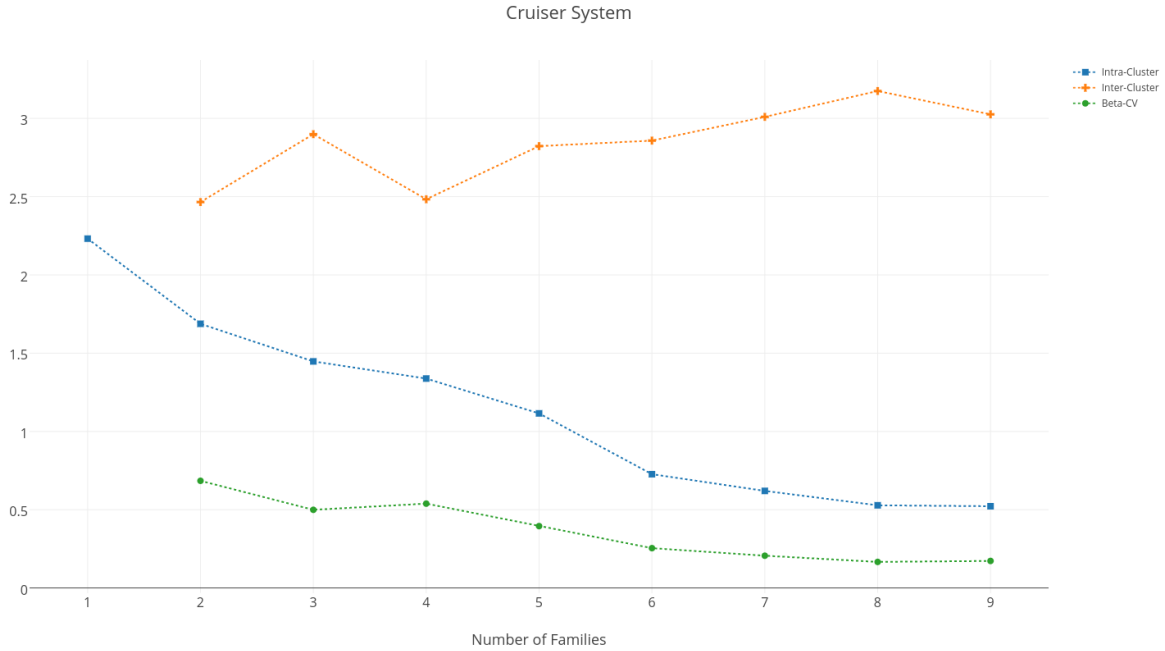


Figura 5.21: Average intra-cluster distance, inter-cluster distance and beta-cv.

Capítulo 6

Conclusão

Os resultados obtidos nas execuções teste foram satisfatórios, considerando que o principal objetivo deste trabalho era estender as possibilidades propostas por Lima [3], enquanto que uma maneira mais simples de tratar os cenários implícitos (quando comparado a Uchitel et al. [6]) era mantida.

A solução proposta também apresenta ao usuário a oportunidade de ajustar as famílias de cenários implícitos ao sistema, já que caso o usuário conheça o sistema a ponto de julgar duas famílias distintas muito similares, a solução propicia a possibilidade de agrupá-las em uma única família. Esta interação com o usuário permite que a solução ofereça resultados específicos ao sistema, o que não seria possível se as únicas análises fossem não supervisionadas.

Porém, como a primeira fase da criação de famílias é conseguida com uma técnica de clusterização, não existe uma garantia que estas famílias terão algum significado para o usuário, visto que o algoritmo EM pode encontrar clusters que somente se relacionam quando vistos com base em estatísticas, e não quando comparados com o sistema. Por causa disto, é importante que os resultados iniciais (i.e. traces dos cenários e frequências das mensagens) sejam exportados, pois eles permitem que o usuário realize seu próprio agrupamento, sem ter a necessidade de executar o processo de detecção novamente.

Por esta técnica de clusterização (EM) ser probabilística, os resultados podem não ser consistentes em diferentes execuções, e por isto é importante a execução da técnica múltiplas vezes, como foi feito no trabalho. Os resultados obtidos mantendo o número de famílias como a moda da frequência das diversas execuções apresentou um resultado desejado, que era a consistência no número de famílias encontradas em diferentes testes.

Além disso, os resultados para o segundo estudo de caso (Sistema Cruiser) foram bons, pois todas as métricas analisadas apontaram para o mesmo número de famílias. Porém, como indicado na análise de resultados, no primeiro estudo de caso (Sistema de Caldeira), outras métricas talvez pudessem ter sido utilizadas para indicar um outro número de

famílias para o conjunto de cenários implícitos analisado que melhor caracterizariam ele. Isto indica que mais estudos devem ser realizados para indicar qual a melhor solução.

Entretanto, como o usuário tem o resultado da clusterização hierárquica à sua disposição, ele pode analisar as famílias por conta própria, caso julgue necessário, e encontrar elementos que, a seu ver, deveriam estar em famílias diferentes. Desta forma, ele seria capaz de reorganizar as famílias, de modo que elas possam ser tratadas de uma forma uniforme. No pior dos casos, se o usuário julgar que nenhuma família está bem determinada e é necessário analisar cenário a cenário, ele terá que analisar o mesmo número de cenários que na solução original, proposta por Uchitel et al. [6].

Ainda é necessário um estudo empírico que comprove que os resultados deste trabalho (assim como do trabalho de Lima [3]) são uma melhora quanto aos resultados da solução original, proposta por Uchitel et al. em [6]. Porém, este estudo não faz parte do escopo deste trabalho e, portanto, é sugerido nos trabalhos futuros (Seção 6.2), assim como outros trabalhos. Na Seção 6.1, ameaças a validade deste trabalho são apresentadas.

Finalmente, com tudo isto considerado, os resultados foram satisfatórios e o trabalho proveitoso, possivelmente podendo ser aplicado em situações reais.

6.1 Ameaças a Validade

Não existe uma base empírica para confirmar que a metodologia proposta é melhor do que as anteriormente propostas, por Lima [3] e Uchitel et al. [6], portanto não é possível afirmar categoricamente que esta solução deve ser utilizada no lugar destas outras.

Também, foi assumido que as mensagens eram assíncronas e poderiam ser analisadas fora de ordem, como foram convertidas em pares de mensagens e frequências, e então agrupadas com base nisto. Porém, a ordem dos traces de execução talvez mudem o resultado final e cenários implícitos, que na verdade são bem diferentes, acabam sendo agrupados em uma mesma família, somente por suas frequências. Uma forma de mitigar este problema seria atribuindo pesos às mensagens, com base na sua posição no trace. Assim, mensagens que ocorrem mais perto do fim, onde ocorreu a falha, teriam mais peso na hora da clusterização.

Por fim, há outro problema que poderia ocorrer na parte de clusterização. Cenários implícitos que contêm muitas mensagens provavelmente seriam agrupados em uma mesma família, visto que eles tem uma maior frequência nestas mensagens. Considere, por exemplo, dois cenários hipotéticos encontrados no sistema *Boiler*, cujas frequências para a mensagem *pressure* são acima de 1000. Isto é, o sistema mediu a pressão da caldeira mais de 1000 vezes durante suas execuções. Eles seriam agrupados na mesma família, pois os outros cenários implícitos encontrados, provavalmente não teriam essa extensão.

6.2 Trabalhos Futuros

Como já dito acima, a primeira sugestão é de um estudo empírico que compare as soluções propostas por Lima [3], Uchitel et al. [6] e a que foi proposta neste trabalho, para verificar, de uma forma quantitativa, qual a melhor abordagem para a detecção de cenários implícitos. Neste trabalho proposto, seria importante comparar a metodologia proposta aqui com a adição de pesos às mensagens e, também, mantendo a ordem dos traces.

Também seria interessante propor uma metodologia para geração de testes de caso com base em famílias detectadas. Ainda mais, verificar se é possível tratar estas famílias enquanto elas são detectadas. Desta forma, o usuário poderia realizar toda a análise (detecção e tratamento) de cenários implícitos em uma só ferramenta.

Por fim, seria desejável encontrar uma condição de parada para o passo de detecção de cenários implícitos, visto que esta é a parte do processo que consome mais tempo. Uma maneira possível seria verificar as famílias de cenários implícitos formadas, enquanto novos cenários são detectados. Caso elas estabilizem em algum ponto, o processo de detecção de novos cenários implícitos poderia ser parado.

Referências

[1] Avizienis, A., J. C. Laprie, B. Randell e C. Landwehr: *Basic concepts and taxonomy of dependable and secure computing*. IEEE Transactions on Dependable and Secure Computing, 1(1):11–33, janeiro 2004, ISSN 1545-5971. `http://ieeexplore.ieee.org/document/1335465/`, acesso em 2017-05-29. x, 1, 5, 8

[2] Uchitel, Sebastian, Jeff Kramer e Jeff Magee: *Detecting implied scenarios in message sequence chart specifications*. página 74. ACM Press, 2001, ISBN 978-1-58113-390-5. `http://portal.acm.org/citation.cfm?doid=503209.503220`, acesso em 2017-06-11. x, 1, 3, 4, 9, 10

[3] Lima, Filipe P.: *Uma Proposta para a Otimização de Análise de Cenários Implícitos*. Monografia (Graduação), Universidade de Brasília, 2016. x, xi, 1, 2, 3, 11, 12, 19, 20, 22, 23, 32, 33, 35, 41, 43, 44, 48, 49, 50

[4] Matteucci, Matteo: *A Tutorial on Clustering Algorithms*. `https://home.deib.polimi.it/matteucc/Clustering/tutorial_html/`, acesso em 2017-06-05. x, 3, 12, 15, 16

[5] Han, Mira V. e Christian M. Zmasek: *phyloXML: XML for evolutionary biology and comparative genomics*. BMC bioinformatics, 10:356, outubro 2009, ISSN 1471-2105. x, 18, 29, 31

[6] Uchitel, Sebastian, Robert Chatley, Jeff Kramer e Jeff Magee: *LTSA-MSC: Tool Support for Behaviour Model Elaboration Using Implied Scenarios*. Em Goos, Gerhard, Juris Hartmanis, Jan van Leeuwen, Hubert Garavel e John Hatcliff (editores): *Tools and Algorithms for the Construction and Analysis of Systems*, volume 2619, páginas 597–601. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003, ISBN 978-3-540-00898-9 978-3-540-36577-8. `http://link.springer.com/10.1007/3-540-36577-X_44`, acesso em 2017-06-08, DOI: 10.1007/3-540-36577-X_44. 1, 3, 11, 19, 32, 33, 48, 49, 50

[7] Reis, Thiago P.: *Uma Abordagem para o Uso de Cenários Implícitos na Geração de Casos de Teste para Sistemas Concorrentes*. Dissertação (Mestrado), Universidade de Brasília, 2015. 1, 3, 11, 19, 32

[8] Alur, R., K. Etessami e M. Yannakakis: *Inference of message sequence charts*. IEEE Transactions on Software Engineering, 29(7):623–633, julho 2003, ISSN 0098-5589. `http://ieeexplore.ieee.org/document/1214326/`, acesso em 2017-06-08. 1, 3, 11, 19

[9] Magee, Jeff e Jeff Kramer: *Concurrency: state models & Java programs*. Wiley, Chichester, England ; Hoboken, NJ, 2006, ISBN 978-0-470-09355-9. OCLC: ocm64084212. 4, 39

[10] Piech, Chris: *K Means*. `http://stanford.edu/~cpiech/cs221/handouts/kmeans.html`, acesso em 2017-06-05. 12, 15, 16

[11] Dempster, Arthur P, Nan M Laird e Donald B Rubin: *Maximum likelihood from incomplete data via the EM algorithm*. Journal of the royal statistical society. Series B (methodological), páginas 1–38, 1977. 12, 15, 22

[12] Borman, Sean: *The expectation maximization algorithm - a short tutorial*. 2004. `http://www.bioen.utah.edu/wiki/images/6/65/EMalgorithm.pdf`. 16

[13] Witten, I. H. e I. H. Witten (editores): *Data mining: practical machine learning tools and techniques*. Elsevier, Amsterdam, fourth edition edição, 2017, ISBN 978-0-12-804291-5. 16, 20, 25

[14] Do, Chuong B e Serafim Batzoglou: *What is the expectation maximization algorithm?* Nature Biotechnology, 26(8):897–899, agosto 2008, ISSN 1087-0156. `http://www.nature.com/doifinder/10.1038/nbt1406`, acesso em 2017-06-06. 16, 28

[15] Seifert, Erich: *VectorGraphics2d*. `https://github.com/eseifert/vectorgraphics2d`, acesso em 2017-06-06. 31

[16] Uchitel, Sebastian, Jeff Kramer e Jeff Magee: *Incremental elaboration of scenario-based specifications and behavior models using implied scenarios*. ACM Transactions on Software Engineering and Methodology, 13(1):37–85, janeiro 2004, ISSN 1049331X. `http://portal.acm.org/citation.cfm?doid=1005561.1005563`, acesso em 2017-05-30. 32