Knowledge Discovery for Avionics Maintenance : An Unsupervised Concept Learning Approach

Luis Palacios Medinacelli

NNT : 2019SACLS130

Thèse de doctorat

Knowledge Discovery for Avionics Maintenance: An Unsupervised Concept Learning Approach

Thèse de doctorat de l'Université Paris-Saclay
préparée à l'Université Paris-Sud

Ecole doctorale n°580 Sciences et Technologies de l'information et de la Communication (STIC)
Spécialité de doctorat : Informatique

Thèse présentée et soutenue à Orsay, le 4 Juin 2019, par

LUIS PALACIOS MEDINACELLI

Composition du Jury :

Bruno DEFUDE Professeur, Telecom SudParis	Président
Myriam LAMOLLE Professeur, Université Paris 8	Rapporteur
Bernard ESPINASSE Professeur, Université d'Aix-Marseille	Rapporteur
Nhan LE THANH Professeur, Université de Nice	Examinateur
Carlos C. INSAURRALDE Senior Lecturer, Bristol - Univ. of the West of England (UWE)	Examinateur
Chantal REYNAUD Professeur, Université Paris-Sud	Directrice de thèse
Yue MA MCF, Université Paris-Sud	Encadrante
Gaëlle LORTAL Ingénieur de recherche, Thales Research and Technology	Encadrante

Acknowledgemnts

I would like to thank to all those that have been involved in this thesis and that have supported me during the last three years. There has been more people to whom I am thankful, than those I can mention in a small thank note.

Nevertheless, I specially need to thank my thesis director Chantal Reynaud and my supervisors Gaëlle Lortal and Yue Ma for their deep involvement, awareness, friendship and guidance on this work. I would also like to extend my gratitude to Claire Laudy, who was my supervisor during the first year and offered me her support during the rest of the thesis.

The work could not have been done without the involvement of engineers, technicians and managers that provided insight, analysis and crucial information about avionics and systems architecture, both in Thales TRT and Thales Avioncis, many thanks to Laurent Laudinet, Amira Drhimi, Guillaume Ruban, Laurent Pepin, Stéphane Brisson and Jerôme Venant.

Many thanks to the teams in both my labs: LRASC (Analysis & Reasoning in Complex System Lab) at Thales R&T, and LaHDAK (Large-scale Heterogeneous DAta and Knowledge) in LRI (Université Paris Sud) for their welcome and finally, but not the least important, to my family who undoubtedly played the most important role in this journey.

To all my friends, specially those I made in Paris, have my deepest sense of appreciation.

Abstract

In this thesis we explore the problem of signature analysis in avionics maintenance, to identify failures in faulty equipment and suggest corrective actions to resolve the failure. The thesis takes place in the context of a CIFRE convention between Thales R&T and the Université Paris-Sud, thus it has both a theoretical and an industrial motivation.

The signature of a failure provides all the information necessary to understand, identify and ultimately repair a failure. Thus when identifying the signature of a failure it is important to make it explainable.

We propose an ontology based approach to model the domain, that provides a level of automatic interpretation of the highly technical tests performed in the equipment. Once the tests can be interpreted, corrective actions are associated to them.

The approach is rooted on concept learning, used to approximate description logic concepts that represent the failure signatures.

Since these signatures are not known in advance, we require an unsupervised learning algorithm to compute the approximations. In our approach the learned signatures are provided as description logics (DL) definitions which in turn are associated to a minimal set of axioms in the A-Box. These serve as explanations for the discovered signatures, thus providing a glass-box approach to trace the reasons on how and why a signature was obtained.

Current concept learning techniques are either designed for supervised learning problems, or rely on frequent patterns and large amounts of data. We use a different perspective, and rely on a bottom-up construction of the ontology. Similarly to other approaches, the learning process is achieved through a refinement operator that traverses the space of concept expressions, but an important difference is that in our algorithms this search is guided by the information of the individuals in the ontology.

To this end the notions of justifications in ontologies, most specific concepts and concept refinements, are revised and adapted to our needs.

The approach is then adapted to the specific avionics maintenance case in Thales Avionics, where a prototype has been implemented to test and evaluate the approach as a proof of concept.

Resumé en Français

Dans cette thèse, nous étudions le problème de l'analyse de signatures de pannes dans le domaine de la maintenance avionique, afin d'identifier les défaillances au sein d'équipements en panne et suggérer des actions correctives permettant de les réparer. La thèse a été réalisée dans le cadre d'une convention CIFRE entre Thales Research & Technology et l'Université Paris-Sud. Les motivations sont donc à la fois théoriques et industrielles. Une signature de panne fournit toutes les informations nécessaires pour identifier, comprendre et réparer la panne. Son identification doit donc être explicable. Nous proposons une approche à base d'ontologies pour modéliser le domaine d'étude, permettant une interprétation automatisée des tests techniques réalisés afin d'identifier les pannes et obtenir les actions correctives associées. Il s'agit d'une approche d'apprentissage de concepts permettant de découvrir des concepts représentant les signatures de pannes. Comme les signatures ne sont pas connues a priori, un algorithme d'apprentissage automatique non supervisé approxime les définitions des concepts. Les signatures apprises sont fournies sous forme de définitions de la logique de description (DL) et ces définitions servent d'explications. Contrairement aux techniques courantes d'apprentissage de concepts conçues pour faire de l'apprentissage supervisé ou basées sur l'analyse de patterns fréquents au sein de gros volumes de données, l'approche proposée adopte une perspective différente. Elle repose sur une construction bottom-up de l'ontologie. Le processus d'apprentissage est réalisé via un opérateur de raffinement appliqué sur l'espace des expressions de concepts et le processus est guidé par les données, c'est-à-dire les individus de l'ontologie. Ainsi, les notions de justifications, de concepts plus spécifiques et de raffinement de concepts ont été révisés et adaptés pour correspondre à nos besoins. L'approche a ensuite été appliquée au problème de la maintenance avionique. Un prototype a été implémenté et mis en œuvre au sein de Thales Avionics à titre de preuve de concept.

La thèse est organisée de la maniere suivante : Le manuscrit est composé de 7 chapitres incluant introduction et conclusion générales. Le premier chapitre fait office d'introduction et introduit la problématique, les contributions de la thèse et met en perspective le contenu du manuscrit ; le chapitre 2 constitue un état de l'art ciblé sur les ontologies, l'apprentissage dans les ontologies, et la modélisation de la maintenance avionique ; le chapitre 3 présente l'approche théorique retenue basée sur la découverte de concepts dans les ontologies formelles avec les algorithmes associés ; le chapitre 4 met en œuvre cette approche théorique

dans l'analyse de signature en maintenance avionique en lien avec une ontologie et à partir de fichiers de résultats de tests ; le chapitre 5 présente le prototype développé ; le chapitre 6 présente les expérimentations réalisées avec ce prototype et sa validation ; le chapitre 7 conclu le manuscrit en faisant un bilan des travaux réalisés et en avançant quelques perspectives. Dans le chapitre un, (Introduction) , on présente tout d'abord le domaine applicatif de la thèse, qui est la maintenance avionique en général et chez Thales en particulier. Parmi plusieurs centaines d'équipement que Thales répare, un équipement particulier, nommé « Elevator and Aileron Computer – ELAC », composé de six cartes électroniques et de deux alimentations, a été retenu pour la recherche. On explique ensuite comment les techniciens testent cet équipement à l'aide d'un banc de test, ainsi que la structure des fichiers « .AR » générés, dont chaque ligne teste une fonction spécifique (GO/NOGO). Ensuite on présente la problématique de la thèse qui est la découverte de signatures de défaillance, et la proposition d'actions correctives aux techniciens, ces propositions devant être impérativement explicables. L'ensemble de symptômes représente la signature de la défaillance, et ces signatures définissent des classes de tests, et les propriétés utilisées pour construire ces classes fournissent leur explication. Une fois les classes disponibles, elles seront utilisées pour classer les tests : pour chaque classe de test un ensemble d'actions correctives est associées, devenant les suggestions/propositions faites au technicien.

L'association de la signature trouvée et des actions correctives est donnée par les données historiques. De cette façon, chaque fois qu'un nouveau test arrive, il est d'abord analysé pour trouver des actions correctives si son type de défaillance existe déjà dans l'ontologie. Sinon, un nouveau type de défaillance est appris grâce aux informations contenues dans ce nouveau test, ce qui enrichit progressivement l'ontologie pour augmenter son exhaustivité. On a choisi de guider la découverte des types de défaillances par les tests disponibles, ce qui permet de réduire l'espace de recherche, et les types de pannes possibles peuvent être trouvés en temps réel. Il s'agit ainsi de découvrir des concepts dans une ontologie en LD d'une manière non supervisée, de telle sorte qu'ils représentent des ensembles intéressants d'individus, et que la définition de chaque concept découvert soit explicite.

Le chapitre deux, (Related Work) on présente une revue de littérature ciblée sur les ontologies et l'apprentissage automatique, toujours en lien avec la problématique de maintenance avionique. On introduit les ontologies, leur rôle dans la représentation des connaissances, leur lien avec le Web Sémantique, les langages RDF et OWL 1 et 2, puis on s'attarde sur les ontologies formelles avec les logiques de description (LD). Après on présente les grandes approches de la maintenance industrielle, permettant ainsi de définir le problème de la

maintenance avionique et la terminologie associée. Tout d'abord est présentée l'approche « Model Based Diagnosis (MBD) » développée par les communautés de l'informatique et de l'intelligence artificielle et fournissant un cadre pour le diagnostic de systèmes. En tant que tel, le diagnostic peut être considéré comme une forme de raisonnement abductif, où étant donné un ensemble d'observations, les raisons possibles des observations sont les diagnostics. Des stratégies visant à éviter d'explorer l'espace exponentiel des diagnostics et d'influencer les préférences sur les diagnostics récupérés ont été largement étudiées dans la littérature.

On s'interesse aussi par les approches de la maintenance basées sur des ontologies. L'objectif de la plupart des travaux rattachés à cette approche est de fournir un modèle formel qui tient compte de toutes les sources d'information hétérogènes nécessaires et disponibles, dans une représentation unique et bien définie, et de faire des inférences automatiques sur les connaissances représentées. Différents travaux sont étudiés et ils visent tous à fournir un modèle qui est une ontologie, qui tient compte de la terminologie appropriée utilisée dans le domaine, sur la base de normes et/ou de recommandations, mais malheureusement, aucun de ces travaux n'a été au-delà de l'implémentation même de l'ontologie. L'utilisation d'ontologie pour découvrir des signatures de défaillances nécessitant un mécanisme d'apprentissage associé à l'ontologie, On s'intéresse à l'apprentissage d'ontologie et l'apprentissage de concepts. L'apprentissage d'ontologie est un domaine multidisciplinaire qui vise la génération automatique d'ontologies. On distingue plusieurs approches : (i) les approches basées sur l'apprentissage à partir de textes, étroitement liée à l'analyse du texte et au traitement du langage naturel ; (ii) les approches basées sur la fouille de données liées (Linked Data Mining) reposent sur des techniques d'exploration de graphes qui nécessitent un graphe complet ; (iii) les approches de « crowdsourcing », une alternative à l'apprentissage entièrement automatique, dans laquelle les humains sont impliqués dans le cycle d'apprentissage, révisant le contenu dans une ontologie, ou y ajoutant explicitement du contenu ; et enfin (iv) les approches basées sur l'apprentissage de concepts dont le but est de découvrir des définitions de concepts. C'est cette dernière approche, l'apprentissage de concepts, plus adaptée à la problématique de recherche que on retient et étudie ensuite.

Dans le contexte des LD, les différentes méthodes d'apprentissage de concept présentent toutes des limitations dans le contexte de la maintenance avionique, notamment lié au fait qu'elles sont supervisées, et nécessitent des échantillons positifs et négatifs définis par rapport à des classes déjà définies, que l'on a pas ici.

A cause de cette limitation, on s'intéresse assez brièvement à l'apprentissage non-supervisé sur des données structurées.

Le chapitre trois, (The Approach : Situation Discovery), on développe notre

approche, qui constitue sa la principale contribution de la thèse. On s'intéresse ainsi à la capacité de déterminer quand un ensemble d'individus peut être distingué des autres, et chacun de ces ensembles, pour lequel on peut trouver une définition appropriée, est appelé une situation. Dans ce chapitre on développe l'approche retenu pour obtenir ces situations, une approche d'apprentissage non supervisée de concepts, basée sur la notion de raffinement de concept mis en œuvre par un opérateur spécifique. L'intuition qui sous-tend ce processus de raffinement est que, compte tenu d'un ensemble d'individus X, on veut construire des descriptions pour des sous-ensembles de X qui peuvent être représentés par un concept d'une ontologie en LD. Ensuite, une fois les descriptions construites, on fournit les moyens d'imposer une préférence à ces concepts de LD afin que seuls ceux qui sont intéressants soient récupérés. Ces concepts sont ensuite intégrés à l'ontologie LD originale, où ils peuvent être utilisés pour la classification. La construction des expressions conceptuelles repose sur les propriétés des individus dans X analysés, et par conséquent les définitions conceptuelles fournissent une explication sur les propriétés des individus qui sont pertinentes pour les distinguer du reste. Dans ce chapitre on présente en détail, et de façon formelle, l'approche de découverte de situation (Situation Discovery) et on explique comment on calcule les situations dans l'ontologie O spécifiée en ELO. Pour cela on définit formellement lun opérateur de raffinement (Refinement Operator), qui repose sur la notion de MSR (Most Specific Representative) utilisée pour la découverte de situation. Les différentes étapes permettant d'affiner un concept C guidé par une instance x et une ontologie O, sont présentées en détail, le résultat étant les concepts obtenus par une étape de raffinement de C. Ce chapitre constitue la contribution majeure de la thèse.

Dans le chapitre quatre, (Situation Discovery in Avionics Maintenance), on montre comment la relation entre les situations et les signatures de défaillance est établie, montrant ainsi comment l'approche générale présentée dans le chapitre précédent est adaptée à l'analyse des signatures en maintenance avionique. Plus précisément les deux fonctions principales que le prototype doit assurer pour aider le technicien dans le processus de diagnostic sont : 1) consulter la base de connaissances pour obtenir les mesures correctives suggérées, et 2) enrichir la base de connaissances grâce aux commentaires des utilisateurs sur la pertinence ou non des suggestions données. L'enrichissement de la base de connaissances nécessite non seulement d'envisager les éventuelles nouvelles actions correctives, mais aussi de découvrir de nouvelles signatures de défaillance.

Dans le chapitre cinq, (Prototype) on présente la mise en œuvre concrète de l'approche, par la réalisation d'un prototype, conçu et déployé pour appuyer le diagnostic de maintenance avionique. On rappelle brièvement le processus de diagnostic de maintenance de l'ELAC, l'origine des données, et le contexte de

Thales où se déroule le prototype implémenté . Après on présente les exigences du prototype, permettant pour le premier (Consult) de consulter la base de connaissances pour obtenir des suggestions, et pour le second (Acquire Feedback) associé au retour de l'utilisateur, permettant de mettre à jour/enrichir la base de connaissances avec de nouveaux fichiers « .AR » et de nouvelles actions correctives. On présente l'architecture du prototype, conçu et réalisé dans un environnement distribué (TRT cluster) permettant un traitement massif des données et un accès à distance. Les fonctions principales du système associées aux deux cas d'utilisation précédents sont détaillées et montré comment l'utilisateur final interagit avec l'outil au travers d'un interface homme-machine spécifique.

Dans le chapitre six, (Evaluation), on présente l'évaluation de l'approche par la mise en œuvre du prototype.On évalue la pertinence et le nombre de suggestions proposées par l'approche. L'hypothèse à valider étant que plus le grain de la base de connaissances est fin, plus on peut trouver des signatures de défaillance spécifiques, et ainsi minimiser le nombre d'actions correctives suggérées. Les résultats obtenus par une première série d'expérimentations relatives à la pertinence et la spécificité des suggestions, a permit d'obtenir une base de connaissances performante formée de 100 échantillons. Une seconde série d'expériences, a permit de vérifier qu'il y avait bien une évolution dans la base de connaissances, à mesure qu'on lui présentait plus d'information. Par évolution, dans notre travail ; on entend que la KB est plus finement grainé, et que ses réponses sont plus précises. Des expérimentations avec 25, 50 et 100 échantillons ont montré que c'est effectivement la tendance. On fournit des mesures et des analyses sur l'efficacité de la mise en œuvre, données par les temps de réponse et évalue le passage à l'échelle (scalability) de la solution proposée. Dans une dernière expérimentation, on compare notre approche à celle du du système DL-learner, un outil similaire utilisé pour découvrir des concepts, mais de façon supervisée.

Finalement, dans le chapitre chapitre sept (Conclusions and Further Work), on conclut le manuscrit en faisant un bilan synthétique de nos contributions, et présente ensuite quelques perspectives. La première concerne l'usage d'une logique de description plus expressive que la LD ELO utilisée dans la thèse qui ne possède pas les constructeurs de disjonction et de négation, ce qui aurait cependant pour conséquence une croissance exponentielle des concepts augmentant ainsi l'espace de recherche. La deuxième perspective est un traitement parallèle des différentes partitions de l'A-box suivi d'une fusion et consolidations des résultats obtenus selon une stratégie de type Map-Reduce. La troisième perspective est d'introduire une distance explicite entre signatures permettant à considérer de nouveaux individus dans les signatures et par la même améliorer les suggestions fournies. Enfin la dernière perspective concerne l'extension du modèle à d'autres

équipements que l'ELAC considéré dans la thèse, ainsi qu'à d'autres tâches de maintenance.

Contents

List of Figures

Acronyms

CIFRE Convention Industrielle de Formation par la Recherche. 1, 155

CMM Component Maintenance Manual. 4

DL Description Logic. viii, 1, 7, 8, 11, 14–16, 26, 31, 32, 37, 38, 41, 44, 45, 153

ELAC Elevator and Aileron Computer. xi, 3–5, 7, 91, 93, 155, 156

FMEA Failure Mode and Effects Analysis. 18, 19

FOL First Order Logic. 14, 15, 17

HTML Hyper Text Markup Language. 13, 14

IMS Intelligent Maintenance Systems. 18

LRU Line Replacement Unit. 2, 3, 7

MBD Model Based Diagnosis. 11, 16, 17

MDC Mutually Disjoint Concepts. 26

ML Machine Learning. 20

msc Most Specific Concept. 34, 35

NASA National Aeronautics and Space Administration. 18

OWL Ontology Web Language. 11, 13, 14, 18, 22, 26, 96

RDF Resource Description Framework. 14, 18, 21, 26

SPSC Spare Parts Supply Chain. 18

SRU Shop Replacement Unit. 2, 94

TAMO Thales Avionics Maintenance Ontology. viii, 68, 74, 79, 91, 154

TRL Technology Readiness Level. 155

TRT Thales Research & Technology. 155

W3C World Wide Web Consortium. 13

XML Extensible Markup Language. 14

Glossary

Failure Signature The necessary and sufficient information about a failure that establishes a strong relationship between failure characteristics and failure mechanism.
NOTE 1 This necessary and sufficient information can include emission microscopy results, morphology data, test data, IV-curves, environmental history, etc., and therefore can be either electrical or physical in nature.
NOTE 2 The scope of application can be time-based, lot-based, package-based, design-based, etc. [JEDEC, 2018] . 3, 5–7, 10, 82

ISO 14224 The ISO 14224 is a standard for petroleum, petrochemical and natural gas industries. Collection and exchange of reliability and maintenance data for equipment. 18

ISO 15926 The ISO 15926 is a standard for data integration, sharing, exchange, and hand-over between computer systems. 18, 19

Signature Analysis (SA) The process of assigning the most likely failure mechanism to a countable failure based on its unique electrical failure characteristics and an established physical analysis database for that mechanism. NOTE
The objective is to reduce the number of comprehensive failure analyses. [JEDEC, 2018] . 65

Test-Bench A special unit designed to test equipment and verify its functions. 3, 4, 68, 94, 95, 154

Chapter 1

Introduction

This thesis takes place in the context of a Convention Industrielle de Formation par la Recherche (CIFRE) between Thales R&T and the Université Paris-Sud, thus it has both: a theoretical and an industrial motivation. The thesis explores the problem of signature analysis in avionics maintenance, and proposes an ontology based approach rooted on concept learning, to approximate Description Logic concepts that represent the failure signatures.

In avionics, a *failure* denotes the loss of the ability of a device to meet the performance specifications that it was intended to meet. The *signature* of a failure in this context, represents all sufficient and necessary information that is strongly related to the failure mechanism [JEDEC, 2018]. Knowing the signature allows to better understand the failure, to predict the behavior of the equipment, and to repair it. In equipment diagnosis, the manifestation of a failure is put down to the bad interaction between some of its components, which are grouped into functional chains. Identifying these functional chains and the faulty components involved, provides the signature of the failure. In model based diagnosis this is known as a diagnosis [De Kleer and Kurien, 2003], and the model aims to predict the intended behaviour of the modeled system. In our case we do not count with such a model. Instead, we are given tests that report the status of the functions in the equipment, and corrective actions made by the maintenance technicians. In this context, our task is to provide an approach and a tool to support maintenance technicians with suggestions on the corrective actions to be taken, given an input test.

1.1 Avionics Maintenance

In avionics maintenance complex and time-consuming actions have to be taken to return a faulty equipment to a fully functional state. The objective of our

work is to support maintenance technicians in their activity with cues of actions or full actions to perform, in order to repair faulty equipment.

During the maintenance processes in avionics, when an equipment is found faulty on an aircraft it is to be repaired or replaced. Avionics maintenance is divided three levels:

- Level 1 (On aircraft) : These are the maintenance actions performed on aircraft. When an aircraft arrives to an airport, the status of its systems and equipments are tested and maintained in place. If a failure exists, the faulty equipment is identified, called a Line Replacement Unit (LRU) and is replaced by another unit with the same capabilities. The main objective of the maintenance action in this stage is to return the aircraft to a fully functional state as soon as possible to minimize the time it stays on ground.
- Level 2 (In Shop) : The LRU found faulty by maintenance level 1 are sent to a specialized shop for investigations and maintenance. Each LRU is composed of several Shop Replacement Unit (SRU), and the task of the technicians is to identify the faulty SRU.
- Level 3 (In Shop) : The third level of maintenance also takes place in the repair workshop, where provided a faulty SRU the technicians find the components that cause the failure to repair the equipment.

Figure 1.1 depicts the three levels in avionics maintenance. Our work is concerned

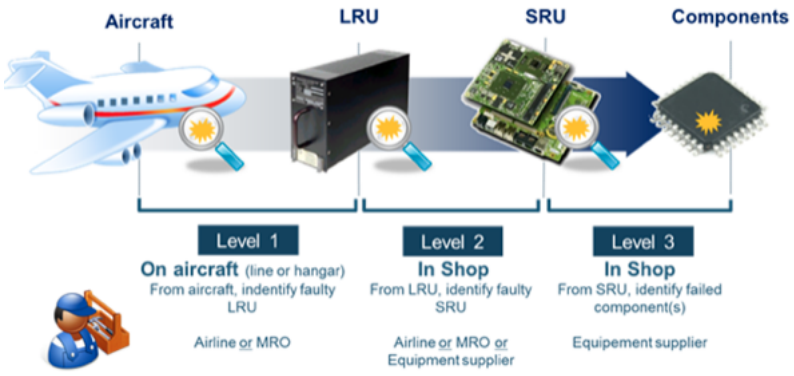


Figure 1.1: The different levels of avionics maintenance. Image credit: Guillaume Ruban, TAV.

with maintenance levels 2 and 3, that is *in shop* maintenance.

To determine the proper corrective actions when a faulty equipment presents a failure, the technicians need to gather all the necessary information that allows

them to identify the failure cause. The technicians make investigations and run tests to evaluate the state of the equipment. All the necessary and sufficient information that is strongly related to the mechanism of the failure is called the *Failure Signature* [JEDEC, 2018] . When a technician makes investigations and evaluates the results of the tests, she/he is looking for the symptoms that are in line with the signature of the failure to isolate the problem, and establish the proper repair actions for the equipment.

This process is time-investing, automatically establishing the most probable actions is useful to shorten the examination and repair time, thus gaining in efficiency and lowering the costs.

Out of the hundreds of types of equipment Thales Avionics repairs, the Elevator and Aileron Computer (ELAC) was selected by Thales for this study because of the availability of the results, the high frequency of maintenance and the complexity of its diagnosis. The ELAC is a Line Replacement Unit (LRU) of which several versions exist (ELAC A, ELAC B, ELAC Téléchargable). The ELAC B used in our approach comprises six boards and two power units. Each of the boards has hundreds of components and in case a failure is found, the right component to be replaced has to be found. Figure 1.2 presents a diagram of an ELAC B computer.

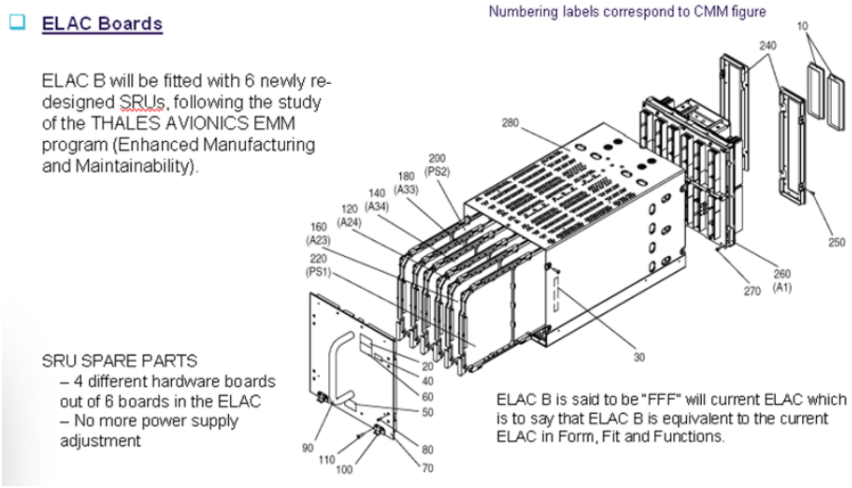


Figure 1.2: A diagram of an ELAC B computer, extracted from the product sheet specification. The figure shows the 6 internal boards and the schematics of the product.

In order to diagnose the problem, the technicians test the ELAC in a special unit called a Test-Bench, which exhaustively tests the equipment functions. Once the tests are run, it is up to mechanic experts to interpret the results,

determine the possible components of the equipment involved in the failure, and the repairs/replacements to be done. For a maintenance process it is difficult to establish *a priori* what are the actions to be taken to return the equipment to a fully functional state. Each function in the equipment is associated to a functional chain, which is a logical circuit comprising tens of components. Once the logical components are identified, their physical locations in the equipment need to be precised to proceed with the repair.

The main source of information for the ELAC technicians to make the repairs, are the tests results that come out of the Test-Bench. These are presented as an .AR (All Results) file, which is divided into chapters, sections and subsections. Each subsection contains up to hundreds of individual tests, one per line. Each line of the .AR file represents a test on a specific function with the sanction GO or NOGO, which tells us if the test was passed. An extraction of an .AR file and its main structure is shown in Figure 1.3.

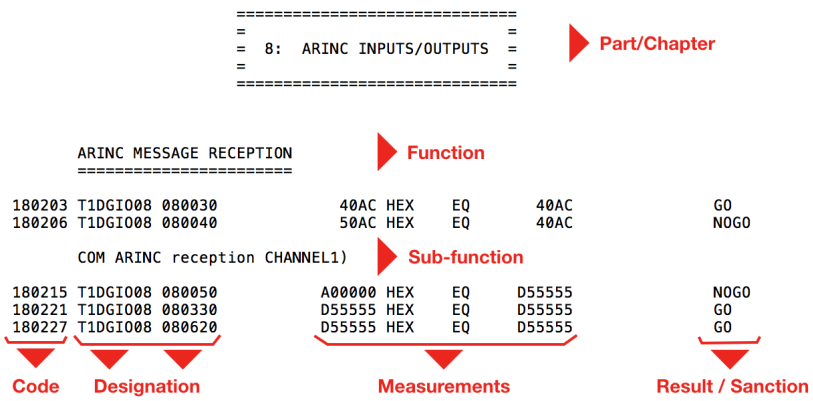


Figure 1.3: Extract of an .AR file. The figure shows the first lines of chapter "8: ARINC INPUT/OUTPUTS", with the function and sub-functions being tested, where two tests present the sanction: "NOGO".

Besides his expertise, to interpret these results the technician relies on a Component Maintenance Manual (CMM), and a visualization tool to identify the physical location of the components.

An additional complication in this problem is the non-deterministic relation between the failures and the corrective actions: a specific failure can be detected by several distinct tests, and distinct tests might have the same corrective action.

1.2 Problematic

Equipment diagnosis is a complex task since on the one hand, the interpretation of the results requires high expertise and experience, and on the other hand the possible corrective actions can be exponential with respect to the number of components.

To determine the proper corrective action, a good understanding of the failure and the test results is needed due to the following reasons. First, the technician needs to know why an action is proposed. He can not blindly change a suggested component, the causes and the effects of the failure need to be clearly determined. Second, once the maintenance process ends, a fully functional and certified equipment is returned to the client. For safety reasons, a certification that the equipment provides the expected services is needed. In this process, the mechanism of the failure has to be clearly established, to explicitly show the failure causes and their exhaustive repair. Third, the explanations influence the adherence and confidence of the technicians in the system. Finally, besides the direct use for the technicians, the explanation for a failure allows to identify it, design tests to detect it, define ways to solve it, and certify equipment which is free of the failure. Thus the explanation is of great value.

Thus, automatizing the diagnosis process requires not only to be able to interpreted the test, but also to provide explanations on the proposed suggestions.

Confronted with this problem, our goal in this thesis is to:

Support avionics maintenance by discovering Failure Signatures and proposing corrective actions, in such a way that the suggestions are explainable.

This problem has to be solved by taking into account the following constraints.

Sparse data A particular aspect of our problem is that the available data is not sufficient for an automatic learning system based on statistical analysis. Classic machine learning techniques aim to approximate a function f which describes the behavior of a real world process, through a hypothesis h, where the accuracy of h to approximate f relies on the amount of data presented to the model. In our case we count only with 150 samples of .AR files and their corrective actions, therefore we can not rely on techniques that require large amounts of data to properly model a process.

Sparse knowledge We have at our disposal a set of 150 test results of a specific equipment (ELAC B) and the associated corrective actions. The correlation between the actions and the tests is given by historical data gathered from the maintenance workshop.

To assign actions for a test result, technicians follow maintenance manuals

stating the specific functions of components, the functional chains involved, and their physical location.

There is no explicit model for this manual process. There exist manuals, visualization tools, and guidelines on the procedures. But a model relating the test results to the components to be replaced is not available. To obtain such a model we count only with the test results and the corrective actions in the historical data.

Number of suggested repair actions The correspondence between the set of NOGO lines in an .AR file and the components that need to be replaced is not evident. Note that the combinations of the functions to be checked in an .AR file is exponential with respect to the number of NOGO lines. For example, a file containing 4 lines with a sanction of NOGO, has $2^4 = 16$ combinations. To solve the failure, all the combinations might need to be considered in the worst case. A file with 20 NOGO lines could easily have associated more than 1 million function combinations to check. In the sample data we have found files with 40, 50, 100 and thousands of lines that present the result NOGO. This is why equipment diagnosis is complex.

Providing hundreds of results as possible repair actions is useless. First, technicians must not have to choose from a large number of suggested actions. Each action chosen triggers a new test-bench session that can last for hours. Secondly, the number of failing components is often low in reality. So in each repair, the suggested components to be replaced should be minimized.

Provide an explanation for the given results As outlined in section 1.2, explainability of the system is important for technicians and security certification.

1.3 The proposed solution

In order to provide suggested corrective actions to the technicians, given an input test, first we have to be able to classify the test. But in our problem we do not count with classes of tests.

To find these classes, we look into the properties of the tests for those that represent the symptoms of a failure. A failure is given by the replaced faulty components. Thus, all those properties that allow to distinguish a set of tests that are solved the same way, represent the symptoms of the failure. This set of symptoms represent the Failure Signature. These failure signatures define classes of tests, and the properties used to construct these classes provide their explanation. Once the classes available, we use them to classify the tests, where for each class a set of corrective actions are associated, which become the suggestions to the technician.

From the historical data, we find that some different tests can be associated to

a same corrective action. Therefore, it is not a unique test result that characterizes a failure, but rather sets of tests. These sets are not known in advance, and therefore we first must discover them. Moreover, the proposal of actions must be explained because technicians have to know the reason for the actions and due to the requirement of security certification as we have already mentioned above. To this end, we provide the definitions of the signatures, that play the role of explanations.

To discover the definitions of the failure signatures, we can rely on the following pieces of knowledge:

- the files containing the test results, one file corresponding to one faulty ELAC/LRU.
- the historical knowledge in avionics maintenance, which is the established correspondence between test result files and the correction actions.

We propose to model the tests and their results in an ontology to enable automatic reasoning over it. Indeed, AI techniques allow to formally represent the knowledge of a domain and perform automatic processing over this knowledge, like reasoning. In this thesis, we propose to use Description Logic (DL) as the underlying language for the ontology, since it is a formal representation, for which the reasoning results are traceable and explainable. The proposed approach allows to enrich this ontology with discovered new Failure Signatures and their definitions incrementally.

Once the discovered signatures are made available, when a new, unseen, test file is presented to the system we use the ontology to determine the type of failure to which it belongs. Thus the ontology provides a level of automatic interpretation of the test files. Then, corrective actions can be provided to the technicians as suggestions.

The association of the signature found and the corrective actions is given by the historical data. In this way, each time a new test arrives, it is first analyzed to find if it corresponds to a failure signature already known by the knowledge base. Otherwise, a new type of failure signature will be learned, thanks to the information this new test carries, thus incrementally enriching the ontology to increase its completeness.

In this DL setting, tests are individuals, and failure signatures are concepts. Consequently, our objective is first to discover these concepts, corresponding to sets of individuals, and secondly associate these concepts to corrective actions.

There exist several approaches to construct and discover concept definitions, most of which rely on positive and negative samples with respect to the categories already defined. As stated before, we do not know these categories in advance, and we cannot used these approaches.

The discovery of the concepts in a DL setting can be achieved even if we do not have a large amount of data, therefore we do not rely on large amounts of historical data. In principle, we could provide all possible combinations of all available properties of the tests in the model, and use them to formulate the categories of the tests. However, the possible concepts we can construct in this way grows exponentially, and might even be infinite if the length of the concept is not limited. To overcome the problem of searching through all possible concept expressions, in our approach we guide the discovery of the failure types by the available tests. In this manner the search space is limited and the possible failure types can be found in tractable time.

From the point of view of the techniques that enable us to obtain these signatures in a DL setting, our objective is to:

Discover concepts in a DL-Ontology in an unsupervised fashion, in such a way that they represent interesting sets of individuals, and the definition of each discovered concept is explicit.

Note that, for each discovered set of individuals, several concept expressions might correspond. Those that are the shortest most specific will be preferred to avoid useless expressions [1].

1.4 The main contributions

- We identify a new problem in DL setting, which is to discover new concepts that are unknown in advance, named *unsupervised concept discovery.*
- We propose the algorithms to the problem of unsupervised concept discovery. The main algorithm is presented as a refinement operator which provides a full control over the refinement process. We show the operator is sound and complete, in that it provides an access to all relevant concept definitions that can be constructed to describe the different sets of individuals in the ontology.
- We provide an explainable method to suggest avionics maintenance actions based on the discovered concepts.
- We provided a prototype used in a Thales division to support avionics maintenance diagnosis, where given a test file as input the suggested corrective actions are returned. The prototype uses information from Thales Avionics (Toulouse, Châtellerault - France) with whom we have

[1] In this context *useless* is understood as expressions that do not contribute in the number of tests captured by the description.

developed the model and selected the data. The final implementation is to be hosted in Thales Research & Technology (Palaiseau - France) using a BigData platform, allowing massive processing and remote access.

- We test and evaluate the approach applied to the avionics maintenance, and provide discussions on the found results.

1.5 The thesis structure

The rest of this document is organized as follows:

Given the context and problematic presented in this introduction, we continue by presenting the state of the art on tools and techniques that allow to solve similar problems.

Chapter 2 presents the related work. We start by introducing ontologies and their role in knowledge representation and knowledge integration. We explain their relation with the semantic web and highlight some challenges of implementing ontologies based systems. Next, we present approaches to model industrial maintenance using ontologies and their relation to our solution. Finally, we introduce ontology learning, for which concept learning is a specific case. We explore the current techniques for concept learning, and outline the limits they present for solving our problem.

Given the research works related to our problem, we next present our solution.

Chapter 3 In this chapter we present our approach which allows us to overcome the technical challenges for concept discovery, under the constraints and conditions stated in our problem definition. A preliminary part introduces the necessary notions and definitions used in our approach. Then this section presents the definitions, the algorithms and the proofs of soundness and completeness for our approach. Our solution is driven by the analysis of one sample at the time to guide the search over the space of concept expressions. This search is realized through the iteration of a refinement operator. In this chapter we specify how the operator generates concept expressions, what are the properties of the discovered definitions and how they can be used to distinguish sets of individuals that share some semantic features in the ontology.

The approach presented in chapter 3 is of general use, thus the next step is to apply it to the avionics maintenance case.

Chapter 4 In this chapter we establish the relation between the *situations*

found by the approach in chapter 3 and our goal of discovering *Failure Signatures* in avionics. To this end, we pose the problem of finding the signatures as finding situations in an ontology. Once the signatures are available, we specify how to obtain the suggested corrective actions for a given .AR file. Finally, we show how to integrate the new knowledge in the ontology. The algorithms and the specifications on what the computations yield are given in this chapter.

Once the design of the algorithms and their specifications are given, we proceed to implement them to develop a prototype.

Chapter 5 In this chapter we detail the implementation process. We present the requirements of the prototype, the context on which it is intended to be used, the architecture of the system, the two main functions: Consult and Feedback, and the different versions of the prototype.

With the prototype in place and the specifications of the approach clearly established, we proceed to make experiments and evaluate our approach.

Chapter 6 This chapter shows the evaluation of the approach, by performing different tests using the prototype and analyzing how well these results comply with the criteria defined for a desirable solution. Several variations in the prototype are made considering: different amount of analyzed data, two different implementations and two different reasoners. A final evaluation considers the feedback from the users on the usefulness of the approach.

Chapter 7 This chapter presents our conclusions and guidelines for further work.

Finally the **Annex** provides details on the data sources, the user manual, and a list of by products of the thesis.

Chapter 2

Related Work

In this chapter we describe techniques related to our problem, that is ontology based concept discovery applied to the maintenance domain.

We start by presenting ontologies. We defined them in a most general sense. We specify their role in computer science. One of their most notorious applications is related to the semantic web, where the Ontology Web Language (OWL) is designed as formalism to annotate content on the web and make knowledge interoperable. These features are important to understand the capabilities, potential and relevance of ontologies and knowledge based systems. Secondly, we introduce Description Logic, the formalism used in this thesis to represent ontologies.

In Section 2.2, we describe approaches for diagnosis and maintenance. First we provide a brief overview of Model Based Diagnosis (MBD), where the problem of defining the causes of failures in equipment is studied. In this setting, given a model of a system, the goal is to use the model to predict the behaviour of the system and provide diagnosis. We explain the similarities, limitations and ideas that intersect with our approach. Next we present an overview of approaches that envisage and study their application in industrial maintenance. Indeed, a model is to provide an abstraction of a real world process. It allows the study of the process, to understand its mechanism and to predict its behaviour. Ontologies provide the means to model industrial processes and systems, in a conceptual way. Because of their features, ontologies have been considered to enable knowledge management in the industry.

Because knowledge discovery involves automatic processing of information, in Section 2.3 and Section 2.4, we provide a general introduction to machine learning, supervised learning, unsupervised learning and clustering. Nowadays machine learning is largely associated with neural networks, statistical analysis and deep learning. These are techniques based on functions that operate over large

amounts of unstructured data which is represented as vectors and matrices of real numbers. In a symbolic setting, like ontologies, the information is discrete and highly structured, which requires a different approach. This is why "classically", knowledge representation and machine learning are regarded as two separate branches of artificial intelligence, where many approaches to combine them exist, aiming to take advantage of both areas [Dietz Saldanha et al., 2018]. Our work also falls in the category of such hybrid approaches, with an emphasis on knowledge representation. It is out of the scope of this thesis to provide an extensive insight into machine learning, but a context to properly position our work, with respect to these approaches is required.

The construction of a model can be very demanding: the knowledge of experts in the domain is needed as well as skilled ontology engineers. The goals for which the model is constructed influence its design and the inferences obtained should be inline with the expected results. These factors associated with the needed resources, impose a constraint on the development of knowledge based systems. A study and methodology for estimating such costs can be found in [Simperl et al., 2012]. We introduce an overview of ontology learning which aims to achieve the challenging goal of automatic construction of ontologies, from which concept learning is a particular case. Concept learning is concerned with automatic construction of concept definitions and general concept inclusions. The main techniques of these approaches are based on least common subsumers, normalization, refinement operators (as in inductive logic programming) and in bisimulations. Some of these approaches provide theoretical studies of the problem, and others go as far as possible to implement software tools. A state of the art tool for supervised concept learning is DL-Learner, based on refinement operators. Our work is inspired by the approach of DL-Learner whose main features are therefore explained in detail.

We finish this chapter with an outline of the tools and techniques in knowledge clustering and graph mining. The motivation behind this is to take advantage of the extensive research and techniques for clustering. In the same line of thought, pattern recognition identifies similar patterns present in the data. Since the information in ontologies can be represented as a graph, an overview of the techniques to mine common subgraphs is presented.

2.1 Ontologies and Description Logics for Knowledge Representation

2.1.1 Ontology: Definition, Role, Representation Languages

The term ontology has its origin in the greek words : *onto* (being; that which is) and *logos* (logical discourse, study of). It can be understood as the "study of being" or "the study of things that exist". As such, it is a very abstract and broad notion that can be defined differently depending on the domain it is being used: philosophy, linguistics, computer science, etc. Ontologies are intrinsically related to the idea of conceptualization and naming things that exist. They study concepts that directly relate to being, in particular becoming, existence, reality, as well as the basic categories of being and their relations.

In computer science, ontologies are regarded as formal representations of knowledge, although there is no general agreement on what characteristics a representation should have to be called an ontology [Lehmann and Völker, 2014]. Depending of the point of view and the domain being modeled, their definitions and characteristics can vary, they can be viewed as: dictionaries, thesauri, taxonomies or highly axiomized formalizations. In this thesis, we are interested in formal ontologies.

Ontologies play a central role in data and knowledge integration: by providing a shared schema, they facilitate query answering and reasoning over disparate data sources. Ontologies provide a computer readable interpretation of multiple sources, e.g. the content of the web. This allows to have a common language for describing the web, where ontologies can refer to each other and where new ways to link information (e.g. through annotations) are available. As such, ontologies are essential to the semantic web. The Ontology Web Language (OWL) defined by the World Wide Web Consortium (W3C) used in the semantic web, specifies the syntactic restrictions to construct the ontologies [Consortium et al., 2012]. This specification standardizes the format of the ontologies, facilitating their access, interoperability and shareability.

In this thesis we do not explore or discuss the details and research on the semantic web. We only highlight how it benefits from ontologies and formal descriptions, and how this has made ontologies more popular, in order to show the scope of applications that can benefit from our approach.

A main motivation for the use of ontologies in the semantic web comes from the limits that the content on the world wide web presents. The Hyper Text Markup Language (HTML) was designed to link documents and tag the content of web pages. The HTML documents are then interpreted by a client (typically a web browser) which provides a representation and access to their content. The

limits of HTML lay in that the tagged content lacks of meaning. We can tell that a specific web site contains text, images, links, etc., and we can even associate a description to these items, but we can not explicitly tell what this information is about: a person, a company, an event, etc. To provide this additional layer of information, and enable automatic access to it, the "semantic web" came into place [Berners-Lee et al., 2001].

An ontology compliant with OWL can be serialized in different file formats: Extensible Markup Language (XML), Turtle, Resource Description Framework (RDF). These files can be used to link the resources used in the web with the terms in the ontology, associating the meaning the ontology carries to the content in the web.

Once an OWL ontology is available, it has to be interpreted. This interpretation is given by the semantics applied to the represented knowledge. Two main semantics are given to interpret OWL ontologies: the direct semantics and the RDF semantics.

The Direct Semantics of an OWL 2 (latest version of OWL) ontology is compatible with the model theoretic semantics of the SROIQ description logic (a fragment of FOL with useful computational properties) [Consortium et al., 2012]. This semantics assigns meaning directly to the structures in the ontology. The advantage of this close connection is that the extensive description logic results in the literature and implementation experience can be directly exploited by OWL 2 tools, extending the reasoning capabilities of DL to OWL knowledge bases. Some syntactic restrictions are required on these ontologies (e.g. transitive properties cannot be used in number restrictions), which are also known as OWL 2 DL ontologies. Section 3.1.2 introduces Description Logics, that provide the logical foundation for OWL DL, and which is the knowledge representation language chosen in this thesis.

On the other hand, RDF-based semantics interpret the ontology as a graph. This semantics is fully compatible with RDF semantics. Every OWL 2 ontology can be serialized as an RDF graph, and thus no restrictions are imposed on these representations. These ontologies are known as OWL-Full ontologies. A drawback on this setting is that the full graph of the ontology has to be materialized, thus making all the knowledge implicitly encoded in the ontology explicit.

2.1.2 Description Logics

Description Logics (DLs) are a family of First Order Logic (FOL) formalisms [Baader, 2003a] that represent knowledge in terms of concepts, individuals that belong to these concepts and the relations between them. DLs are equipped with formal, logic-based semantics, and provide reasoning as a central service:

reasoning allows to infer implicitly represented knowledge from what is explicitly expressed in the knowledge base.

Description Logics have their roots in semantic networks [Lehmann, 1992, Quillian, 1967, Sowa, 1987] and frames [Minsky, 1974], which are early formalisms for representing knowledge and which allow automatic reasoning. Even though they present differences, both are structures that represent knowledge as concepts (sets of individuals) and relations between them. Their implementation was initially intended for artificial intelligence, in providing a model on how humans reason, and in machine translation as one specialized area of application. A drawback of such approaches was the lack of clear and well defined semantics. A recognition that frames could be given formal semantics through First Order Logic came in [Hayes, 1981], where the sets of individuals are unary predicates and the relations between them are binary predicates. Nevertheless, such representations did not require all the machinery of FOL to capture the semantics of frames. Furthermore, First Order Logic is undecidable, meaning that for some decision problems, an answer can not be given in finite time, thus the need to restrict to a fragment of the language.

In logic, the predicate symbols provide the vocabulary to express facts about the world and the syntax of the language allows to combine these symbols in words or well formed formulae with well defined semantics. Typically the formalism provides unary (negation) and binary (conjunction, disjunction, implication, equivalence) connectors, which can be combined with quantifiers (existential restriction, universal restriction) to form valid words over the sets of symbols. These comprise the constructors of the language. The constructors and the values that can be passed as parameters for the predicates, define the expressivity of the corresponding language.

There is a trade-off between the expressivity of the language and the complexity of the reasoning procedures: the more expressive the DL language, the more complex reasoning becomes. On the other hand, less expressive DL languages might have very efficient decision procedures, but might not be rich enough to properly represent the necessary concepts and relations in the target domain.

Thus depending on the constructors allowed by each language, a computational complexity to reason over the represented knowledge is associated.

Extensive research and efforts have been made in providing procedures that are decidable and tractable (they should terminate, provide an answer in finite time, and preferable in polynomial time, i.e. avoid exponential blowup) for the different families of DLs.

Thus, Description Logic comprises a family of FOL languages for knowledge representation, with well defined semantics, associated complexity results and for which efficient algorithms for reasoning exist.

As mentioned before, the different DL languages are distinguished by the constructors they provide. The selection of a language depends on the application domain and on the associated complexity. The description logic language $\mathcal{AL}$ (which stands for Attributive Language) is considered as the reference language for DLs, from which modifications and extensions exist. In this thesis we are interested with the DL language $\mathcal{ELO}$ as defined in [Kazakov et al., 2012].

The language $\mathcal{ELO}$ is different from $\mathcal{AL}$ in that it does not allow negation nor disjunction. However it allows to refer to individuals in complex concepts thanks to nominals. In this thesis we have chosen $\mathcal{ELO}$ because it is the simplest DL language that allows to describe the avionics domain. As stated before, the more expressive the language the more expensive it is to reason over the represented knowledge. Allowing negation and disjunction can easily lead to exponential blowup and have not been considered as strictly necessary to describe our domain. Thus we do not intend to use unnecessarily expressive languages, unless required. Our goal is to provide an approach that can solve a basic case, and establish the cornerstone for extensions to more expressive DLs, if it is considered necessary. Under these considerations, a simpler language $\mathcal{EL}$ (which considers only conjunction and existential restrictions) could suffice, but we find that it is not expressive enough for our problem. Details about the maintenance tests made in avionics are expressed as individuals. If we expect to make distinctions between the tests, nominals have to be allowed in the concept definitions. Therefore, after an initial analysis of our data, the minimal and sufficient DL family for our approach is $\mathcal{ELO}$, detailed in chapter 3.

Regardless of the chosen language, in Description Logics a distinction between the terminological knowledge and the assertional knowledge is made. The terminology is given by the vocabulary of the ontology, where the names of the concepts, the names of the relations and their definitions (if they are formed of sub concepts, of intersections of other concepts, the domain and range of a relationship) are given. This part of the ontology is called the T-Box. On the other hand, we have the assertions, which are statements of facts about the world expressed in terms of the T-Box. A T-Box and an A-Box all together are known as a Description Logics knowledge base, which in turn is also called a DL-Ontology.

2.2 Approaches for Diagnosis and Maintenance

2.2.1 Model Based Diagnosis

Model Based Diagnosis (MBD) is a framework for system diagnosis developed by the Computer Science and Artificial Intelligence communities. A system

is provided in terms of its components and their interaction. The model of a system allows to predict its expected behaviour. When an observation, typically a measurement, about the state of the system differs from its predicted behaviour, it is called a symptom of a failure. A failure is put down to the bad interaction of a set of components in the system. The intuition is that when a symptom which depends on the interaction of a set of components is detected, it can not be the case that all the components involved are working properly. Each such set of components is called a conflict set for the observed symptom. A minimal conflict set, is a conflict set for which no other proper subset is a conflict set. These are desirable because we would like to identify as precisely as possible the causes for the detected failure, that is the faulty components. A diagnosis in this setting is a hypothesis on how the system differs from its model. The size of the diagnosis space is exponential in the number of components of the system. The model of a system and the notions of conflict sets and diagnosis can be formally defined through First Order Logic (FOL), where a system is a triple composed of: the system description, the components of the system and the observations. The diagnoses are found through logical inferences that make the proposed diagnoses, the observations and the description of the system consistent. As such, diagnosis can be seen as a form of abductive reasoning, where given a set of observations, the possible reasons for the observations are the diagnoses.

Strategies to avoid exploring the exponential space of diagnoses, and imposing preferences on the diagnoses retrieved have largely been investigated. An introductory document and the main techniques for MBD can be found in [De Kleer and Kurien, 2003].

Similarly to our problem, we aim to point out the faulty components that may return the equipment (system) to a fully functional state. We also rely on First Order Logic (FOL) formalizations to model our domain. But we do not count with an explicit model of the system. We do not have the explicit relation of the sequence and interdependence of the components in the equipment, and we are not modeling the equipment itself. Instead, we model the test results that report the status of functions in equipment and corrective actions having been made by the maintenance technicians. This knowledge is formalized using DL ontologies and the possible diagnoses are given by the discovered signatures for the failures.

We next present approaches that leverage ontologies for industrial maintenance processes.

2.2.2 Ontologies in Approaches for Maintenance

There exist several works that research the applicability, advantages and considerations of using ontologies to model maintenance, and support the overall process. The main objective in these works [Karray et al., 2012, Regal and Pereira, 2014, Ebrahimipour and Yacout, 2015, Palacios Medinacelli et al., 2016, C. Insaurralde, 2018, Keller, 2016] is to provide a formal model that considers all the available\necessary heterogeneous sources of information, in a single well-defined representation. Such a model has the advantage of increasing the shareability and accessibility of the knowledge, and allows for a computer readable representation, which enables to draw automatic inferences over the represented knowledge.

In [Emmanouilidis et al., 2010] the importance to develop a domain ontology in the context of Condition Based Maintenance is analyzed. The approach emphasizes the relevance of an incremental modelling capacity along with a domain ontology, to provide a diagnosis of assets and a prognosis on how they will evolve over time. The role of the ontology in this work is to structure the available knowledge and deal with the heterogeneous data sources.

In [Regal and Pereira, 2014] they study how an ontology can be constructed to leverage from the interaction of Intelligent Maintenance Systems (IMS) and Spare Parts Supply Chain (SPSC). IMS allow to forecast failures which can avoid downtime and provide competitive advantages. The interaction of this information with SPSC can enable a more precise demand and store planning for spare parts, thus the relevance of the interaction of both systems. It is pointed out that some of the challenges of a successful interaction between such systems is due to the semantic differences between these areas, with diverse concepts and diverse vocabularies. Thus in [Regal and Pereira, 2014], it is proposed an ontology as a common vocabulary, to overcome these limitations and to serve as a basis for the future construction of an integration system.

In [Ebrahimipour and Yacout, 2015] a methodology for constructing an ontology schema is presented which relies on industry standards (ISO 14224, ISO 15926) as basis for providing a generic, shared and standard vocabulary for maintenance. The aim of this methodology is to support maintenance knowledge representation by providing a shareable and operable format that facilitates knowledge retrieval and semantic extraction during fault diagnosis process. The ontology is represented using OWL\RDF. It allows to overcome the heterogeneity of the vocabulary and of the data sources found in maintenance documents.

There are also studies [Ebrahimipour et al., 2010, Dittmann et al., 2004] on ontology based systems for Failure Mode and Effects Analysis (FMEA). FMEA was initially introduced by NASA on the 60's for its space program. It is a method to analyze potential reliability problems all along the development cycle of a

project. It facilitates taking actions to overcome reliability issues and estimate risks in design time, thus enhancing the reliability through design. Characterizing failures and preventing them, is central to FMEA, where the knowledge about the different systems that involve: design, monitoring, maintenance and evaluation of systems, requires a centralized and common access. The works using ontologies for FMEA have a generic scope, they provide well based general-purpose terms as a foundation for more specific domains. Their focus is to provide access to the heterogeneous information involved in safety and reliability systems. The work of Regal and Pereira [Regal and Pereira, 2014] provides an outline of these approaches.

Following the same line of thought, the work of [Wu et al., 2014] relies on ATA ISPEC2200 and ISO 15926 to obtain a generic system-level representation model for maintenance in aircraft, where the ontology helps to overcome heterogeneity and data inconsistency in the maintenance report. Similarly, in previous work [Palacios Medinacelli et al., 2016], research on proper terminology, challenges and benefits of implementing a maintenance support system are explored. The work states the rationale and potential of an ontology based approach to support avionics maintenance, where knowledge discovery can be used to identify failures, and ontology alignment techniques would allow integration with upper ontologies.

The efforts in the above presented approaches aim to provide a model which considers the proper terminology used in the domain, based on standards and/or recommendations. This model is an ontology that offer a common access point to the resources and systems involved in maintenance to the multiplicity of actors involved in the process. A common challenge on these works is the different formats of the sources of information and the access to the data for each involved actor. The approaches provide analysis and a guidance on how these knowledge bases should be built, operated and highlight their potential. It is envisaged that such knowledge bases, besides providing a common language and common access point to the represented knowledge, would also serve as corner stones to build intelligent systems on top of them. But none of the above mentioned works go beyond the implementation of the ontology. Previous work [Palacios Medinacelli et al., 2017] evaluates the use of ontologies aided by machine learning to build a prototype system to support avionics maintenance, nevertheless the approach provides only guidance on how the search is to be done, and does not reach the implementation phase.

2.3 Ontology Learning and Concept Learning

The literature on machine learning is extensive [Cherkassky and Mulier, 2007] covering a wide spectrum of techniques, applications, perspectives and disciplines.

It is out of the scope of this thesis to provide an extensive survey of the field. In this section we provide an intuition of its general concepts, and define the context under which this thesis takes place in the area.

Machine Learning (ML) is a field of artificial intelligence which relates to automatic learning by computers in a broad sense. The algorithms used in ML can mainly be classified in supervised learning, unsupervised learning and hybrid approaches. Supervised learning refers to the problem of approximating an unknown function f through a hypothesis h, where the learning process is guided by positive (and possibly negative) samples of the correct (resp. incorrect) outputs of the function f. The hypothesis h can then be used to approximate the output of f. In unsupervised learning, there is no function f to approximate. The data is given without any labels (like positive or negative samples) and the task is to identify sets of data, that can be grouped together according to some of their features. Usually a distance between the elements in the data is defined, forming *neighborhoods* of elements and the notions of minimum distance between neighbors and maximum distance between neighborhoods is applied to form clusters. The notion of unsupervised comes from the non-involvement of an external influence to classify the data. A third type of algorithms called hybrid, usually working on semi-labeled data, mix both types of algorithms [Goodfellow et al., 2016].

From these techniques we are interested in those that allow for automatic learning in ontologies, presented next. Other works related to unsupervised approaches for structured data will be presented in Section 2.4.

2.3.1 Ontology Learning

A major challenge for ontology based approaches is the construction of the ontology. Top-level descriptions of a domain are easier to model than detailed descriptions, nevertheless the model should be accurate enough to solve the problem/tasks for which it was designed. When creating a knowledge base, the designer has to select the relevant pieces of information to be represented and organize them accordingly. The features of the represented knowledge, the terminology used and the detail to which the descriptions are given depend on the objective for which the knowledge base was built.

This can be a difficult and time consuming task because of many factors: it requires expertise in the domain, it requires expertise in creating a knowledge base, and it is subject to the perspective of the designers involved, among others.

Ontologies can vary on their size, complexity, usage, maintainability, etc. All these characteristics influence on the feasibility of implementing ontology based systems. A cost analysis method to evaluate the resources needed for ontology

engineering has been proposed in [Simperl et al., 2012], which reflects the variety and complexity of tasks and resources needed to obtain an effective ontology. Thus, automatic ontology learning has a very practical application and benefits, if carried out properly.

Ontology learning is a multidisciplinary field which aims at the automatic generation of ontologies. The term "ontology learning" was coined by Mädche and Staab in 2001, and the first ontology learning workshop held in 2000 brought together people from very different research communities, with works based on ripple down rules, word sense clustering, and information extraction. In [Lehmann and Voelker, 2014] they present a compilation of approaches in the area of ontology learning including contributions by the concept learning community as well as "classical" works on ontology learning from text or other semi-structured resources. It provides an overview of a broad range of ontology learning approaches, from which we borrow the following classification:

- Ontology Learning from text mostly focuses on the automatic or semi-automatic generation of lightweight taxonomies by means of text mining and information extraction. Many of the methods used in ontology learning from text (e.g. lexico-syntactic patterns for hyponymy detection or named-entity classification) are inspired by previous work in the field of computational linguistics, essentially designed in order to facilitate the acquisition of lexical information from corpora. Some ontology learning approaches do not derive schematic structures, but focus on the data level. Such ontology population methods derive facts from text. An example is the Never-Ending Language Learning (NELL) project [Carlson et al., 2010], which reads the web to add statements to its knowledge base and improves its performance over time, via user feedback.

- Linked Data Mining refers to the process of detecting meaningful patterns in RDF graphs. One of the motivations behind this research area is that Linked Data publishers sometimes do not create an explicit schema for their dataset upfront, but focus on publishing data first. Being able to detect the structure within published RDF graphs can, on the one hand, simplify the later creation of schemata and, on the other hand, allow to detect interesting associations between elements in the RDF graph. This can be achieved via statistical schema induction or statistical relational learning methods [Bühmann and Lehmann, 2012, Bühmann and Lehmann, 2013, Völker and Niepert, 2011], which mine frequent patterns and correlations in large data sets. In Linked Data mining, clustering approaches can be used to group related resources and provide an enhanced structure for the underlying data.

- Concept Learning in Description Logics and OWL is a direction of research that aims at learning schema axioms, such as definitions of classes, from existing ontologies and instance data. Most methods in this area are based on Inductive Logic Programming methods [Nienhuys-Cheng and De Wolf, 1997]. While many algorithms, such as DL-FOIL [Fanizzi et al., 2008b] and OCEL [Lehmann and Hitzler, 2010] are generic supervised machine learning approaches for description logics, there are also specific adaptations to ontology learning, in terms of performance and usability. Closely related to concept learning in Description Logics is onto-relational learning, which combines methods for learning OWL axioms with rule learning approaches [Lisi and Esposito, 2009].

- Crowd-sourcing ontologies is an interesting alternative to purely automatic approaches as it combines the speed of computers with the accuracy of humans. Provided that the task to be completed is simple enough, it only requires the right incentives for people to contribute. Examples of crowd-sourcing in the field of ontology learning include taxonomy construction via Amazon mechanical turk, and games with a purpose for ontology population [Karampinas and Triantafillou, 2012, Chilton et al., 2013].

To sum up the approaches based on learning from text are closely related to text analysis and natural language processing. Although in maintenance reports there exist valuable information in the form of text, the automatic interpretation of the textual content of these reports is out of the scope of our approach. We aim to discover concepts based on the values of their instances and through the vocabulary and constructors allowed in an ontology. This is independent from interpreting what is written in the reports, and different from discovering relations between textual elements.

The approaches in the linked data category rely on techniques for graph mining which require the full graph. This implies that the all the knowledge that can be inferred, must be explicitly represented in this graph (in the context of OWL this is called the materialization of the ontology). Graph mining techniques are further detailed in Section 2.4.2. Additionally, they rely on frequent patterns in large amounts of data. In our problem such large datasets are not available.

In contrast, the approaches in the concept learning category are interesting for our approach because their goal is to discover concept definitions. Either these approaches are designed for supervised machine learning or are based on clustering techniques. These two kinds of approaches are described in more detail in Section 2.3.2 and in Section 2.4 respectively.

Crowdsourcing is an alternative to fully automatic learning, where the humans are involved in the learning cycle, reviewing the content in an ontology, or

explicitly adding content to it. These alternatives can not be applied to our problem because it is not possible to ask users to classify information in categories that are not know in advance. The idea of involving the user in the learning cycle is interesting. In our problem such a process could be used to validate the discovered categories.

2.3.2 Concept Learning

Inductive machine learning studies the problem of learning a function f for which samples of its inputs and outputs are given as pairs of the form $\{x, f(x)\}$. The task is to learn a function h, called hypothesis, that best approximates f. Each sample x is described by its features, which are represented as a vector. Each value of each feature of a sample can be either discrete or a real number [Goodfellow et al., 2016]. If f is a binary function, the problem is called binary classification.

Binary classification in description logics is called concept learning [Tran et al., 2012] since the function to be learned is expected to be characterized by a concept expression, for which the output is binary. Concept learning differs from the classical setting mentioned above in that an object is described not only by the values of its features, but also by its relations to other objects.

In [Tran et al., 2015] the related work on methods of concept learning in DLs are classified into three groups. The first group focuses on learnability in DL and presents some relatively simple algorithms [Cohen and Hirsh, 1994]. The second group studies concept learning in DLs using refinement operators as in inductive logic programming [Badea and Nienhuys-Cheng, 2000, Lehmann and Hitzler, 2010, Ratcliffe and Taylor, 2017]. The third group exploits bisimulation for concept learning in DLs [Ha et al., 2012, Nguyen and Szałas, 2013, Tran et al., 2015]. Our work is inspired by the latter two types of approaches.

Within the second group of approaches (based on refinement operators) DL-Learner [Lehmann, 2009] is a state-of-the art tool for concept learning. It provides a framework for supervised machine learning using several algorithms which are highly parameterizable. It uses refinement operators like CELOE [Lehmann et al., 2011] for OWL expressions and ELTL [Lehmann and Haase, 2009] for the *EL* family of DL. Depending on the desired properties of the operator and the DL-constructors allowed, the operator traverses the space of possible concept expressions in a top-down or bottom-up manner. Then these concept expressions are evaluated, using heuristics, to find the most suitable ones. Shorter and more simple expressions are preferred by these algorithms. DL-learner, likewise the other approaches in this group, first generates concept expressions from the available classes and constructors of the underlying ontology, and then uses the

positive (and possibly negative) samples to decide which expressions are more promising to iterate the process over them.

The approaches of the third group use the notion of bisimulation from modal logics that characterizes distinguishability between objects. A proper bisimulation definition is given to suit DL and it is used to define partitions in the set of objects of the ontology [Ha et al., 2012, Nguyen and Szałas, 2013, Tran et al., 2015]. These partitions define equivalence classes between the objects, i.e. those objects that can not be distinguished between them. Each one of these partitions is a DL-concept, and finding the partitions is equivalent to learn concepts. The way these partitions are found, is similar to those approaches based on refinement operators in that combinations of the constructors allowed by the language are generated, and then evaluated against the samples.

To sum up, the approaches based on refinement operators require positive (and possibly negative) samples. In our problem, we require to distinguish between the samples based on their features, and not by a label, i.e. we do not count with positive/negative samples, this problem has been partially studied in [Palacios Medinacelli et al., 2018, Palacios Medinacelli et al., 2017]. With regard to the approaches based on bisimulation, the notion of equivalence classes is interesting because it allows to formally specify: if the model can distinguish between samples, the properties that separates them, and provide approximations to the concepts that are looked for. Finally, in both above mentioned types of approaches, the search of concept expressions is done by combining the constructors allowed in the language, without regard of the information in the samples. This search space grows exponentially. The samples are only used after the concepts are generated, to avoid an exhaustive search in this space. The samples are not used to generate the concepts themselves.

2.4 Unsupervised Learning over Structured Data

2.4.1 Clustering in Knowledge Bases

Clustering in computer science refers to an area of research concerned with data analysis and interpretation. It is a field with extensive literature and applications, like data mining, pattern recognition, image analysis, etc. In this thesis we do not intend to provide a deep review of the field, but to highlight the applicability to our problem. A general reference to clustering can be found in [Pedrycz, 2005]. Intuitively, clustering is a methodology to identify interesting groups of data within a given dataset, based solely on the features of the data. The dataset is given without further information on how it can be organized. Clustering provides means to distinguish sets of data that share some meaningful features.

Each such set is called a cluster. In this sense, the principle in clustering is similar to our problem, where distinctions between groups of samples in a dataset are searched for, and where the construction of such groups does not depend on the labeling of the analyzed samples (as in opposition to supervised learning).

Two central notions in techniques for clustering are: similarity and distance. A similarity measure allows to compare the elements in a dataset, and it can be defined for each type of data the clustering problem handles. A distance on the other hand, is a metric that allows to establish how close the elements in a dataset are to each other. The clusters are then formed by minimizing the distance between the elements inside each cluster, and maximizing the distance between the clusters. The similarity measure, the distance and the very notion of cluster can vary depending on the perspective, the underlying data and the goal of the clustering process. Some of the main techniques include: hierarchical clustering, k-means clustering, distribution-based clustering, density-based clustering. These techniques are developed for data whose features are represented as vectors of discrete or real values. When considering description logic knowledge bases, these approaches can not be directly applied, since the features are not only discrete or continuous values, but also other objects with their own features. Thus adaptations or new approaches are required to handle these cases. In the following we present some of the work in this area where the concept of clustering is applied to knowledge bases.

Some approaches have been proposed to adapt well-known clustering techniques to ontologies and knowledge bases [Fanizzi et al., 2008a, Fanizzi and d'Amato, 2007, Lee and Gray, 1998, Nowak-Brzezińska, 2016]. These approaches define a similarity measure based on a lexicographic distance between the rules, text-based similarity in the instances of the knowledge base, and similarity based on a set of defined features (like concepts). Then the distance and the clusters formed are obtained using a suitable and well studied clustering algorithm (such as k-means or Hopfiled network). The work of Fanizzi and d'Amato [Fanizzi and d'Amato, 2007] presents an adaptation of hierarchical clustering to semantic knowledge bases, like OWL ontologies. Namely, it can be used to discover interesting groups of semantically annotated resources in a wide range of concept languages. The method exploits a dissimilarity measure that is based on the resource semantics with respect to a number of dimensions corresponding to a committee of features. These features are represented by a group of concept descriptions. The algorithm is an adaptation of the classic Bisecting k-Means to complex representations typical of the ontology in the Semantic Web.

An algorithm to cluster rules in a knowledge base is presented in [Lee and Gray, 1998] based on the syntactic information in the rules and their lexical similarity. Clustering is achieved by using a neural network algorithm based on

Hopfield network. The goal of this approach is to support Knowledge Based Systems maintenance, by rendering the content of the knowledge base more accessible through the clustering of similar rules. The clusters are aimed to provide an adequate amount of information and acceptable sized clusters.

The above mentioned approaches transform the information present in the ontology into vectors of discrete/real values. Then, a similarity measure and a distance are defined over them, to proceed with clustering. Full vectors of the same size are needed to compare them. This in our case is undesirable. Indeed, in our problem, the information can be incomplete (some of the values on such vectors might be unknown) and converting the information in the ontology into vectors, may lead to ignore relations between instances.

A problem similar to ours has been investigated [Fanizzi et al., 2004, Kietz and Morik, 1994] for the automated construction of terminologies from assertions in Description Logics (DLs) ($\mathcal{ALC}$). These approaches are based on first finding the most specific generalization (msg) of each sample in a set of samples, and then finding the most general descriptions (mgd) for the msc's found. To select which mgd's represent a cluster, the notion of maximal set of mutually disjoint concepts is used.

To obtain clusters, in [Fanizzi et al., 2004] an initial taxonomy is constructed based on the content of a given A-Box, thus building a hierarchy of *super concepts*. Each super concept in this taxonomy is analyzed to obtain its Mutually Disjoint Concepts (MDC). Each MDC serves as the basis for an iteration of the same process to find sub-clusters. Here, a supervised phase takes place where the disjoint concepts (each MDC) are generalized/specialized using upward/downward operators to refine them. Since clusters in this setting need to be disjoint, the instances of other concepts are used as negative samples. When a concept covers negative samples, it needs to be specialized with a downward refinement operator. This method relies on the assumption that the MDC is given. However, as the authors point out, finding the MDCs has a superpolynomial worst case complexity.

Finally, some tools [Lehmann et al., 2017] have been developed for data analytics considering OWL knowledge bases. SANSA[1] is a big data processing engine for scalable processing of large-scale RDF data. It is a state of the art knowledge analytics system, which uses machine learning algorithms to exploit the graph structure and semantics of the background knowledge specified using the RDF and OWL standards. The machine learning layer of SANSA currently supports algorithms for unsupervised learning, from which for clustering we find: RDF Modularity Clustering, BorderFlow Clustering, Power Iteration Clustering, Link based Silvia Clustering. It also provides Frequent Pattern Mining through

[1] http://sansa-stack.net

Association Rule Learning. These algorithms are based on embedding of entities and relations into low-dimensional vectors [Yang et al., 2014].

Such tools are intended to either find relations between concepts, mostly based on textual analysis, and find concepts based on frequent patterns. In our problem, we do not search to propose new rules between similar concepts, since these concepts are not known in advance. And we can not solely rely on frequent patterns, since we are also interested in finding rare occurrences.

2.4.2 Graph Mining

Finally, the instances in an ontology can be represented as graphs, we need to consider graph mining techniques that could be applied to achieve our goal.

One of the main problems addressed by the different works with graph structures lies in the subgraph isomorphism issue, through which common graph structures can be found. The work in [Yan and Han, 2003] proposes an algorithm, CloseGraph to mine closed frequent graphs. [Motoda, 2007], [Inokuchi et al., 2000], [Inokuchi et al., 2003] and [Yoshida et al., 1994] present two approaches for extracting frequent subgraphs: AGM and GBI. AGM relies on the representation of the graphs by adjacency matrices. Each graph is assigned a unique label that encodes the adjacency matrix. The subgraph isomorphism problem is solved by comparing the codes of the graphs and an Apriori-like algorithm that generates graphs of size K joining two subgraphs of size k-1 and verifies their frequency. The use of a taxonomy enables the extraction of generalized subgraphs. GBI relies on the chunking of adjacent nodes in order to generate subgraphs and the rewriting of the graphs given the selected subgraphs as new nodes. Typicality and chunking criteria are used in order to select the pairs of adjacent nodes. The Subdue graph-based mining system [Ketkar et al., 2005] also relies on the use of heuristics to evaluate the potential subgraph patterns. [Karunaratne et al., 2010] presents an approach for mining subgraphs, using an itemset mining approach. The MFI (Maximal Frequent Itemset) [Aggarwal and Yu, 1998] algorithm is applied on graph structures that are formerly transformed into itemsets.

While the approach appears to be very efficient for a purpose of graph classification, contrary to the approaches described above, the lack of comprehensibility of the subgraphs that are discovered is a limitation for the purpose of concept discovery.

Using graph mining, the most relevant structures of a set of graphs can be established, and patterns that discriminate positive from negative samples can be found. These approaches search for similar structures. This is close to our goal, since similar instances are expected to belong to the same concepts. Nevertheless, these structures are required to be presented as graphs and are

based on detecting frequent patterns, which implies that non-frequent structures might be ignored. Since we are working with sparse data we need to consider unfrequent patterns as well. Additionally, the data represented as graphs are not guaranteed to have a representation in an ontology.

2.5 Summary

In this chapter we have provided an extensive range of studies related to our problem. This illustrates the breadth of the research that influences this thesis. We have presented related work from the perspective of description logics, system diagnosis and maintenance, concept learning, and clustering. All these areas intersect part of our work, and provide the foundations for our approach.

As explained in the introduction (Chapter 1), we aim to support avionics maintenance by discovering failure signatures. The terminology, statement of the problem, and approaches that aim to support diagnosis were presented in the Section 2.2: model based diagnosis section. We have seen that extensive work on this area has been done, which shows and emphasizes the need to minimize the number of repair suggestions (diagnosis) and the importance of providing the reasons for the diagnosis (explainability). Some logic-based research [Horridge, 2011] propose an abductive approach to the problem, where the detected failure is seen as an observation, and the logic statements that make this observation true, are seen as possible explanations for the detected failure. Nevertheless, central to all these approaches is the availability of the model of the system/equipment to be diagnosed. In our case, not only this model is not available, but we require to model the diagnosis process, which goes beyond considering only the model of the equipment, where the experience of the technicians plays a central role in the possible corrective actions for a failure.

We have also seen that in industrial maintenance, several actors, facilities, and locations are involved. This implies a high heterogeneity in the systems and in the data sources that take part in maintenance. To allow a centralized access to the heterogeneous and distributed information, several approaches that propose the use of ontologies in maintenance have been presented. As we have seen, they highlight the relevance of a standardized model to enable and increase accessibility and shareability to the available information, but they do not propose specific implementations of support systems based in those models.

Finally, from a more technical point of view, the use ontologies to discover information (i.e. approximate failure signatures) requires automatic learning in ontologies. We have outlined the main areas of ontology learning, and focused on concept learning. We have seen that there exist supervised machine learning tools and algorithms, and approaches for clustering in ontologies. Those approaches

that are designed for supervised learning are not suited to our problem, since the negative samples can not be determined *a priori*, and the positive samples are not complete. Thus, clustering in ontologies provides alternative means to discover and extract the concept definitions that may represent signature failures. As we have seen, clustering techniques, as well as those techniques based on graph structures, rely on frequent patterns or in information presented as graphs. In our case these represent a limitations, since we are interested on frequent and unfrequent occurrences of failure signatures, and in meaningful descriptions in terms of the ontology.

To overcome the above stated limitations: non availability of a model of the process, the lack of positive and negative samples, the ability to provide results over sparse data and provide explainable suggestions, in chapter 3 we present our approach.

Chapter 3

The Approach : Situation Discovery

We are interested in the ability to determine when a set of individuals can be distinguished from the rest. Each such set, for which we can find a proper definition is called a *situation* in this chapter. To obtain these situations we present our approach for unsupervised concept learning, which is based on the notion of concept refinement. The intuition behind the refinement process is, that given a set of individuals $\mathcal{X}$, we want to construct descriptions for sub-sets of $\mathcal{X}$ that can be represented by a DL concept. Next, once the descriptions are constructed, we provide means to impose a preference on these DL concepts so that only the ones that are interesting for us are retrieved. These concepts are later integrated to the original ontology, where they can be used for classification. The construction of the concept expressions rely on the properties of the individuals in $\mathcal{X}$ analyzed, and therefore the concept definitions provide an explanation on which are the properties of the individuals that are relevant to distinguish them from the rest.

To this end, we start from a concept C, for which all individuals in $\mathcal{X}$ are instances, and construct refinements C' that are each time more specific than C, and thus capture each time a smaller set of individuals $\mathcal{X}' \subseteq \mathcal{X}$. If we iterate this process we obtain a concept for every subset of $\mathcal{X}$, that can be described by a DL concept. Each one of these DL concepts is referred as a *situation* in the ontology.

A challenging aspect of our problem is that we do not know *a priori* the individuals in the target sets we are looking for, and thus we propose a process to construct all relevant concept expressions (i.e. related to the individuals) from which we can select those that best suit a preference criteria.

In the following we present the preliminary theoretical background and next we introduce the definitions and notions necessary to specify in detail the approach. Then, the algorithms to refine concepts and to iterate over the resulting refinements are presented. Finally, we show how this process enables us to obtain the relevant subsets of the given input set of instances $\mathcal{X}$.

3.1 Preliminaries

In this section we formally introduce DL [Baader et al., 2010, Baader et al., 2005, Baader et al., 2017] ontologies and in particular the DL language $\mathcal{ELO}$ which belongs to the OWL 2 EL profile[1] and has a practical polynomial reasoning algorithm [Kazakov et al., 2012]. We also define some non-classical DL reasoning tasks that are pertinent to our approach.

3.1.1 $\mathcal{ELO}$ Ontologies

A DL-Knowledge Base or DL-Ontology (ontology for short) represents the knowledge of an application domain in terms of concepts and relations, where each element in the domain is called an individual. An ontology comprises two components: a T-Box and an A-Box. The T-Box contains the terminology of the knowledge base, that is the vocabulary for representing the knowledge, and the A-Box contains assertions about named individuals in terms of the vocabulary. These notions are formalized next.

Syntax of $\mathcal{ELO}$

Definition 3.1 (Concept names, role names and individuals)**.** *The vocabulary of the ontology comprises three disjoint sets N_c, N_r, and Δ. N_c is a set of unary relations called* concept names *or* atomic concepts*, N_r is a set of binary relations called* role names*, and Δ is a set of arguments that these relations take, called* individuals*.*

The concepts in the ontology denote sets of individuals and the roles denote binary relationships between individuals.

Example 3.1. *To describe the domain of* human*, we may have the following concepts $N_C = \{Person, Male, Female, Man, Woman, Mother, Father\}$, the roles $N_r = \{hasChild, hasGrandChild\}$, and two individuals $\Delta = \{Jean, Marie\}$.*

In addition to atomic concept and role names, complex concept descriptions can be defined. The syntax of a specific DL language specifies the constructors

[1] https://www.w3.org/TR/owl2-overview/#Profiles

allowed to form words over the concept and role names. These words are well-formed sentences or *formulae* with well-defined semantics, according to a set of rules specific to each language. In Figure 3.1 the set of constructors of $\mathcal{ELO}$, their syntax and their semantics are presented. Complex concepts are defined recursively using the constructors in Figure 3.1.

In the following we denote (possibly complex) *concept names* by uppercase letters like C and D, *role names* by lowercase letters like r and s, and *individual names* by lowercase letters like x, y and z.

In the remainder of this thesis all concepts are assumed to be $\mathcal{ELO}$ concepts.

	Syntax	Semantics
Concepts:		
atomic concept	A	$A^{\mathcal{I}}$
nominal	$\{a\}$	$\{a^{\mathcal{I}}\}$
top	$\top$	$\Delta^{\mathcal{I}}$
conjunction	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
existential restriction	$\exists R.C$	$\{x \mid \exists y \in C^{\mathcal{I}} \colon \langle x, y\rangle \in R^{\mathcal{I}}\}$
Axioms:		
concept inclusion	$C \sqsubseteq D$	$C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$

Figure 3.1: Syntax and semantics of $\mathcal{ELO}$.

Definition 3.2 ($\mathcal{ELO}$ Ontology, T-Box and A-Box)**.** *An ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A}\rangle$ is a pair, where $\mathcal{T}$ is called the T-Box and $\mathcal{A}$ is called the A-Box. The T-Box $\mathcal{T}$ is a finite set of concept inclusion axioms of the form $C \sqsubseteq D$. The A-Box is a finite set of concept and role assertions of the form $C(x)$ and $r(x, y)$, respectively.*

A concept equivalence $C \equiv D$ is an abbreviation for the two concept inclusions $C \sqsubseteq D$ and $C \sqsubseteq D$, and it is called a concept definition. Concept equivalences can be used to assign names to complex descriptions in a T-Box. The description language has a model-theoretic semantics detailed next. Thus, statements in the T-Box and in the A-Box can be identified with formulae in first-order logic.

Semantics of $\mathcal{ELO}$

Definition 3.3 (Interpretation)**.** *An interpretation $\mathcal{I}$ consists of a non-empty set Λ called the* domain *of $\mathcal{I}$ and an interpretation function $\cdot^{\mathcal{I}}$ that assigns to each $C \in N_C$ a set $C^{\mathcal{I}} \subseteq \Lambda^{\mathcal{I}}$, to each $r \in N_r$ a binary relation $r^{\mathcal{I}} \subseteq \Lambda^{\mathcal{I}} \times \Lambda^{\mathcal{I}}$, and to each $x \in \Lambda$ an element $x^{\mathcal{I}} \in \Lambda^{\mathcal{I}}$. The interpretation function is extended to complex concepts as shown in Figure 3.1.*

Definition 3.4 (Model)**.** *An interpretation $\mathcal{I}$ satisfies (also called a model of) an axiom $C \sqsubseteq D$ (resp. $A(x), r(x, y)$), written $\mathcal{I} \models C \sqsubseteq D$ (resp. $\mathcal{I} \models A(x), \mathcal{I} \models r(x, y)$), if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ (resp. $x^{\mathcal{I}} \in C^{\mathcal{I}}, (x^{\mathcal{I}}, y^{\mathcal{I}}) \in r^{\mathcal{I}}$). If an interpretation $\mathcal{I}$ satisfies all axioms in an ontology $\mathcal{O}$, $\mathcal{I}$ is called a model of $\mathcal{O}$ (written $\mathcal{I} \models \mathcal{O}$).*

Definition 3.5 (Logical consequence)**.** *An axiom β is a logical consequence of an ontology $\mathcal{O}$, written $\mathcal{O} \models \beta$, if every model of $\mathcal{O}$ satisfies β.*

If x is an instance of C w.r.t. an ontology $\mathcal{O}$, i.e. $\mathcal{O} \models C(x)$, and $\mathcal{O}$ is clear from the context, we say that $C(x)$ holds (in $\mathcal{O}$) for short.

Definition 3.6 (Subsumption)**.** *A concept C is subsumed by concept D w.r.t. $\mathcal{O}$ if $\mathcal{O} \models C \sqsubseteq D$.*

If $\mathcal{O} \models C \sqsubseteq D$ (i.e. C is subsumed by D) we say that C is *more specific* than D. Inversely, we say that D is *more general* than C.

We next introduce ontology materialization is a reasoning task that computes logical consequences of an A-Box w.r.t. a T-Box and it is the most important task in some languages, e.g. OWL 2 RL. [Glimm et al., 2017].

Definition 3.7. *We say that $\mathcal{O}$ is materialized if (1) $\mathcal{O} \models A(a)$ implies $A(a) \in \mathcal{O}$ for each concept name A from $\mathcal{O}$ and each individual a from $\mathcal{O}$; and (2) $\mathcal{O} \models r(a, b)$ implies $r(a, b) \in \mathcal{O}$ for each role name r from $\mathcal{O}$ where a, b are individuals in $\mathcal{O}$.*

Materialization is a stepping stone for rewriting based query answering for languages that allow existential quantification [Kontchakov et al., 2011].

Definition 3.8. *An A-Box A is called acyclic iff there are no $n \geq 1$ and individuals $a_0, a_1, \cdots, a_n$ and roles $r_1, \cdots, r_n$ such that*

- $a = a_0$,
- $r_i(a_{i-1}, a_i) \in A$ *for* $1 \leq i \leq n$,
- *there is* $j, 0 \leq j < n$ *such that* $a_j = a_n$.

Definition 3.9 (Most Specific Concept (msc))**.** *Given an ontology $\mathcal{O}$ and an individual x, we say that C is the* most specific concept *for x if:*

$$\mathcal{O} \models C(x)$$

and for every other concept D, with $\mathcal{O} \models D(x)$ we find

$$\mathcal{O} \models C \sqsubseteq D$$

One must note that the existence of the msc is not guaranteed in DL. In [Baader and Molitor, 1999] it is highlighted that the msc in $\mathcal{EL}$ needs not to exist, if cyclic A-Boxes are allowed. For example for the simple A-Box $\{r(a, a), A(a)\}$, the msc does not exist, since the concepts of the form $\exists r.\exists r.\exists r. \ldots . A$ for which a is an instance, are infinite. Even if we restrict to acyclic A-Boxes (as in our case)

the existence of the msc is not guaranteed [Peñaloza and Turhan, 2010]. For instance, for $A = \{A(a)\}$, $T = \{A \sqsubseteq \exists r.A\}$, the msc would require an infinite chaining of existential restrictions, which is not a concept.

To guarantee the existence of the msc, a sufficient condition is that T-Box is terminology and A-Box is acyclic [Baader, 2003b]. A T-Box is a terminology if it is comprised of only primitive concepts definitions, of the form $A \equiv D$ and T-Box does not contain multiple definitions, i.e. there can not be two distinct concept definitions D_1 and D_2 such that both $A \equiv D_1$ and $A \equiv D_2$ belong to the T-Box. Note that these restrictions still allow for concept definitions that refer to themselves.

In this thesis we restrict our work to acyclic A-Boxes and terminology T-Boxes. In fact, the signatures discovered are always concept definitions of the form $C \equiv D$, thus the Most Specific Concept (msc) is guaranteed to exist.

Moreover without loss of generality, we assume that A-Boxes only contain assertions about concept names and role names. This can be achieved by introducing a definition for a concept description appearing in an A-Box, that is, if we have $C(a) \in A$ where C is a complex concept, then we add into the T-Box a concept definition $A_c = C$ and $A_c(a)$ into A.

Definition 3.10 (Sub-concept)**.** *Given a (possibly complex) concept C, the set of its* sub-concepts $\mathsf{Sub}(C)$ *is defined recursively by:*

$$\begin{aligned} \mathsf{Sub}(B) &= \{B\} \\ \mathsf{Sub}(\exists r.C) &= \{\exists r.C\} \cup \mathsf{Sub}(C) \\ \mathsf{Sub}(C_1 \sqcap C_2) &= \{C_1 \sqcap C_2\} \cup \mathsf{Sub}(C_1) \cup \mathsf{Sub}(C_2) \end{aligned}$$

Definition 3.11 (Concept Size)**.** *Given a (possibly complex) concept C, its* size $\mathsf{Size}(C)$ *(in symbols $|C|$) is defined by:*

- *If $C = A \in N_c \cup \{\top, \bot\}$, then $\mathsf{Size}(C) = 1$.*
- *If $C = C_1 \sqcap C_2$, then $\mathsf{Size}(C) = 1 + \mathsf{Size}(C_2) + \mathsf{Size}(C_1)$.*
- *If $C = \exists r.D$, then $\mathsf{Size}(C) = 1 + \mathsf{Size}(D)$.*

Definition 3.12 (Concept substitution)**.** *Given a (possibly complex) concept C, a* concept substitution *of D by E in C, written $C|_{D \to E}$, is the replacement of an occurrence of the sub-concept $D \in \mathsf{Sub}(C)$ in C by E.*

To illustrate these notions, consider the following example:

Example 3.2. *Consider the set of individuals:*

$$\Delta = \{Jean, Marie\}$$

the set of concept and the set of role names:

$$N_C = \{Person, Male, Female, Man, Woman\} \text{ and } N_r = \{hasChild\}$$

and the T-Box:

$$\begin{aligned} \mathcal{T}_1 = \{ \quad Man &\equiv Person \sqcap Male \\ Woman &\equiv Person \sqcap Female \\ Father &\equiv Man \sqcap \exists hasChild.Person\} \end{aligned}$$

If we unfold the concept $Father$, we have:

$$Unfold(Father) = (Person \sqcap Male) \sqcap \exists hasChild.Person$$

Then, the set of sub-concepts of $Father$ is:

$$\begin{aligned} \mathsf{Sub}(Unfold(Father)) = \{ \quad & (Person \sqcap Male) \sqcap \exists hasChild.Person, Person \sqcap Male, \\ & Person, Male, \exists hasChild.Person\} \\ |\mathsf{Sub}(Unfold(Father))| = \quad & 5 \end{aligned}$$

And the size of $Father$, is:

$$\mathsf{Size}(Unfold(Father)) = \quad 1(+(1+1+1)+(1+1)) = 6$$

Note that the number of sub-concepts of C and the size of C, are different, and that $\mathsf{Sub}(Unfold(Father))$ is bounded by $\mathsf{Size}(Unfold(Father))$.

Next, using concept substitution, we can obtain a concept equivalent to $Woman$ from the concept Man:

$$\begin{aligned} Man|_{Male \to Female} &= Person \sqcap Female \\ \text{We find this is equivalent to:} & \\ Woman &\equiv Man|_{Male \to Female} \end{aligned}$$

We can also see from the T-Box that $Father$ is subsumed by Man, in symbols:

$$Father \sqsubseteq Man$$

(since every Father necessarily is a Man, but not every Man is necessarily a Father)
These notions allow us to make some inferences, for example if we are given the following set of facts in the A-Box:

$$\begin{aligned} \mathcal{A}_1 = \{ \quad & Person(Jean), Male(Jean), hasChild(Jean, Marie) \\ & Person(Marie), Female(Marie)\} \end{aligned}$$

We can conclude that:

$$\mathcal{O} \models \{Man(Jean), Woman(Marie), Father(Jean)\}$$

3.1.2 Justifications in DL Ontologies

In this section we review the notion of ontology justification [Kalyanpur et al., 2007, Baader et al., 2007], which refers to the most pertinent information, from a (possibly big) ontology, for deducing a particular entailment. We consider similar basis, for our ideas on how to extract the most useful properties for concept refinement.

Definition 3.13 (Justification)**.** *Let $\mathcal{O}$ be a knowledge base and let β be an axiom. A subset $\mathcal{M} \subseteq \mathcal{O}$ is called a* justification *for β iff*

$$\mathcal{M} \models \beta$$

and

$$\mathcal{M}' \not\models \beta$$

for every $\mathcal{M}' \subset \mathcal{M}$.

Intuitively a justification for a concept inclusion β, is a minimal subset of axioms $\mathcal{M}$ from the ontology, such that they are sufficient and necessary for β to be a logical consequence of $\mathcal{M}$. The number of possible justifications for such a β in $\mathcal{ELO}$ may be exponential in the number of axioms in $\mathcal{O}$. Thus it can not be ensured that all justifications can be computed in polynomial time with respect to the size of the ontology. Nevertheless, it is possible to compute one justification in polynomial time for $\mathcal{ELO}$ language [Baader et al., 2007][2].

Notice that the notion of justifications is given with respect to a whole ontology. There has been some work [Arif et al., 2016, Kazakov and Skocovský, 2018] on efficient computation of justifications with respect to $\mathcal{EL}$ T-Box for subsumption axioms of the form $C \sqsubseteq D$. In our approach we focus on an A-Box. Thus this notion is later modified to compute minimal subsets of an $\mathcal{ELO}$ A-Box. These minimal A-Boxes enable us to construct concept refinements.

In the next section we present the notions and provide the definitions that are specific to our approach.

3.2 Approach Definitions

Given an ontology $\mathcal{O}$ we aim to find interesting subsets of its individuals, and for each one of these sets provide a description using the vocabulary and the constructors allowed by the language. We call each one of this sets a *situation* in O (defined next). In this section we formally define this problem in terms of finding the situations in an ontology. Then we provide a *refinement operator*

[2]The cited paper considers the family $\mathcal{EL}$, but it is shown that $\mathcal{ELO}$ has the same complexity since the logical consequence in $\mathcal{ELO}$ is polynomial.

which enables us to discover and extract these situations, in the form of complex DL concepts definitions. Before the algorithms are presented, some notions necessary to specify the results and their properties are introduced.

Let us continue by formally defining the interesting subsets of individuals in an ontology.

Definition 3.14 (A representative concept)**.** *Let Δ be a set of all individuals in an ontology $\mathcal{O}$, and let $\mathcal{X} \subseteq \Delta$. For a concept C, we say that $\mathcal{X}$ is* represented *by C (or C represents $\mathcal{X}$) w.r.t. $\mathcal{O}$ and Δ, if:*

$$C(x) \text{ holds for all } x \in \mathcal{X}, \text{ i.e. } \mathcal{O} \models C(x), \text{ and}$$

$$C(y) \text{ does not hold for any } y \in \Delta \setminus \mathcal{X}, \text{ i.e., } \mathcal{O} \not\models C(y).$$

If there exists a concept C that represents $\mathcal{X}$, we say that $\mathcal{X}$ is representable.

Example 3.3 (Representative Concept)**.** *Consider the set of individuals $\Delta = \{f_1, f_2, f_3\}$, a subset of individuals $\mathcal{X} = \{f_1, f_2\}$, and the following ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$:*

$$\begin{aligned} \mathcal{T} &= \{C \equiv \exists r.\top\} \\ \mathcal{A} &= \{A(f_1), B(f_2), E(f_3), r(f_1, f_3), r(f_2, f_3)\} \end{aligned}$$

By checking the two following conditions, we determine if C represents $\mathcal{X}$. We find:

$$\mathcal{O} \models \{C(f_1), C(f_2)\}$$

$$\mathcal{O} \not\models \{C(f_3)\}$$

Thus $\mathcal{X}$ is represented by C.

However, it is not true that each set of instances can always be represented.

Example 3.4 (Example 3.3 contd.)**.** *Consider the set $\mathcal{X}' = \{f_2, f_3\}$, there is no $\mathcal{ELO}$ concept that can represent $\mathcal{X}'$.*

A set of individuals that can be represented have to share some common properties merely among them, which are made explicit by the concepts that represent them. By reading the concept definition, we get an explicit explanation of their common properties. For example, the individuals f_1 and f_2 share the property that they are connected to some individual via the role r. Whilst, f_2 and f_3 do not have any property in common that can distinct them from f_1.

There are two sets of individuals which can be always represented as shown by the following lemma.

Lemma 3.1. *Given an ontology $\mathcal{O}$ and a set Δ of individuals, we have*

- $\top$ *is a representative concept for* Δ.
- $\bot$ *is a representative concept for* $\emptyset$.

Proof. It is clear that for all $x \in \Delta$, $\mathcal{O} \models \top(x)$ holds and for any $y \in \Delta \setminus \Delta$, it satisfies $\mathcal{O} \not\models \top(x)$. So $\top$ represents Δ w.r.t. any ontology $\mathcal{O}$ and Δ. It can be proved that $\bot$ represents $\emptyset$ w.r.t. any ontology $\mathcal{O}$ and Δ in a similar way. □

Proposition 3.1. *Given an ontology* $\mathcal{O}$*, the set* Δ *of individuals in* $\mathcal{O}$*, and* $\mathcal{X} \subseteq \Delta, \mathcal{X}' \subseteq \Delta$*. If* $\mathcal{X}$ *and* $\mathcal{X}'$ *are representable, then* $\mathcal{X} \cap \mathcal{X}'$ *is representable in* $\mathcal{ELO}$*.*

Proof. Since $\mathcal{X}$ and $\mathcal{X}'$ are representable, we assume that C and C' represent $\mathcal{X}$ and $\mathcal{X}'$ respectively. Then we can see that $C \sqcap C'$ represents $\mathcal{X} \cap \mathcal{X}'$. This is because for any $x \in \mathcal{X} \cap \mathcal{X}'$, $\mathcal{O} \models C(x)$ and $\mathcal{O} \models C'(x)$, which implies $\mathcal{O} \models C \sqcap C'(x)$. For any $y \in \Delta \setminus (\mathcal{X} \cap \mathcal{X}')$, if $y \notin \mathcal{X}$, then we know $\mathcal{O} \not\models C(y)$; If $y \notin \mathcal{X}'$, then $\mathcal{O} \not\models C'(y)$. Therefore, $\mathcal{O} \not\models (C \sqcap C')(y)$. So, $C \sqcap C'$ represents $\mathcal{X} \cap \mathcal{X}'$. □

However, the conclusion is not anymore true for set union or set complement. Consider Example 3.3, let $\mathcal{X}_1 = \{f_2\}, \mathcal{X}_2 = \{f_3\}$. We can see that $\mathcal{X}_1$ is represented by the concept B w.r.t. $\mathcal{O}$ and Δ, and $\mathcal{X}_2$ is represented by the concept E w.r.t. $\mathcal{O}$ and Δ. However, we have known that $\mathcal{X}_1 \cup \mathcal{X}_2$ is not representable. Let $\mathcal{X}_3 = \{f_1\}$. Then we have that $\Delta \setminus \mathcal{X}_3 = \{f_2, f_3\}$ is not representable in $\mathcal{ELO}$.

The following lemma tells that any concept naturally represents a special set of individuals. That is, every concept can represent some set of individuals given in an ontology.

Lemma 3.2. *Given a concept* C*, an ontology* $\mathcal{O}$*, the set* Δ *of individuals in* $\mathcal{O}$*,* C *represents the set of individuals* $S = \{x \in \Delta \mid \mathcal{O} \models C(x)\}$*.*

Proof. By the definition o,f S, we know that for any $x \in S$, $\mathcal{O} \models C(x)$, but for any $y \in \Delta \setminus S$, $\mathcal{O} \not\models C(x)$. Then by Definition 3.14, C represents the set S w.r.t. $\mathcal{O}$ and Δ. □

Example 3.5 (Example 3.3 contd.)**.** *A represents the set* $\{f_1\}$*, B represents the set* $\{f_2\}$*, and E represents represents the set* $\{f_3\}$*. And the concept* $A \sqcap B$ *represents* $\emptyset$*.*

Note that when a concept represents an empty set of individuals of an ontology, it means that this concept is irrelevant to characterize the properties of the individuals from this ontology. Hence, from now on, we are interested in only the concepts that represent a non-empty set.

Proposition 3.2. *Given an $\mathcal{ELO}$ ontology $\mathcal{O}$, a set Δ of individuals, a concept C and a set $\mathcal{X} \subseteq \Delta$, the decision problem:*

$$\textit{Does } C \textit{ represent } \mathcal{X} \textit{ w.r.t. } \mathcal{O}?$$

can be solved in ***P-Time****.*

Proof. By Definition 3.14, C represents $\mathcal{X}$ w.r.t. $\mathcal{O}$ iff:

- For each $x \in \mathcal{X}$, we find $\mathcal{O} \models C(x)$ and
- For each $y \in \Delta \setminus \mathcal{X}$, we find $\mathcal{O} \not\models C(y)$

To verify the above conditions we need to perform instance checking for each $i \in \Delta$. That is, we need at most $|\Delta|$ instance checking operations. Since instance checking in $\mathcal{ELO}$ is polynomial, the whole procedure to verify if C represents $\mathcal{X}$ w.r.t. $\mathcal{O}$ is still in polynomial time. □

Proposition 3.3. *Let $\mathcal{O}$ be an $\mathcal{ELO}$ ontology and Δ the set of individuals in $\mathcal{O}$. For a given set of individuals $\mathcal{X} \subseteq \Delta$, and an integer $n > 0$, the decision problem* $\mathsf{Representability}_n$*:*

$$\textit{Is there a concept } C \textit{ with } |C| < n \textit{ that represents } \mathcal{X} \textit{ w.r.t. } \mathcal{O}?$$

can be solved in ***ExpTime****.*

Proof. We can have at most $(|N_c| + |N_r| + |\Delta| + |\mathsf{Op}|)^n$ concepts constructed from a finite concept set N_c, a finite role set N_r, a finite set of individuals Δ, and a finite set of constructors Op[3]. By Proposition 3.2, deciding if each concept is a representative concept of $\mathcal{X}$ can be done in P-time. Therefore, deciding if there is a concept whose length is less than n can be decided in exponential time. □

By the proof of Proposition 3.3, if n is bounded by a constant, then the problem $\mathsf{Representability}_n$ is solvable in **P-Time**.

Note that a set of instances $\mathcal{X}$ can be represented by several concepts, and that their number might even be infinite (Example 3.6). To avoid dealing with an infinite number of concepts, we use the notion of equivalence classes. The concepts representing $\mathcal{X}$ are equivalent in the sense of their instances, and thus they define a class of equivalent concepts. Each one of these classes is called a *situation* in $\mathcal{O}$ and it suffices to provide one concept belonging to the class to characterize it. Therefore to define a situation, our problem is reduced to finding a single representative for the situation instead of finding all of the concepts that comprise it.

[3]The constructors of the underlying language, like conjunction ($\sqcap$) or existential restriction ($\exists$). If the parenthesis are taken into account, this adds at most 2^{n-1} to the number of concepts constructed, which remains exponential w.r.t. $|C|$.

Definition 3.15 (Situation in $\mathcal{O}$)**.** *Given an ontology $\mathcal{O}$, a set Δ of individuals in $\mathcal{O}$, and a set $\mathcal{X} \subseteq \Delta$, a* situation *for $\mathcal{X}$ in $\mathcal{O}$ is the set:*

$$||\mathcal{X}||^{\mathcal{O}}_{\Delta} = \{C \mid C \text{ represents } \mathcal{X} \text{ w.r.t. } \mathcal{O} \text{ and } \Delta\}.$$

We call Δ the domain of the situation.

When the ontology $\mathcal{O}$ and the set of individuals Δ are clear from the context, $||\mathcal{X}||^{\mathcal{O}}_{\Delta}$ is shortened as $||\mathcal{X}||$. By an abuse of symbol, we also use a situation to refer to an element from $||\mathcal{X}||$.

Intuitively, a situation in $\mathcal{O}$ explicitly characterizes, via concept descriptions, a given set of individuals in the ontology.

Additionally, among the concepts that belong to a situation there might be some of them that are not equivalent in the classical sense (i.e. w.r.t. subsumption), as illustrated by Example 3.6.

Example 3.6. *Consider again example 3.3 and the following T-Box:*

$$\begin{aligned} \mathcal{T}_2 = \{ \quad & C_1 \equiv \exists r.\top, \\ & C_2 \equiv \exists r.\{f_3\}, \\ & C_3 \equiv \exists r.\top \sqcap \exists r.\{f_3\}\} \end{aligned}$$

Then, C_1, C_2, C_3 all represent $\mathcal{X}$. Thus $\{C_1, C_2, C_3\} \in ||\mathcal{X}||$. Note that another concept $C_4 \equiv \exists r.\top \sqcap \exists r.\top \sqcap \cdots \sqcap \exists r.\top$ would also represent $\mathcal{X}$, this shows that the set $||\mathcal{X}||$ could be infinite, even with very simple DLs. Furthermore note that $C_1 \not\equiv C_2 \not\equiv C_3$, but $\{C_1, C_2, C_3\} \in ||\mathcal{X}||$, thus concept equivalence is strictly more specific than the notion of situations.

Indeed, the notion of situation is more general than the notion of standard equivalence class as shown in the following conclusion.

Proposition 3.4. *Given an ontology $\mathcal{O}$ and the set Δ of individuals in $\mathcal{O}$, we assume $\mathcal{O} \models A \equiv B$. Then we have that $A \in ||\mathcal{X}||$ if and only if $B \in ||\mathcal{X}||$ for any $\mathcal{X} \subseteq \Delta$.*

Proof. Since $\mathcal{O} \models A \equiv B$, it holds that $\mathcal{O} \models A(x)$ iff $\mathcal{O} \models B(x)$ for any individual $x \in \Delta$. By Definition 3.14, A represents a set of individuals $\mathcal{X}$ iff B represents a set of individuals $\mathcal{X}$. □

Note that not every subset of individuals can lead to a (non-empty) situation. For instance, the set $\mathcal{X}' = \{f_2, f_3\}$ from Example 3.3 can not be represented under $\mathcal{ELO}$, therefore the set $||\mathcal{X}'||$ is empty.

Assume we are given a set of individuals $\mathcal{X} \subseteq \Delta$, and we need to determine weather a situation representing $\mathcal{X}$ exists. If the response is no, to ensure a complete and sound answer, we would require to access all possible situations in

$\mathcal{O}$, and for each of them, test if it represents $\mathcal{X}$. We could also be interested in all the possible ways we could use DLs to discriminate between the individuals in $\mathcal{X}$, to extract a particular set. It is then natural to ask for the subsets of $\mathcal{X}$ for which a DL representation exists, thus discovering the situations in $\mathcal{O}$.

Definition 3.16 (Situation discovery problem)**.** *Let $\mathcal{O}$ be an ontology and Δ a set of individuals in $\mathcal{O}$. For $\mathcal{X} \subseteq \Delta$, the situation discovery problem is to compute the following set: $\Xi_{\mathcal{O}}(\mathcal{X}) = \{\mathcal{X}_1, \ldots, \mathcal{X}_n \mid \mathcal{X}_i \subseteq \mathcal{X}, ||\mathcal{X}_i||_{\mathcal{O}} \neq \emptyset\}$. That is, to find all the subsets of $\mathcal{X}$ that are representable w.r.t. $\mathcal{O}$.*

We also shorten $\Xi_{\mathcal{O}}(\mathcal{X})$ as $\Xi(\mathcal{X})$ when the ontology $\mathcal{O}$ is clear from the context. Since each $\mathcal{X}_i \in \Xi_{\mathcal{O}}(\mathcal{X})$ leads to a situation $||\mathcal{X}_i||_{\mathcal{O}}$, we will also call such $\mathcal{X}_i$ a situation by an abuse of terminology.

Lemma 3.3. *Let $\mathcal{O}$ be an ontology and Δ be a set of individuals in $\mathcal{O}$. For any $\mathcal{X} \subseteq \Delta$, $\Xi(\mathcal{X}) \neq \emptyset$.*

Proof. By Lemma 3.1, it is easy to see that $\emptyset$ is representable by $\bot$, so $\emptyset \in \Xi(\mathcal{X})$. □

Lemma 3.3 shows that we can compute at least one situation with no computation cost. Henceforth, we omit this trivial situation in the rest of this thesis.

Moreover, Example 3.4 shows that it happens that $\mathcal{X}$ is not representable, but $\Xi(\mathcal{X})$ contains subsets of individuals that are nevertheless representable, such as $\mathcal{X}_1 = \{f_3\}$ having a representative concept E.

Definition 3.17. *Let $\mathcal{O}$ be an $\mathcal{ELO}$ ontology and Δ a set of individuals in $\mathcal{O}$. For a given set of individuals $\mathcal{X} \subseteq \Delta$ and an integer $n > 0$, the decision problem SD_n is defined as follows:*

Does there exist a situation C for some $\mathcal{X}' \subseteq \mathcal{X}$ in $\mathcal{O}$ with $|C| \leq n$?

If the answer to SD_n is positive, it means that there exists a nonempty subset $\mathcal{X}' \subseteq \mathcal{X}$ such that C is a representative concept for $\mathcal{X}'$ w.r.t. $\mathcal{O}$ and $|C| \leq n$.

Proposition 3.5. *SD_n is in **ExpTime**.*

Proof. For a given subset of individuals $\mathcal{X}'$, by Lemma 3.3, checking if there exists a concept of size less than n that represents $\mathcal{X}'$ w.r.t. $\mathcal{O}$ can be achieved in ExpTime, which is the $\mathsf{Representability}_n$ problem. Checking $\mathsf{Representability}_n$ for all possible $\mathcal{X}' \subseteq \mathcal{X}$ requires at most $2^{|\mathcal{X}|}$ checks, thus we can decide SD_n in exponential time w.r.t. to size of $\mathcal{X}$ and n. □

According to the proof, if the $|\mathcal{X}|$ and n are bounded by a constant, SD_n is in **P-Time**.

The following conclusion shows that for a set of individuals $\mathcal{X}$, the $\Xi(\cdot)$ operator satisfies monotonicity in the sense that (1) the set of concepts representing $\mathcal{X}$ might decrease when the situation domain increases; (2) a concept A that characterizes $\mathcal{X}$ still characterizes some set of individuals when the situation domain increases. Nevertheless, the set of individuals that A characterizes might no longer be $\mathcal{X}$. In fact, it could be the case that $\mathcal{X}$ is no longer representable.

Proposition 3.6. *Let $\mathcal{O}$ be an ontology and Δ be a set of individuals in $\mathcal{O}$. Consider $\Delta_1 \subseteq \Delta$. Suppose that $\mathcal{X} \in \Xi(\Delta_1)$ is represented by a concept A w.r.t. $\mathcal{O}$ and Δ_1. Then the following conclusions hold:*

1. $||\mathcal{X}||_{\Delta} \subseteq ||\mathcal{X}||_{\Delta_1}$
2. *$\mathcal{X}$ is not necessarily representable w.r.t. Δ.*
3. *The concept A still represents some set of individuals $\mathcal{X}'$, that is, $\mathcal{X}' \in \Xi(\Delta)$.*

Proof. To prove $||\mathcal{X}||_{\Delta} \subseteq ||\mathcal{X}||_{\Delta_1}$, we suppose $C \in ||\mathcal{X}||_{\Delta}$. Then it holds that $\mathcal{O} \models C(x)$ for any $x \in \mathcal{X}$ and $\mathcal{O} \not\models C(y)$ for any $y \in \Delta \setminus \mathcal{X}$. Since $\Delta_1 \subseteq \Delta$, we have $\mathcal{O} \models C(x)$ for any $x \in \mathcal{X}_1$ and $\mathcal{O} \not\models C(y)$ for any $x \in \Delta_1 \setminus \mathcal{X}$. Thus C represents $\mathcal{X}$ w.r.t. Δ_1, that is, $C \in ||\mathcal{X}||_{\Delta_1}$.

To see that $\mathcal{X}$ is not necessarily still representable w.r.t. $\mathcal{X}_2$, we consider the following example extended from Example 3.3: $\mathcal{O}' = \mathcal{O} \cup \{B(f_4), r(f_4, f_3)\}, \Delta = \{f_1, f_2, f_3, f_4\}, \Delta_1 = \{f_1, f_2, f_3\}$, and $\mathcal{X} = \{f_1, f_2\}$. We know from Example 3.3 that $\mathcal{X}$ is represented by C w.r.t. $\mathcal{O}$ and Δ_1. However, $\mathcal{X}$ is not representable w.r.t. Δ because $\mathcal{O} \models f_2 \equiv f_4$, then there is no concept D that can satisfy both $\mathcal{O} \models D(f_2)$ and $\mathcal{O} \not\models D(f_4)$.

To prove the third item, we define $\mathcal{X}' = \mathcal{X} \cup \{x \in \Delta \mid \mathcal{O} \models A(x)\}$. Obviously, $\mathcal{X}' \subseteq \Delta$. For each $x \in \mathcal{X}'$, if $x \in \mathcal{X}$, we know $\mathcal{O} \models A(x)$ since A represents $\mathcal{X}$; if $x \in \mathcal{X}' \setminus \mathcal{X}$, then it is obvious that $\mathcal{O} \models A(x)$ by the definition of $\mathcal{X}'$. For each $y \in \Delta \setminus \mathcal{X}'$, if $y \in \Delta_1 \setminus \mathcal{X}$, then $\mathcal{O} \not\models A(y)$ since A represents $\mathcal{X}$ w.r.t. $\mathcal{O}$ and Δ; if $y \notin \Delta_1$, since $y \notin \mathcal{X}'$, then $y \notin \{x \in \Delta \mid \mathcal{O} \models A(x)\}$. Hence, $\mathcal{O} \not\models A(y)$. In short, A represents $\mathcal{X}'$ w.r.t. $\mathcal{O}$ and Δ. □

3.3 Computing Situations in An Ontology

In this section we present the algorithms that allow us to extract the situations present in an ontology. To this end, we proceed in several steps:

1. First, we detail how to obtain a single DL concept representing a situation in $\mathcal{O}$. For this we define a refinement operator that allows to traverse the space of situations in $\mathcal{O}$. The operator is guided by an individual $x \in \Delta$ and constructs concepts that describe situations in $\mathcal{O}$ to which x belongs.

2. Once we count with the means to discover situations for a single individual x, we extend the process to extract situations from a whole set of individuals $\mathcal{X}$, based on the refinement operator.

3. Finally, once we can describe a set of individuals $\mathcal{X}$, we show how to find the situations for the sub-sets $\mathcal{X}' \subset \mathcal{X}$ for which a DL description exists.

3.3.1 Refinement Operator

To obtain the concepts describing the situations in $\mathcal{X}$, we start with a "root" concept C, that captures all individuals in $\mathcal{X}$ and proceed to refine it. These refinements will at some point separate the sub-sets in $\mathcal{X}$ that can be described, thus discovering the situations in $\mathcal{X}$.

To refine a concept, we need a way to traverse the space of concept expressions in an ordered manner. This is where the refinement operator comes into play. In this section, we present an algorithm to obtain the refinements of a concept $\mathcal{C}$ that describe a set of given samples.

In our approach, we first select an instance $x \in \mathcal{X}$, obtain its minimal A-Box and establish the possible extensions. Each extension represents a property of x (directly or indirectly) and for each such extension a set of possible DL expressions can be constructed. Each DL expression obtained this way is then "added" to the original concept C to construct the refinements. The steps of the refinement process are outlined in Figure 3.2 and are specified in Algorithm 3.2. To detail the process, we first need some basic notions given in the next subsection.

3.3.1.1 Basic Notions

In this section we first define the notion of a *minimal A-Box*, which tells us properties of x that are necessary to validate the instantiation of x with respect to a given concept C. These properties will be used by the refinement operator to define the extensions of x and propose the refinements of C. Since $\mathcal{ELO}$ allows for nominals, we would like to make this information explicit in the A-Box. Thus in the following we assume that each node a in the graph representation of the A-Box, has attached the corresponding nominal $\{a\}$ (i.e. $\{a\}(a) \in$ A-Box).

Definition 3.18 (Minimal A-Box)**.** *Let $\mathcal{O} = (T, A)$ be an ontology and $C(x)$ be an instance assertion implied by $\mathcal{O}$, i.e. $\mathcal{O} \models C(x)$. An A-Box A' is called a*

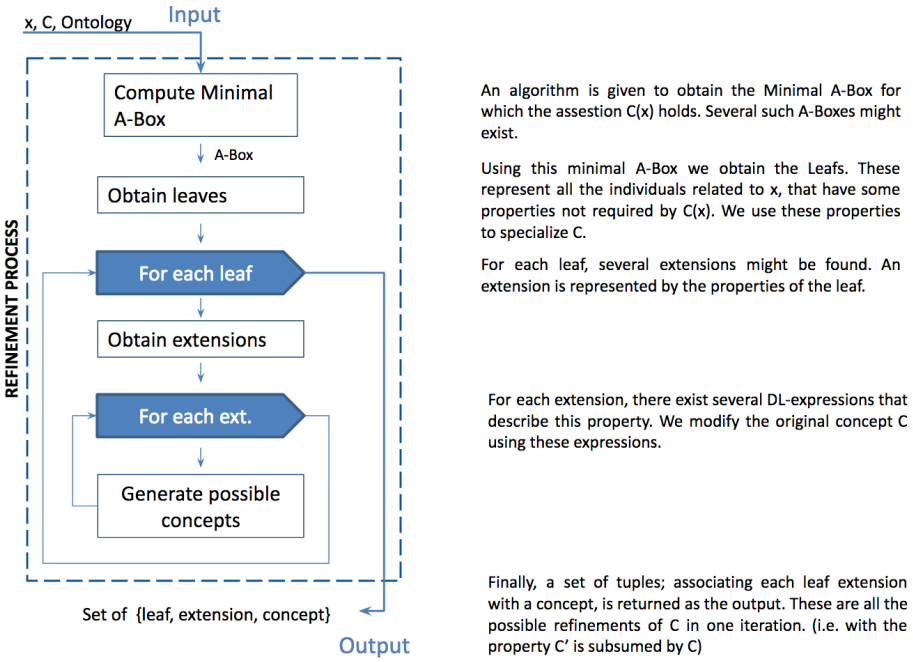


Figure 3.2: Steps to refine a concept C guided by an instance x and an ontology $\mathcal{O}$; The output is the concepts obtained by one refinement step of C.

minimal A-Box for $C(x)$ *w.r.t.* $\mathcal{O}$ *(in symbols* A_x^C*) if (1)* $T \cup A' \models C(x)$*, and (2) there is no A-Box* $A'' \subset A'$ *such that* $T \cup A'' \models C(x)$*.*

These properties identified by a minimal A-Box involving x or any other individual connected to x, called extensions. These extensions are used to construct DL expressions and obtain the refinements of C.

To facilitate the definition of the refinement operator, in the remaining of this chapter we assume that the materialization of the ontology [Kontchakov et al., 2011, Glimm et al., 2017] has been done first, i.e. all concept and role inferences are explicitly stated in the A-Box. In this case, the minimal A-Box is simplified as below.

Definition 3.19. *Let* A *be an A-Box and* $C(x)$ *be an instance assertion implied by* A*, i.e.* $A \models C(x)$*. An A-Box* A' *is called a minimal A-Box for* $C(x)$ *if (1)* $A' \models C(x)$*, and (2) there is no A-Box* $A'' \subset A'$ *such that* $A'' \models C(x)$*.*

Let us illustrate the intuition behind this process. Any individual in an ontology $\mathcal{O}$ and its relations to other individuals can be represented as a directed (acyclic) graph, with the nodes being the individuals and the edges being the relations between them, as stated by the A-Box (figure 3.3). For example consider

the A-Box :

$$\mathcal{A}_1 : \{r_1(x, y), r_2(y, w), r_3(y, z)\}$$

we obtain:

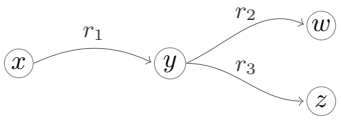


Figure 3.3: The graph representation of individual x w.r.t. $\mathcal{A}_1$.

Assume a concept $C \equiv \exists r_1.\exists r_2.\top$, we find that x is an instance of C. We can determine which are the properties of x that are necessary for $C(x)$ to hold. Both $r_1(x, y)$ and $r_2(y, w)$ are necessary, but the assertion $r_3(y, z) \in \mathcal{A}_1$ is not relevant for deciding whether $C(x)$ holds. From this observation, a simpler representation of x can be obtained , i.e. the A-Box : $\mathcal{A}_1 \setminus \{r_3(y, z)\}$, for which x still remains an instance of C. This can be seen as a cut in the graph representation of x (figure 3.4) between the necessary and unnecessary assertions in the A-Box. This

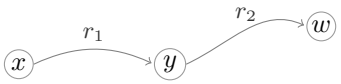


Figure 3.4: The graph representation of individual x, w.r.t. $\mathcal{A}_1 \setminus \{r_3(y, z)\}$.

representation allows us to establish up to which point the assertions in $\mathcal{A}_1$ are relevant for $C(x)$ to hold. All the necessary assertions comprise a minimal A-Box for which $C(x)$ holds, and all unnecessary assertions provide the basis to further specialize C and still capture x. These specializations are the refinements we are looking for. In order to isolate and manipulate these elements, first we need to establish all individuals connected to x. This is formalized in the following definition:

Definition 3.20 (Binary relation closure)**.** *Given an ontology $\mathcal{O}$, two individuals x, y and an A-Box $\mathcal{A}$, we say that there exists a* binary relation closure *between x and y, denoted by $r \uparrow (x, y)$, if $x = y$ or if there exists a path of the form:*

$$\{r_1(x, z_1), r_2(z_1, z_2), \ldots, r_m(z_{n-1}, z_n)\} \subseteq \mathcal{A}$$

$$\textit{with } n \geq 1,\ z_n = y,\ \textit{and } r^j \textit{ a role in } \mathcal{O}$$

Next, we want to establish which of the edges of the graph representing individual x are necessary for a given concept C to capture it. To this end we introduce the notion of necessity:

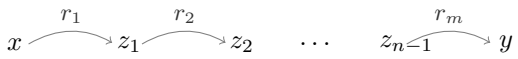


Figure 3.5: The representation of a path from individual x to individual y.

Definition 3.21 (Necessity)**.** *Given a concept C, an instance x of C, an A-Box $\mathcal{A}$ and a role $r(y,z) \in \mathcal{A}$ with $r \uparrow (x,y)$. We say that:*

1. *Role $r(y,z)$ is* necessary *for $C(x)$ iff:*

$$\mathcal{A} \setminus \{r(y,z)\} \not\models C(x)$$

We say $r(y,z)$ is unnecessary *for $C(x)$ otherwise.*

2. *Individual i is* necessary *for $C(x)$ iff:*

$$i = x \text{ or}$$

$$\text{there exists } z \text{ such that } r(z,i) \in \mathcal{A} \text{ and } r(z,i) \text{ is necessary.}$$

We say i is unnecessary *for $C(x)$ otherwise.*

3. *Given a necessary individual i, then a concept assertion $D(i) \in \mathcal{A}$ is* necessary *for $C(x)$ iff:*

$$\mathcal{A} \setminus \{D(i)\} \not\models C(x)$$

We say $D(i)$ is unnecessary *for $C(x)$ otherwise.*

Note that depending on the concept C and the content of the A-Box, a unnecessary role might become necessary, therefore several possible answers might exist. For example take the concept $C \equiv \exists r.\top$ and the A-Box: $\mathcal{A} = \{r(x,y), r(x,z)\}$, then we have:

$$C(x) = \top \text{ w.r.t } \mathcal{A} \setminus r(x,y)$$

We conclude that $r(x,y)$ is not necessary, but only as long as $r(x,z) \in \mathcal{A}$ (and vice-versa).

Given a definition of the necessary assertions for an individual x to be an instance of a concept C, we can also obtain those assertions that are not necessary. These (unnecessary) assertions are linked to the individual x but are not required by C. As such they can be seen as candidate properties for specializing C. These are the assertions about special individuals hereafter called leaves of x, defined by:

Definition 3.22 (Leaves)**.** *Given a concept C, an instance x of C, and an A-Box $\mathcal{A}$, the set of* leaves *of x w.r.t C is given by:*

$$\begin{aligned} &Leaves_{x,C} = \{y \mid B(y) \in \mathcal{A}, y \text{ is necessary, } B(y) \text{ is unnecessary for } C(x)\} \\ &\cup \{y \mid r(y,z) \in \mathcal{A} \text{ where } y \text{ is necessary, but } r(y,z) \text{ is unnecessary for } C(x)\} \end{aligned}$$

Intuitively the set $Leaves_{x,C}$ represents all those individuals y in the frontier of x w.r.t. C, in the sense that no further edges of the graph representing x are considered by C to decide whether the individual x belongs to it. In our example $r_3(y, z)$ is not necessary.

Definition 3.23 (Extensions)**.** *Given an individual x, a concept C and the set $Leaves_{x,C}$ w.r.t. an A-Box $\mathcal{A}$, the set of* extensions $Ext_{x,C}$ *to specialize C w.r.t. x is defined by:*

$$Ext_{x,C} = \{r(y, z) \in \mathcal{A} \mid y \in Leaves_{x,C} \text{ and}$$

$$r(y, z) \text{ is unnecessary for } C(x) \} \cup$$

$$\{B(y) \in \mathcal{A} \mid y \in Leaves_{x,C}, B(y) \text{ is unnecessary for } C(x) \}$$

The set $Ext_{x,C}$ captures all those assertions through which C can be further specialized. The set of leaves and their extensions allow us to compute the refinements of C.

3.3.1.2 Computing Minimal A-Box

To obtain the refinements of a concept C, we first introduce an algorithm that computes a minimal A-Box.

We now introduce an algorithm to compute a minimal set of necessary assertions, from which we can extract $Leaves_{x,C}$ and $Ext_{x,C}$ to construct the refinements. The algorithm takes as input an individual x, a concept C and an A-Box $\mathcal{A}$. It will test all individuals connected to x for necessity, and output a set of necessary assertions for $C(x)$ to hold under $\mathcal{A}$. To illustrate the algorithms behavior consider the concept

$$\mathcal{C} \equiv \exists r.\top \text{ and the A-Box } \mathcal{A} = \{r(x, y), r(y, w), r(x, z)\}$$

as inputs. In Algorithm 1 we first compute $\mathcal{R}_x$ (Line 2), which is the subset of the A-Box $\mathcal{A}$ containing all those role assertions in a path from x:

$$r(x, y) \in \mathcal{R}_x \text{ since } r \uparrow (x, x) \text{ and } r(x, y) \in \mathcal{A}$$

$$r(y, w) \in \mathcal{R}_x \text{ since } r \uparrow (x, y) \text{ and } r(y, w) \in \mathcal{A}$$

$$r(x, z) \in \mathcal{R}_x \text{ since } r \uparrow (x, x) \text{ and } r(x, z) \in \mathcal{A}$$

$$\text{yields } \mathcal{R}_x = \{r(x, y), r(y, w), r(x, z)\}$$

(3) makes a copy $Copy_{\mathcal{R}}$ of $\mathcal{R}_x$ from which we will sequentially remove the last elements of each path. (4) establishes as the candidates $Cand_{\mathcal{R}}$ to be tested all

Algorithm 1 MinAbox

input: $(x, C(x) = \top, \mathcal{A})$
$\mathcal{R}_x = \{r(y,z) \mid r \uparrow (x,y), r(y,z) \in \mathcal{A}\}$
$Copy_{\mathcal{R}} = \mathcal{R}_x$
$Cand_{\mathcal{R}} = \{r(y,z) \in \mathcal{R}_x \mid \nexists w \text{ s.t. } r(z,w) \in \mathcal{R}_x\}$
$\mathcal{C}_x = \{D(z) \in \mathcal{A} \mid z = x \text{ or } \exists y \text{ s.t. } r(y,z) \in \mathcal{R}_x\}$
while $Cand_{\mathcal{R}} \neq \emptyset$ **do**
    for $r(y,z) \in Cand_{\mathcal{R}}$ **do**
        if $C(x) = \top$ w.r.t. $(\mathcal{R}_x \setminus \{r(y,z)\}) \cup \mathcal{C}_x$ **then**
            remove $r(y,z)$ from $\mathcal{R}_x$
        end if
        remove $r(y,z)$ from $Copy_{\mathcal{R}}$
    end for
    $Cand_{\mathcal{R}} = \{r(y,z) \in Copy_{\mathcal{R}} \mid \nexists w \text{ s.t. } r(z,w) \in Copy_{\mathcal{R}}\}$
end while
for $D \in \mathcal{C}_x$ **do**
    if $C(x) = \top$ w.r.t. $(\mathcal{C}_x \setminus \{D\}) \cup \mathcal{R}_x$ **then**
        remove D from $\mathcal{C}_x$
    end if
end for
return: $\mathcal{A}_x = \mathcal{R}_x \cup \mathcal{C}_x$

those role assertions that do not have further outgoing edges (that is, the last elements of a path):

$$Cand_{\mathcal{R}} = \{r(y,w), r(x,z)\}$$

(5) creates the set of all concept assertions about all the individuals that take part in a role in $\mathcal{R}_x$, this is necessary to establish if a concept assertion is necessary for the final A-Box. In our example this set is empty. Then the *while* loop starts, which tests all assertions for necessity until no more candidates are found. (7) takes one candidate at the time, and (8) tests if it is necessary. Recall that a role assertion is unnecessary if $C(x)$ holds when the assertion is removed from the A-Box. The unnecessary assertions are removed from $\mathcal{R}_x$ (9), which at the end of the algorithm will contain only necessary role assertions. Any assertion already tested is removed from $Copy_{\mathcal{R}}$ by (11) in order no to test them twice. In our example we test $Cand_{\mathcal{R}} = \{r(y,w), r(x,z)\}$ for necessity:

Initially $\mathcal{R}_x$	$= \{r(x,y), r(y,w), r(x,z)\}$
Test $r(y,w)$ for necessity :	
$C(x)$	$= \top$ w.r.t. $\{\mathcal{R}_x \setminus r(y,w)\} \cup \mathcal{C}_x$, then we remove $r(y,w)$
$\mathcal{R}_x$	$= \{r(x,y), r(x,z)\}$
$Copy_{\mathcal{R}}$	$= \{r(x,y), r(x,z)\}$
Test $r(x,z)$ for necessity :	
$C(x)$	$= \top$ w.r.t. $\{\mathcal{R}_x \setminus r(x,z)\} \cup \mathcal{C}_x$, then we remove $r(x,z)$
$\mathcal{R}_x$	$= \{r(x,y)\}$
$Copy_{\mathcal{R}}$	$= \{r(x,y)\}$

Once all identified candidates are tested, the set $Cand_{\mathcal{R}}$ is re-computed (13) considering only those assertions remaining in $Copy_{\mathcal{R}}$,

$$Cand_{\mathcal{R}} = \{r(x,y)\}$$

A second run of the while loop yields:

Test $r(x,y)$ for necessity :	
$C(x)$	$= \bot$ w.r.t. $\{\mathcal{R}_x \setminus r(x,y)\} \cup \mathcal{C}_x$, then we keep $r(x,y)$
$\mathcal{R}_x$	$= \{r(x,y)\}$
$Copy_{\mathcal{R}}$	$= \{\}$

Since there are no more candidates to test for necessity the *while* loop ends and the modified set $\mathcal{R}_x$ is a minimal set of necessary role assertions for $\mathcal{C}(x) = \top$.

Until this point we have tested all role assertions in $\mathcal{A}$ for necessity, but we have kept all class assertions $\mathcal{C}_x$ related to x. The final *for* loop (15) tests each class assertion in $\mathcal{C}_x$ for necessity, to keep only the necessary ones. This is where

the algorithms stops, and the output $\mathcal{A}_{x,C}$ (20) is then a set of minimal role and class assertions, a minimal A-Box, necessary for C_x to hold:

$$\mathcal{A}_{x,C} = \{r(x,z)\}$$

From the set $\mathcal{A}_{x,C}$ we can easily construct $Leaves_{x,C}$ and $Ext_{x,C}$, following their definitions. Consider again: $C \equiv \exists r.\top$ and $\mathcal{A} = \{r(x,y), r(y,w), r(x,z)\}$, we have the output (of one run) of algorithm 1 $\mathcal{A}_{x,C} = \{r(x,y)\}$ from where we obtain:

$$Leaves_{x,C} = \{x, y\}$$

and

$$Ext_{x,C} = \{r(y,w), r(x,z)\}$$

where the possible extensions to specialize C are $r(y,w), r(x,z)$, which is the intended answer.

Soundness of Algorithm 1 . We show that the output of Algorithm 1, $\mathcal{A}_{x,C}$, a minimal A-Box that entails $C(x)$. The first part of the proof, follows from Lines 8 and 16 in Algorithm 1 which remove from $\mathcal{A}_{x,C}$ only those assertions that are unnecessary for $C(x)$. That is, assertions will be removed from $\mathcal{A}_{x,C}$ only as long as $\mathcal{A}_{x,C} \models C(x)$, thus at the end of the process $\mathcal{A}_{x,C} \models C(x)$ is ensured. For the second part, we proceed by contradiction. If $\mathcal{A}_{x,C}$ is not minimal, it implies that there exists another A-Box $A' \neq A$, with $A' \subset \mathcal{A}_{x,C}$ and $A' \models C(x)$ (A' is minimal). Since $A' \subset \mathcal{A}_{x,C}$ there must be an assertion $\beta \in \mathcal{A}_{x,C}$ such that $\beta \notin A'$. Lines 8 and 16 in Algorithm 1 test each assertion for necessity. Therefore $\beta \in \mathcal{A}_{x,C}$ implies that $\mathcal{A}_{x,C} \setminus \beta \not\models C(x)$. $\mathcal{ELO}$ is monotonic. Thus, $\mathcal{A}_{x,C} \setminus \beta \not\models C(x)$ implies $A' \setminus \beta \not\models C(x)$, because $A' \subset \mathcal{A}_{x,C}$. Since $\beta \notin A'$, $A' \setminus \beta = A'$. Thus $A' \not\models C(x)$, a contradiction. □

Note that, a minimal A-Box is not unique, but several might exist. The other minimal A-Boxes can be obtained by subsequent runs of algorithm 1 in the fashion of [Kalyanpur et al., 2007] via Hitting Set Tree search.

The minimal A-Box allows to select the unnecessary assertions, which can be used to construct concepts that represents situations in the ontology.

3.3.1.3 Computing Refinements

In this section, we introduce the process of computing concept refinements. From a constructive point of view, a refinement $\mathcal{C}'$ is a modified version of $\mathcal{C}$, where a part of the original concept $\mathcal{C}$ has been replaced by a more specific $\mathcal{ELO}$

expression. This modification depends on the *leaf* and the *extension* selected. The output of Algorithm 2 is a set of refinements of the form:

$$\alpha_x(\mathcal{C}) = \{\mathcal{C}'_1, \ldots, \mathcal{C}'_n\}.$$

Before we present the algorithm for the refinement process, we need to introduce the notion of *reason for a leaf*. This is important because in the refinement process we start by a concept C, and construct new concept expressions by modifying C. The modifications are related to the leafs, and take place in a specific position within C. Thus, the link between a position in C and a leaf, has to be established. This notion is captured by the following definition.

Definition 3.24 (Leaf reason)**.** *Given a complex concept C, the set $\{D_1, \ldots, D_n\}$ of its sub-concepts, an instance x of C and a leaf $l \in Leaves_{x,C}$. An let D_i be the shortest concept for which it holds:*

$$l \text{ is necessary for } C(x) \text{ , but}$$

$$l \text{ is unnecessary for } C|_{D_i \to \top}(x)$$

The reason *for l, in symbols $\mathcal{R}_l$, is defined by:*

$$\text{if } D_i \text{ is of the form } \exists r.B, \text{ then } \mathcal{R}_l = B, \text{ or}$$

$$\mathcal{R}_l = D_i \text{ otherwise.}$$

For the case $l = x$ we have $\mathcal{R}_l = C$.

The idea behind the refinement process is that given the leafs for $C(x)$, we can use the unnecessary properties of the leafs (role and class assertions) to obtain specializations of C. To this end, we must first identify the part of the complex concept C that refers to a leaf l (that is the reason for l), and then by adding constraints to this reference, we ensure the resulting concept requires more properties from l than the original C.

Example 3.7 (Reason of a leaf)**.** *Consider the following A-Box:*

$$A_4 = \{r_1(x, y), r_2(x, z), A(y), B(y), D(z)\}$$

and the concept:

$$C \equiv \exists r_1.A \sqcap \exists r_2.D$$

where its sub-concepts are:

$$\begin{aligned} C'_1 &\equiv \exists r_1.A \sqcap \exists r_2.D \\ C'_2 &\equiv \exists r_1.A \\ C'_3 &\equiv \exists r_2.D \\ C'_4 &\equiv A \\ C'_5 &\equiv D \end{aligned}$$

we find that $C'_4 \equiv A$ is the reason for leaf y, since for the concept

$$C|_{C'_4 \to \top} \equiv \top \sqcap \exists r_2.D$$

where C'_4 is replaced by $\top$, y is no longer necessary nor a leaf. If we would make the replacement using C'_1, y would also become unnecessary, but C'_1 is not the shortest.

Given the reason $C'_4 \equiv A$, if we want to specialize C by establishing that any individual connected to x through role r_1 should not only be of class A ($\exists r_1.A$) but additionally of class B, we can achieve this by replacing C'_4 with a conjunction of C'_4 and B:

$$C' \equiv \exists r_1.(A \sqcap B) \sqcap \exists r_2.D$$

In this way we obtain C' which is a modified version of C, for which it holds:

$$C'(x) \text{ and } C' \sqsubseteq C$$

We now formally define the refinement operator in Algorithm 2.

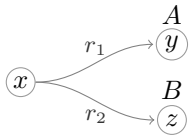
Algorithm 2 Operator α

input: $(x, C, \mathcal{O} = \{\mathcal{T}, \mathcal{A}\})$
$Ref = \emptyset$
$\mathcal{A}' = MinAbox(x, C, \mathcal{A})$
$Leafs_{x,C} = \{l_1, \ldots, l_n\}$
$Ext_{x,C} = \{r_1(l_1, y_1), \ldots, r_m(l_n, y_k), B_1(l_1), \ldots, B_h(l_n)\}$
for $l \in Leaves_{x,C}$ **do**
    Obtain $\mathcal{R}_l$ (the reason for l)
    for $B(l) \in Ext_{x,C}$ **do**
        $C' = C\,|_{\mathcal{R}_l \to \mathcal{R}_l \sqcap B}$
        add C' to Ref
    end for
    for $r(l, z) \in Ext_{x,C}$ **do**
        $C' = C\,|_{\mathcal{R}_l \to \mathcal{R}_l \sqcap \exists r.\top}$
        add C' to Ref
    end for
end for
return: Ref

Let us use the following example to illustrate Algorithm 2.

Example 3.8 (Refinements of a concept C)**.** *Consider the following concept C, its instance x and the A-Box A_5 as an input for our algorithm:*

$$
\begin{array}{rcl}
C & \equiv & \exists r_1.\top \\
A_5 & = & \{r_1(x,y), A(y), \\
 & & r_2(x,z), B(z)\}
\end{array}
$$



In Algorithm 2 we initialize the set of refinements Ref to $\emptyset$ (Line 2). Then we compute the minimal A-Box for $C(x)$ (Line 3) using algorithm 1, from where we can obtain the leaves (Line 4) and the extensions (Line 5) of x with respect to C.

$$
\begin{array}{lrl}
(2) & Ref & = \emptyset \\
(3) & A' & = \{r_1(x,y)\} \\
(4) & Leaves_{x,C} & = \{x,y\} \\
(5) & Ext_{x,C} & = \{r_2(x,z), A(y)\}
\end{array}
$$

Next, for each leaf l (6) we first obtain the position in C to make the specialization, that is the reason $\mathcal{R}_l$ (7) for the leaf l. Within the extensions of each leaf, we find two types: concept assertions and role assertions. If the extension is a concept assertion of the form $B(l)$ (8), then the corresponding refinement C' is a copy of C where the reason $\mathcal{R}_l$ has been replaced by the expression $\mathcal{R}_l \sqcap B$ (9). That is, besides imposing that the leaf l is of type $\mathcal{R}_l$, C' imposes now that it additionally has the type B. Each C' is a new concept name, which is added to the set of refinements (10). If the extension is of the form $r(l,z)$, the loop in (12) works in a similar way. The difference is that instead of imposing a new type for l, it imposes that l has to be connected to some individual z through a role r (13).

$$
\begin{array}{lll}
(6) & \text{For } x \in Leaves_{x,C}: & \mathcal{R}_x = C \\
(12) & \text{For } r_2(x,z) \in Ext_{x,C}: & C'_1 \equiv \exists r_1.\top \sqcap \exists r_2.\top \\
(6) & \text{For } y \in Leaves_{x,C}: & \mathcal{R}_y = \top \\
(8) & \text{For } A(y) \in Ext_{x,C}: & C'_2 \equiv \exists r_1.(\top \sqcap A) \\
(17) & Ref = \{C'_1, C'_2\} &
\end{array}
$$

Finally we return the set Ref containing the refinements (17). It is easy to see that C'_1 and C'_2 are proper refinements, since for both it holds:

$$C'_1 \sqsubseteq C \text{ and } C'_2 \sqsubseteq C, \text{ and}$$

$$A_5 \models C'_1(x) \text{ and } A_5 \models C'_2(x)$$

Note that $C'_1 \equiv \exists r_1.\top \sqcap \exists r_2.\top$ imposes that any instance of C'_1 needs to be connected to some individual by a role r_2. From A-Box A_5 we know that $r_2(x,z)$ and $B(z)$, thus a refinement could also impose that the range of such role r_2 should be of type B (i.e. $\exists r_2.B$). This refinement can be obtained, if the operator is iterated.

We consider some properties of a refinement of a concept C, formalized next.

Definition 3.25. *Given an ontology $\mathcal{O}$, a concept C and x, an instance of C, we say that a concept C' is a* refinement *of C if it holds:*

$$\mathcal{O} \models C'(x) \text{ and } C' \sqsubseteq C$$

Additionally we say that C' is a direct refinement *of C, if:*

$$|\mathsf{Sub}(C')| - |\mathsf{Sub}(C)| = 1$$

The operator is intended to provide access to the concepts describing sets of individuals in a top-down manner. That is from more general (capturing more individuals) to more specific (capturing less individuals). This can be achieved by iterating the operator. At each iteration of α, we have to ensure that no intermediate concept expressions are left unexplored, ensuring that the operator is sound and complete. The operator is sound if: x is an instance of each refinement C' of C, and each C' is more specific than C. The operator is complete if no refinements C' are left unexplored. These properties are precised in the following.

Proposition 3.7. *Given an ontology $\mathcal{O} = \{\mathcal{T}, \mathcal{A}\}$, a concept C, an instance x of C, and $\alpha_x(C) = \{C'_1, \ldots, C'_n\}$. Let $C' \in \alpha_x(C)$ be one of the refinements of C. Then it holds:*

$$\mathcal{O} \models C'(x)$$

and

$$C' \sqsubseteq C$$

Proof. For the first part of the proof, concerning $\mathcal{O} \models C'(x)$, we start by the observation that C is of the form:

$$C \equiv D_1 \sqcap D_2 \sqcap \cdots \sqcap D_n$$

Where for $C(x)$ to hold, each $D_i(x)$ must hold as well. Without loss of generality[4], each D_i is a concept of the form:

$$D_i \equiv \exists r_1.\exists r_2.\ldots.\exists r_n.\mathcal{R}_l$$

Since $D_i(x)$ holds, there exists a path $r_1, r_2, \ldots, r_n$ from x to an individual l of type $\mathcal{R}_l$. This implies that the A-Box $\mathcal{A}$ must contain axioms of the form: $r_n(u, l) \in \mathcal{A}$ and $\mathcal{R}_l(l) \in \mathcal{A}$. If $\mathcal{R}_l$ is to be replaced by a new expression $\mathcal{R}'_l$,

[4]The case $D_i = \mathcal{R}_l$ and the cases where each existential restriction might be of the form $\exists.r.(A \sqcap \exists r...)$, the concept $\mathcal{R}_l$ will always refer to the type of leaf l.

resulting in D'_i and C' respectively, we must ensure that l is also an instance of $\mathcal{R}'_l$. That is if $\mathcal{R}'_l(l)$ holds, $D'_i(x)$ and $C'(x)$ hold as well.

Algorithm 2 will replace $\mathcal{R}_l$ by $\mathcal{R}'_l$ in one of the following two forms:

$$\mathcal{R}'_l \equiv \mathcal{R}_l \sqcap B \text{ where } B(l) \in Ext_{x,C}$$

or

$$\mathcal{R}'_l \equiv \mathcal{R}_l \sqcap \exists r_j.\top \text{ where } r_j(l,l') \in Ext_{x,C}$$

Thus, necessarily $\mathcal{O} \models \mathcal{R}'_l(l)$. This ensures that $D_i(x)$ holds, and since this is the only modification in C, we have that $C'(x)$ holds as well.

For the second part of the proposition $C' \sqsubseteq C$, we have to prove that the replacement $\mathcal{R}'_l \sqsubseteq \mathcal{R}_l$. Note that a replacement is always of the form $\mathcal{R}'_l \equiv \mathcal{R}_l \sqcap B$, where B is a (possibly complex) concept expression. Thus we find that $y \in \mathcal{R}'_l \rightarrow y \in \mathcal{R}_l$, but $y \in \mathcal{R}_l \nrightarrow y \in \mathcal{R}'_l$. This implies $\mathcal{R}'_l \subseteq \mathcal{R}_l$, which in DL notation becomes $\mathcal{R}'_l \sqsubseteq \mathcal{R}_l$, from which follows $D'_i \sqsubseteq D_i$ and $C' \sqsubseteq C$. □

Proposition 3.8. *Let C be a concept, x be an instance of C, and $\mathcal{O}$ be an ontology such that $\mathcal{O} \models C(x)$. Then $C' \in \alpha_x(C)$ implies that C' is a* direct refinement *of C.*

Proof. Given C, there exists a minimal A-Box, A_x for x w.r.t. C. If C' is a refinement of C, then it was constructed by α_x using some extension β for x. This implies there exists an axiom $\beta \notin A_x$, but $\beta \in A'_x$, where A'_x is the minimal A-Box for x w.r.t. C'. For each such β, the operator α will replacement a sub-concept $D \in \mathsf{Sub}(C)$, by a new concept D', where D' is of the form $D' \equiv D \sqcap \exists r.\top$ or $D' \equiv D \sqcap B$, for some named concept B. Thus, $|\mathsf{Sub}(D')| - |\mathsf{Sub}(D)| = 1$. Since this is the only change made to C, then it follows $|\mathsf{Sub}(C')| - |\mathsf{Sub}(C)| = 1$. □

Proposition 3.9. *Given an ontology $\mathcal{O} = \{\mathcal{T}, \mathcal{A}\}$ and an individual $x \in \mathcal{O}$. Let $A' \subset \mathcal{A}$, be a tree with root x, and let $C_{A'}$ be the most specific concept for x w.r.t. A', then $C_{A'}$ can be obtained by α_x.*

Proof. Given an individual x, let $A_n \subset A$, where A_n is a tree with root in x. We have to prove that the MSC for A_n can be obtained by α. The MSC need to comply with two conditions. Let C_n be the MSC for A_n, then it holds:

$$A_n \models C_n(x), \text{ and}$$

it does not exist another concept D such that

$$D \sqsubset C_n \text{ and } A_n \models D_n(x)$$

Let A_{n-1} be a subtree of A_n rooted as well in x, with

$$(1) A_n = A_{n-1} \cup \beta$$

where β is an assertion in A, and let C_{n-1} be the MSC of A_{n-1}. Note that because C_{n-1} is the MSC, A_{n-1} is the minimal A-Box for C_{n-1}, and thus necessarily $\beta \in Ext_{x,C_{n-1}}$. If we apply the operator α_x to C_{n-1} , it yields:

$$\alpha_x(C_{n-1}) = \{C'_1, C'_2, \ldots, C'_m\}$$

where some C'_i was constructed using β. Because $C'_i \sqsubset C_{n-1}$, the minimal A-Box of C'_i needs at least all assertions in A_{n-1}. But since C_{n-1} is already the MSC for A_{n-1}, the minimal A-Box for C'_i has to be larger than A_{n-1}:

$$|A_{C'_i}| > |A_{n-1}|$$

From α and (1) we know that $A_{n-1} \cup \beta \models C'_i(x)$, thus $A_{n-1} \cup \beta$ is a minimal A-Box for $C'_i(x)$.

$$A_{C_i} = A_{n-1} \cup \beta = A_n$$

Therefore $A_n \models C'_i(x)$.

For the second condition, assume another concept $D \sqsubset C'_i$ exists with $A_n \models D(x)$. This implies that $A_D > A_{C'_i}$, but since $A_{C'_i} = A_n$ it follows that $A_D > A_n$, and thus D can not be the MSC for A_n. □

3.3.2 Algorithms for Situation Discovery

In this section, we will apply the refinement operator for computing situations, consisting of two components: refinement over one instance and refinement over a set of instances. The first component is used in the second to find situations.

We have seen that multiple concepts can exists in a situation. To characterize the maximal details of a situation, it is interesting to the most specific concept in a situation defined below.

Definition 3.26 (Most Specific Representative MSR)**.** *Given a set of individuals $\mathcal{X} = \{x_1, \ldots, x_n\}$ and the set of its representative concepts $||\mathcal{X}|| = \{\mathscr{S} \mid \mathscr{S}$ represents $\mathcal{X}\}$, the* Most Specific Representative *of the set $\mathcal{X}$, written $\mathsf{MSR}_{\mathcal{X}}$, is the concept $\mathscr{S}_i \in ||\mathcal{X}||$ such that:*

$$\forall \mathscr{S}_j \in ||\mathcal{X}||, \text{ we find } \mathscr{S}_i \sqsubseteq \mathscr{S}_j.$$

Example 3.9 (Most Specific Refinement)**.** *As an example consider $\Delta = \{x, y, z, z'\}$, the set $\mathcal{X} = \{x\}$ and the A-Box:*

$$\mathcal{A}_3 = \{r(x, y), r(z, z'), A(y), B(y), C(z')\}$$

and the concepts:

$$\begin{aligned}
\mathscr{S}_0 \equiv &\quad \exists r.\top & &= \{x, z\} \\
\mathscr{S}_1 \equiv &\quad \exists r.A & &= \{x\} \\
\mathscr{S}_2 \equiv &\quad \exists r.\{y\} & &= \{x\} \\
\mathscr{S}_3 \equiv &\quad \exists r.(A \sqcap B \sqcap \{y\}) & &= \{x\} \\
\mathscr{S}_4 \equiv &\quad \exists r.(A \sqcap B \sqcap \{y\}) \sqcap \exists r.A & &= \{x\}
\end{aligned}$$

We find that $\mathscr{S}_0 \notin ||\mathcal{X}||$ since $z \in \mathscr{S}_0$. In contrast, all other concepts do represent $\mathcal{X}$, thus we have $\mathscr{S}_1, \mathscr{S}_2, \mathscr{S}_3, \mathscr{S}_4 \in ||\mathcal{X}||$ (note that the set $\mathscr{S}_\mathcal{X}$ can be infinite). The subsumption relation between them is given by:

$$\begin{aligned}
\mathscr{S}_3 \sqsubseteq &\quad \mathscr{S}_1, \mathscr{S}_2, \mathscr{S}_4 \\
\mathscr{S}_4 \sqsubseteq &\quad \mathscr{S}_1, \mathscr{S}_2, \mathscr{S}_3 \\
\mathscr{S}_3 \equiv &\quad \mathscr{S}_4
\end{aligned}$$

Two of these concepts are more specific than the rest: $\mathscr{S}_3$ and $\mathscr{S}_4$, and equivalent to each other. To select between $\mathscr{S}_3$ and $\mathscr{S}_4$ we prefer shorter concepts, since $|\mathscr{S}_3| < |\mathscr{S}_4|$, the most specific representative $MSR_\mathcal{X}$ is $\mathscr{S}_3$. Given yet another concept :

$$\mathscr{S}_5 \equiv \exists r.(\{y\} \sqcap A \sqcap B)$$

We have that $\mathscr{S}_5 \equiv \mathscr{S}_3$ and $|\mathscr{S}_5| = |\mathscr{S}_3|$, thus they are two versions of the same concept.

3.3.2.1 Refinements of One Individual

If we apply α to each refinement iteratively we can traverse the space of refinements of C, as given in Algorithm 3. From Propositions 3.7 and 3.9 we have that each refinement obtained by α is the most specific concept of an equivalence class w.r.t. x, therefore using α we traverse the space of equivalence classes of x. The space of possible refinements is a lattice (Fig. 3.6), where each node's descendants are as well sub-sets of their predecessors, since the refinements go from more general to more specific. In this way we can find the MSR for $\{x\}$. Note that, even though applying α to two different refinements C' and C'' will not output the same sets, it suffices to choose only one refinement in each iteration to arrive to the MSR for $\{x\}$. This is because the MSR has to consider all possible properties of x, and by continuously applying α to any refinement C' we will leave no necessary properties unexplored. By selecting any of the output refinements of $\alpha(C)$ as the input of the next iteration, we will arrive to the MSR when the output of α is empty.

Without loss of generality, we consider that each role in the A-Box is unique. Then each refinement C' obtained by the operator will make a unnecessary assertion in the A-Box for C, necessary for C'. Since the A-Box is finite, iterating the

Algorithm 3 Computing the MSR for an instance x of C

1: **input:** $(x, C, \mathcal{O})$
2: $C_i = C$
3: **while** $\alpha_x(C_i) \neq \emptyset$ **do**
4: Set C_i to any $C' \in \alpha_x(C_i)$
5: **end while**
6: **return:** C_i

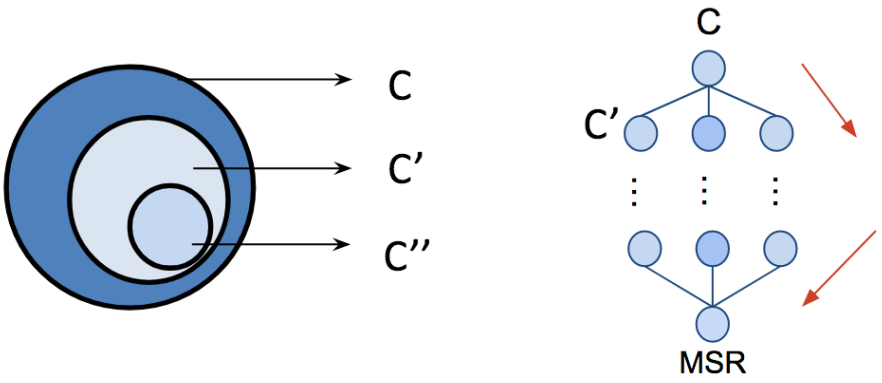


Figure 3.6: The lattice of refinements of C w.r.t. x.

operator with Algorithm 4 will become all unnecessary assertions will necessary. At this point the set of extensions $Ext_{x,C}$ will become empty, and the iteration process will stop.

In Algorithm 4, regardless of the refinement chosen to continue the iteration process (Line 4) all extensions in $Ext_{x,C}$ will be explored by α. And thus if the iteration process is allowed to finish, a version of the MSR will be obtained in the end.

3.3.2.2 Refinements of a Set of Individuals

From Proposition 3.9 it follows that using α_x we can the most specific representative for a situation to which a single instance $x \in \mathcal{X}$ belongs. But this does not necessarily hold for a set of instances $\mathcal{X}$. The instances in $\mathcal{X}$ might share a limited number of properties, thus the MSR of x does not necessarily represent $\mathcal{X}$. There might exist several refinements found by α_x that represent $\mathcal{X}$, but from which none is the MSR of $\mathcal{X}$. This is because the refinements found by α, depend on the instance chosen. Using an instance $x_2 \in \mathcal{X}$, with $x \neq x_2$

might give different results. This implies that even if we find all the most specific refinements that represent $\mathcal{X}$ using x, none of them might be the most specific with respect to the whole set. Therefore, *a priori*, we would have to explore all of the refinements of all the instances in $\mathcal{X}$.

To find the refinements that best represent $\mathcal{X}$ we assign a rank to them, based on their recall of $\mathcal{X}$.

The first observation here is that, since for each refinement C' of C it holds $C' \subseteq C$, whenever C' no longer represents $\mathcal{X}$ (i.e. some individual in $\mathcal{X}$ is not an instance of C') it implies no descendant of C' can represent $\mathcal{X}$, and therefore we do not require to further refine C'. The second observation is that, from all the refinements of α_x with 100% recall, we can not, *a priori*, discard any of them as candidates for the iteration process. Consider the following case:

Example 3.10. *The MSR of a set of individuals* $\mathcal{X}$

$$\mathcal{X} = \{x, x'\}$$

$$\mathcal{A}_6 : \{r_1(x,y), r_2(y,w), r_3(y,z)\} \cup$$
$$\{r_1(x',y'), r_1(x',z'), r_2(y',w'), r_3(z',u')\}$$

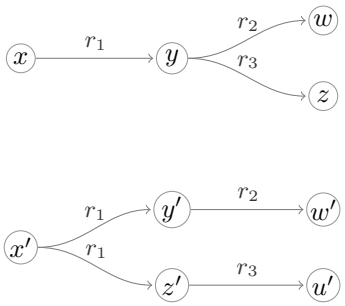


Figure 3.7: The graph representation of individuals x and x'.

If we refine a concept $\mathscr{S} = \exists r_1.\top$ *guided by* x*. Applying* $\alpha_x(\mathscr{S})$ *we find two refinements* $\mathscr{S}_1$ *and* $\mathscr{S}_2$ *with a recall for* $\mathcal{X}$ *of 100%, where:*

$$\mathscr{S}_1 = \exists r_1.(\exists r_2.\top)$$

$$\mathscr{S}_2 = \exists r_1.(\exists r_3.\top)$$

The iteration process will stop here, since any further refinement will require the individuals w *or* z *(not connected to* x'*) or be of the form:*

$$\mathscr{S}_3 = \exists r_1.(\exists r_2.\top \sqcap \exists r_3.\top)$$

The recall of $\mathscr{S}_3$ is 50%, since $x' \notin \mathscr{S}_3$ (i.e. $\mathcal{A}_6 \not\models \mathscr{S}_3(x')$). Guided by x we can not find better representatives than $\mathscr{S}_1$ and $\mathscr{S}_2$ for $\mathcal{X}$, nevertheless there exists a concept that represents $\mathcal{X}$ which is more specific than both :

$$\mathscr{S}_4 = \exists r_1.\exists r_2.\top \sqcap \exists r_1.\exists r_3.\top$$

This concept can not be found by α using x. The operator α is designed to provide the MSR for the given minimal A-Box. We can see that $A_{x,\mathscr{S}_3} = A_{x,\mathscr{S}_4}$, and since $\mathscr{S}_3 \sqsubseteq \mathscr{S}_4$, α_x will prefer $\mathscr{S}_3$. But we find that this is not true with respect to x'. In other words, to find $\mathscr{S}_4$ we need to use x'.

Definition 3.27. *Let $\mathcal{X}$ be a set of instances and $\mathcal{O}$ be an ontology. Given a concept C, the recall of C is defined as $recall(C) = \frac{|\{x \in \mathcal{X} \mid \mathcal{O} \models C(x)\}|}{|\mathcal{X}|}$.*

At a first sight, this would mean that we have either to test all individuals in $\mathcal{X}$ or somehow select the correct individual from the beginning to find the concept we are looking for. Two observations can help us to avoid this exhaustive search in a more clever way: first, the MSR of a set $\mathcal{X}$, if it exists, is necessarily a refinement of α for some individual $x \in \mathcal{X}$ and second, given a refinement $\mathscr{S}'$ that represents $\mathcal{X}$ obtained by applying α to $\mathscr{S}$ using an arbitrary x, the MSR of the set $\mathcal{X}$ is necessarily a refinement of $\alpha_{x'}(\mathscr{S}')$ for some x'. This implies we can select any individual $x \in \mathcal{X}$ to refine the input concept $\mathscr{S}$, and as soon as we do not find any more refinements with a 100% recall guided by the current individual x, we can continue the process by selecting a different individual x'. If this process is continued, we will arrive to the MSR for $\mathcal{X}$. This is the intuition behind Algorithm 4 presented next.

Algorithm 4 Get-MSR$_{\mathcal{X}}$

```
1:  input: (C, O, X)
2:  S ← C
3:  for x ∈ X do
4:      Best={S}
5:      while Best ≠ ∅ do
6:          S ← any C' ∈ Best
7:          Best = {C' ∈ α_x(S) | recall(C') = 100%}
8:      end while
9:  end for
10: return: S
```

In Algorithm 4 we obtain all those refinements of $\mathscr{S}$ with a recall of 100% and hold them in the set Best (4,7). Initially, the set Best contains only the input concept C (2,4). As long as this set is not empty (5), we select any of

its elements (which are ensured to be concepts with a 100% recall) as the next concept $\mathscr{S}$ to be refined (6). In (7) we use the operator α guided by the current individual x to obtain all refinements of $\mathscr{S}$ with a 100% recall, and store them in the set Best. If we can still find such kind of refinements, this process is repeated (5). Once we do not find any best refinement, the set Best is empty and the *while* loop in (5) stops. At his point $\mathscr{S}$ is the last best refinement found. Then, the *for* loop in (3) will select another element in the input set $\mathcal{X}$ and repeat the process, until all individuals in $\mathcal{X}$ are explored. Finally, $\mathscr{S}$ holds the 100% refinement for the set $\mathcal{X}$ verified by all instances, which is returned as the output (10).

Continuing with the previous example, from all the refinements of C we find two types: those for which the recall of the set $\mathcal{X}$ is 100%, and those for which the recall is below. Once no more 100% concepts are found, algorithm 4 stops, and a set of all of those concepts that are immediate refinements of a 100% concept is available (i.e. $\alpha_x(\mathscr{S})$). Each concept in this set, represents a (possibly overlapped) subset of individuals of $\mathcal{X}$ that share some common features.

Proposition 3.10. *Given a set of individuals $\mathcal{X}$, whenever the output of Algorithm 4 $\mathscr{S} \neq \emptyset$, $\mathscr{S}$ is the MSR for $\mathcal{X}$.*

Proof. From Line 7 in Algorithm 4, the output $\mathscr{S}$ holds:

$$\mathcal{O} \models \mathscr{S}(x) \text{ for all } x \in \mathcal{X}.$$

We have to show that there does not exist $\mathscr{S}' \sqsubseteq \mathscr{S}$, s.t. for all $x \in \mathcal{X}$, $\mathcal{O} \models \mathscr{S}'(x)$. We proceed by contradiction. Assume such $\mathscr{S}'$ exists. Since every $D_i \in \alpha_x$ is the most specific concept w.r.t. $A_x^{D_i}$, and since the output $\mathscr{S}$ was obtained by α_x, it means that $\mathscr{S}$ is as well the msc for some $x' \in \mathcal{X}$. This implies that $|A_{x'}^{\mathscr{S}'}| > |A_{x'}^{\mathscr{S}}|$. Thus there exists an axiom $\beta \in A_{x'}^{\mathscr{S}'}$ with $\beta \notin A_{x'}^{\mathscr{S}}$. This implies β is an extension of x' w.r.t. $\mathscr{S}$. From Proposition 3.9 for every such minimal A-Box $A_{x'}^{D}$ w.r.t. some concept D, the operator $\alpha_{x'}$ produces the $msc_{x'}$. But Line 7 tests each such concept, and thus $msc_{x'}$ w.r.t. $A_{\mathscr{S}}$ does not have a recall 100%. Thus $\mathcal{O} \not\models \mathscr{S}'(x)$ for some $x \in \mathcal{X}$.

□

In fact, using a initial set of "samples", and the operator α we can find all the representable set of individuals and thus obtaining the situations characterizing them.

Proposition 3.11. *Given a set of individuals $\mathcal{X}$, an A-Box A and a concept D such that $\forall x \in \mathcal{X}$ we find $A \models D(x)$. Then, D is a refinement of α for some $x \in \mathcal{X}$.*

Proof. $\forall x \in \mathcal{X}, A \models D(x)$ implies $D \in ||\mathcal{X}||$. Thus for each x, there exists a minimal A-Box w.r.t D s.t. $A_x^D \models D(x)$. For each minimal A-Box, α_x will

generate the most specific concept s.t. A_x^D is minimal . Let $D_x \in \alpha_x$ be a refinement obtained guided by x, for which A_x^D is minimal. If $D_x \notin ||\mathcal{X}||$, it implies that for some $x' \in \mathcal{X}$, we find $A_{x'}^{D_x} \not\models D_x(x')$.

The set $\mathcal{X}$ represented by D, implies $D_x \sqsubset D$. Since D is not the most specific concept for A_x^D, but A_x^D is the minimal A-Box for D, there must exist another minimal A-Box of another individual $x' \in \mathcal{X}$ for which D is the most specific concept. Since otherwise D_x would suffice. This implies D is a refinement for some $x' \in \mathcal{X}$. □

Once we can obtain the MSR for a set of individuals $\mathcal{X}$, we know that any refinement of such MSR obtained by the operator α w.r.t. some x will define a strict subset $\mathcal{X}' \subset \mathcal{X}$. Since all these individuals in $\mathcal{X}'$ are instances of the obtained refinement, the set $\mathcal{X}'$ defines a situation. By iterating this process over each subset found, and for each corresponding MSR, we can obtain all the situations in $\mathcal{X}$. Algorithm 5 specifies this process.

Algorithm 5 SD($\mathcal{X}$)

```
1:  input: (C, O, X)
2:  Ξ = {X}
3:  ToRefine = {X}
4:  while ToRefine ≠ ∅ do
5:      for Y ∈ ToRefine do
6:          for y ∈ Y do
7:              for D ∈ α_y(Get-MSR(Y)) do
8:                  Inst_D = {y ∈ Y | O ⊨ D(y)}
9:                  Add Inst_D to ToRefine
10:             end for
11:         end for
12:         remove Y from ToRefine
13:     end for
14:     Add ToRefine to Ξ
15: end while
16: return: Ξ
```

In Algorithm 5 the set of all situations Ξ is initialized with $\mathcal{X}$ (Line 2), since C representing $\mathcal{X}$ implies $\mathcal{X} \in \Xi$. Next we define a set $ToRefine$ of all those sets that need to be analyzed to look for sub-situations (Line 3). For each such set Y (Line 5) we obtain its MSR computed by Algorithm 4, and for each individual in Y (Line 6), we refine $MSR(Y)$. In this fashion, we obtain the representable subsets of Y. Intuitively, if there exists a subset of Y that can be represented, there exists a concept $D \sqsubset MSR$. This concept can be found

by applying $\alpha_y(MSR)$ for some $y \in Y$. For every such refinement found, we record the sets its instances in $Inst_D$ (Line 8). Because there exists a concept D for each one of these sets, they define a situation. And thus all the different sub-sets are added to $ToRefine$ (Lien 9), to explore if sub-situations can be found. Since all subsets found this way are representable, we add all of them to Ξ. This process is repeated for the MSR every subs set found, and refined with every instance in such sets. Finally, when no more subsets can be found. The refinements of every MSR will be empty (there exists no concepts that are more specific than those found for the discovered sets) and thus the while loop (Line 5) will stop. The output of Algorithm 5 is the set of all situations in $\mathcal{X}$.

Proposition 3.12. *Given an ontology $\mathcal{O}$, a concept C and the set of its instances $\mathcal{X}$, all elements in the output Ξ of Algorithm 5 are situations in $\mathcal{X}$.*

Proof. We have to prove that every subset $\mathcal{X}' \subset \mathcal{X}$ that belongs to Ξ can be represented. If there exists a subset $\mathcal{X}' \subset \mathcal{X}$ s.t. $\mathcal{X}'$ is representable, then there exists its most specific representative $MSR_{\mathcal{X}'}$. From $\mathcal{X}' \subset \mathcal{X}$ we have $MSR_{\mathcal{X}'} \sqsubset MSR_{\mathcal{X}}$.

From Proposition 3.9 we have that the MSR of a set Y, is a refinement obtained by α_y for some $y \in Y$. That is if Y is representable, its MSR can be found by α. In Algorithm 5 for all $x \in \mathcal{X}'$ (Line 6) we obtain all the immediate refinements $D \sqsubseteq MRS_{\mathcal{X}}$ (Line 7). Thus $MSR_{\mathcal{X}'} \subseteq D$ for some D. In (8) we obtain all those individuals that are represented by D, and the iteration process obtains its MSR. Thus every $\mathcal{X}' \subset \mathcal{X}$ added to Ξ is representable. □

3.4 Summary

We have presented an approach to discover interesting subsets of individuals in an ontology $\mathcal{O}$ called situations. Each situation in $\mathcal{O}$ defines a set of individuals that can be described by a DL concept, and that might have an infinite number of concepts associated. To uniquely characterize each situation, we adapted the notions of justifications in ontologies and the notion of most specific concepts. We have presented an algorithm to compute a minimal A-Box, based on the notions of necessity. The minimal A-Box allows us to construct refinements of a given input concept, with respect to one of its instances. To this end we have presented a sound and complete algorithm, in the form of a refinement operator, to obtain the immediate refinements of a given concept. The operator allows to characterize all the situations in $\mathcal{O}$ to which the given input instance belongs. This process is then extended from a single instance, to obtain the MSR of a set of instances. Finally we have presented how to characterize all the situations present in the input set, where for each situation a MSR can be obtained, thus

solving the situation discovery problem defined at the beginning of this chapter, in Section 3.2. The algorithms and processes detailed in this chapter allow for concept learning in an unsupervised fashion. We do not use positive nor negative samples, and provide the most general and complete version of our operator to traverse a search space, characterized by the notion of situation in $\mathcal{O}$. In this respect, the specifications in this chapter are highly customizable and can be adapted to solve a problem where automatic descriptions for sets of instances need to be discovered. The selection of which of these descriptions are to be kept, is case dependent, and in this regard the approach provides the shortest and most specific concepts, that provide access to situations in $\mathcal{O}$.

In Chapter 4 the relation between situations and failure signatures is established, showing how the general approach presented in this chapter can be adapted for Signature Analysis (SA).

Chapter 4

Situation Discovery in Avionics Maintenance

In this chapter we detail how the refinement process presented in chapter 3 can be used in the avionics maintenance domain. There are two main functions the prototype should provide to support the technician on the diagnosis process : 1) Consult the knowledge base to obtain suggested corrective actions, and 2) Enrich the knowledge base through the users feedback on whether or not the given suggestions were relevant. The enrichment of the knowledge base requires not only to consider the possibly new corrective actions, but also to discover new failure signatures. This chapter specifies the results these processes should yield, and how these results can be obtained. To this end, we first present the constructed ontology for avionics maintenance on which the approach is based. The ontology has been designed considering the two main data sources: the .AR files and the corrective actions, detailed in the next section. Next we explain how the signature of a failure can be defined as a *situation* in the ontology, enabling the situation discovery process presented in chapter 3 to be used to solve the problem of discovering failure signatures. Finally, we present how to integrate the technician's feedback into the knowledge base, before turning to the summary.

4.1 The Data and The Ontology

In this section we present the data that was provided for the approach and the ontology constructed to capture its most relevant features. The ontology is to represent the information of the diagnosis process, in terms of *concept names* and *role names*, which are combined using the DL language constructors (conjunction, existential restriction, nominals) to create concept definitions.

These definitions conform the terminological knowledge or T-Box of the ontology. Once these concept definitions are available, we can populate the ontology by adding assertions to the A-Box, which denote facts expressed according to the terminology in the T-Box.

The information about the diagnosis process has two main sources: the .AR files containing the results of the tests made to each equipment, and the corrective actions associated to each such test. This information has been modeled in the ontology Thales Avionics Maintenance Ontology (TAMO), aided by the Thales Avionics experts.

We start by detailing the structure of the .AR files and the ontology to describe them.

4.1.1 The All Results (.AR) files

In the diagnosis process the technician tests the ELAC in a special unit called a Test-Bench. This unit checks exhaustively all the ELAC functions, and the output of this process is an .AR file (All Results). The .AR files are the main source of information for the ontology. They are presented in plain text format and contain up to thousands of lines. Each line of an .AR file represents an individual test on a specific function of the ELAC, with the sanction GO or NOGO which indicates if the test was passed. Thus, an .AR file is a set of individual test results (thus the name All Results files). Each .AR file is divided in three main sections:

1. The header, with information about the context of the test, including: the equipment to be tested, the time and date, the type of test.

2. The test results, which are organized in three levels: *part*, *function* and *sub-function*. This section contains the individual tests on the functions of the ELAC.

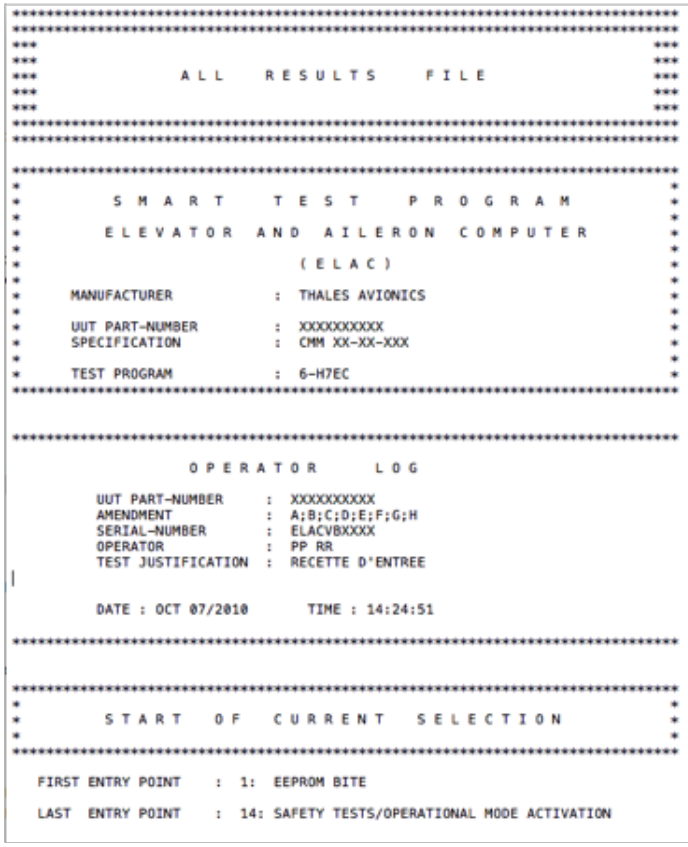
3. The footer, summarizing the results and length of the full test.

Figure 4.1 shows an extract of the header of an .AR file. On the very top we find the title of the file "ALL RESULTS FILE". Next, we have the "SMART TEST PROGRAM" section, where the *manufacturer*, the *part-number* of the tested equipment, the corresponding maintenance manual with the *specification* of the equipment, and the specific *test program* run for this examination are shown.

The "OPERATOR LOG" section, is concerned with the parameters given by the operator to run the test: characteristics of the tested equipment (*amendment*), the *operator* (technician) who made the test, the *justification* of the test (whether

it is a initial test, a loop test or a final test), and the *date-time* when the test was started.

Each .AR file can test a specific set of functions from the ELAC. The section "START OF CURRENT SELECTION" in the header, provides the first and last entry point of the test, i.e. which functions are tested. The functions tested by the test-bench are organized in *parts*. The parts to be tested, depend on the equipment and the program selected. It is for the operator to decide if he/she wants to test all parts, just a sub-set of them, and how many times. In the specific case of the ELAC, there are fourteen parts that can be tested. In the *initial test* and *final tests*, it is mandatory that all fourteen parts are checked. After an initial examination, it is possible that a test of all the parts does not detect any failure. If nevertheless the technician suspects a failure is still present in a specific function, he/she can choose to test one specific part, several times to stress the equipment and find the failure. This is called a *loop test*.



```
**************************************************************************
**************************************************************************
***                                                                    ***
***                                                                    ***
***              A L L    R E S U L T S     F I L E                    ***
***                                                                    ***
***                                                                    ***
**************************************************************************
**************************************************************************

**************************************************************************
*                                                                        *
*          S M A R T     T E S T     P R O G R A M                     *
*                                                                        *
*        E L E V A T O R   A N D   A I L E R O N   C O M P U T E R       *
*                                                                        *
*                              ( E L A C )                               *
*                                                                        *
*      MANUFACTURER            :  THALES AVIONICS                        *
*                                                                        *
*      UUT PART-NUMBER         :  XXXXXXXXXX                             *
*      SPECIFICATION           :  CMM XX-XX-XXX                          *
*                                                                        *
*      TEST PROGRAM            :  6-H7EC                                 *
**************************************************************************


**************************************************************************
                   O P E R A T O R      L O G

          UUT PART-NUMBER     :  XXXXXXXXXX
          AMENDMENT           :  A;B;C;D;E;F;G;H
          SERIAL-NUMBER       :  ELACVBXXXX
          OPERATOR            :  PP RR
          TEST JUSTIFICATION  :  RECETTE D'ENTREE

          DATE : OCT 07/2010       TIME : 14:24:51
**************************************************************************


**************************************************************************
*                                                                        *
*          S T A R T    O F    C U R R E N T    S E L E C T I O N       *
*                                                                        *
**************************************************************************

   FIRST ENTRY POINT    :  1:  EEPROM BITE

   LAST  ENTRY POINT    :  14: SAFETY TESTS/OPERATIONAL MODE ACTIVATION
```

Field	Description
Section: S M A R T T E S T P R O G R A M	
MANUFACTURER	The manufacturer of the tested equipment.
UUT PART-NUMBER	Any equipment is uniquely identified by the Part-Number and the Serial Number. The part number represents a family of equipments (e.g. ELAC) and the serial number represent the specific equipment.
SPECIFICATION	The corresponding manual where the meaning of each test run can be found.
TEST PROGRAM	The specific piece of software used to run the test.
Section: O P E R A T O R L O G	
UUT PART-NUMBER	Idem as above.
AMENDMENT	If the equipment (ELAC) had some amendments, they are listed here. The amendments can be seen as upgrades/updates to the equipment.
SERIAL-NUMBER	See UUT PART-NUMBER.
OPERATOR	The technician that run the test.
TEST JUSTIFICATION	The type of test run (Initial test, Loop test or Final test).
DATE	The date and time of the test.
Section : S T A R T O F C U R R E N T S E L E C T I O N	
FIRST ENTRY POINT	Fourteen sections (parts) can be tested for each equipment. This value indicates the first part tested. A initial test must run all parts (1-14), loop tests can run any subset of them.
LAST ENTRY POINT	Last section (part) tested.

Figure 4.1: To the left, an extract of the header of an .AR file, to the right the description for each field.

Once the context of the full test is set in the header, the second section of the .AR file (figure 4.2) shows the results of the individual tests, hierarchically organized in parts, functions, and sub-functions. The largest sub-section of the results is the *part*. An ELAC .AR file can have between one and fourteen parts. In figure 4.2 it is shown *part* 8: ARINC INPUTS/OUTPUTS, where two tests are made at the level of the *function* ARINC MESSAGE RECEPTION and three tests made at the level of the *sub-function* COM ARINC reception CHANNEL1). Each individual test, is presented in a single line with the following format: first a *code* (eg. 180203) and a *designation* (eg. T1DGIO08 080030), which allows to identify the corresponding entry in the CMM to interpret the test result. The *designation* has two levels, and is divided in *designation 1* (e.g. T1DGIO08) and *designation 2* (e.g. 080030). Next, we have the *measurements* (e.g. 40AC HEX EQ 40AC) obtained by the test-bench on the specific function. The type and format of the measurements depend on the specific functions being checked, and are usually ranges of voltage values. If the measurement of the voltage value is within the expected range, the test is passed, otherwise is failed. In figure 4.2 the measurement 40AC HEX EQ 40AC on line with code 180203, indicates that the obtained value is 40AC (left) and it should be equal to 40AC (right). Since they are equal, the *result* of the test is GO. In the next line (code 180206), the expected value is also 40AC but the read value is 50AC, and thus the test is not passed, and the result is NOGO.

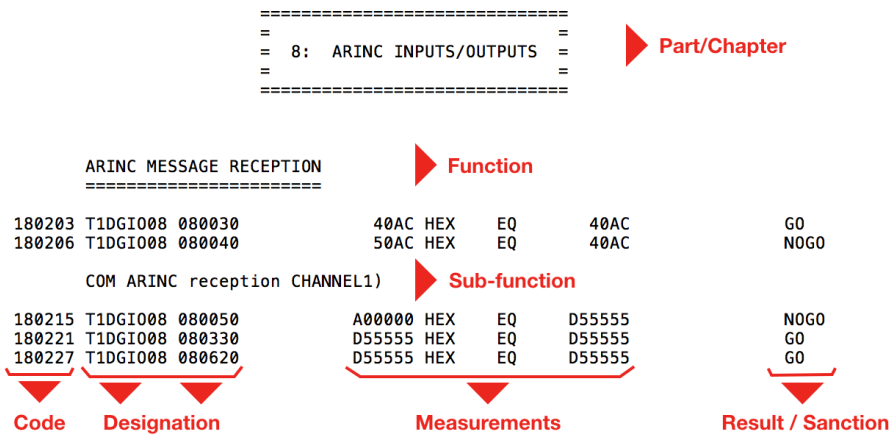
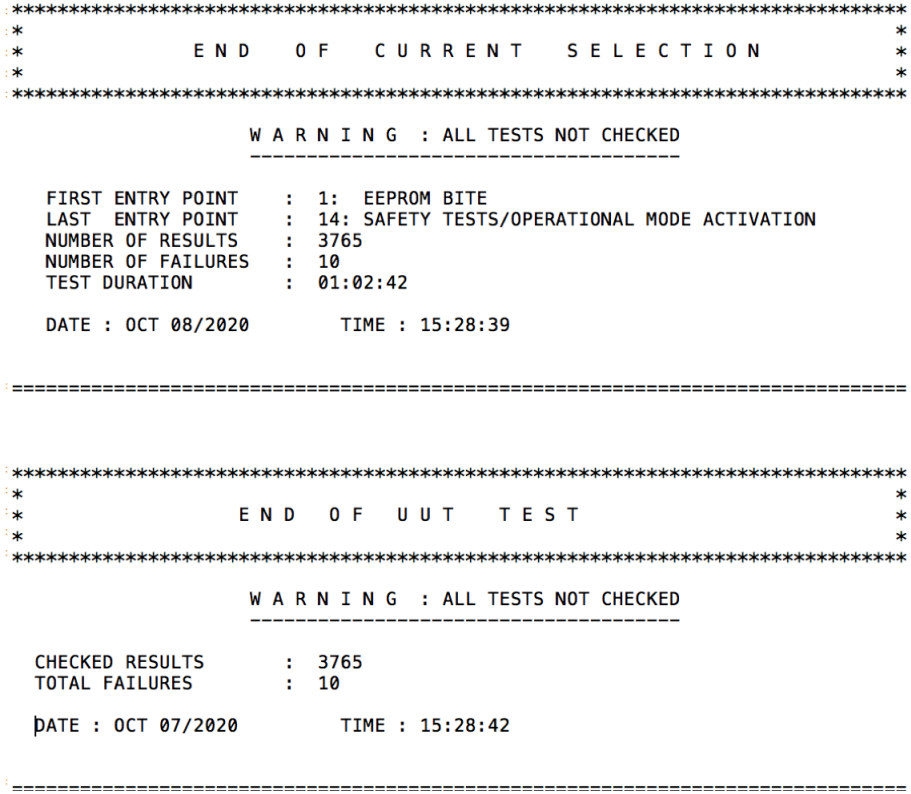


Figure 4.2: An extraction of the structure of an .AR file.

Finally, the footer section (figure 4.3) presents the summary of the results. It is divided in two sub-sections: END OF CURRENT SELECTION and END OF UUT TEST. In case a *loop test* is run, the technician can instruct the test-bench

to test a sub-set of parts any given number of times. This is called a *selection*. Once the *selected parts* have been tested, the section END OF CURRENT SELECTION summarizes its results: the first and last tested *parts*, the number of individual tests made, the number of failed tests and the date/time when the test was finished.

The second section in the footer of the .AR file is "END OF UUT TEST" and it displays the aggregated results of each selection of parts. This section shows, the total number of individual tests made, the total number of failures, and the date/time of the conclusion of the whole test.



```
*******************************************************************************
*                                                                             *
*              E N D    O F    C U R R E N T    S E L E C T I O N            *
*                                                                             *
*******************************************************************************

                    W A R N I N G  : ALL TESTS NOT CHECKED
                    --------------------------------------

   FIRST ENTRY POINT    :  1:  EEPROM BITE
   LAST  ENTRY POINT    :  14: SAFETY TESTS/OPERATIONAL MODE ACTIVATION
   NUMBER OF RESULTS    :  3765
   NUMBER OF FAILURES   :  10
   TEST DURATION        :  01:02:42

   DATE : OCT 08/2020        TIME : 15:28:39


===============================================================================



*******************************************************************************
*                                                                             *
*                  E N D   O F   U U T   T E S T                              *
*                                                                             *
*******************************************************************************

                    W A R N I N G  : ALL TESTS NOT CHECKED
                    --------------------------------------

  CHECKED RESULTS       :  3765
  TOTAL FAILURES        :  10

  DATE : OCT 07/2020         TIME : 15:28:42


===============================================================================
```

Field	Description
Section: END OF CURRENT SELECTION	
FIRST ENTRY POINT	The first *part* tested.
LAST ENTRY POINT	The last *part* tested.
NUMBER OF RESULTS	The number of individual tests performed.
NUMBER OF FAILURES	The number of tests that were not passed.
TEST DURATION	The duration of the current selection
DATE/TIME	The date and time of the end of the current *selection*.
Section: END OF UUT TEST	
CHECKED RESULTS	The aggregation of all individual tests made, in all *selections*.
TOTAL FAILURES	The aggregation of tests failed, in all *selections*.
DATE/TIME	The date and time of the end of the whole test.

Figure 4.3: To the left, an extract of the footer of an .AR file, to the right the description for each field.

4.1.2 Modelling AR Files in the Ontology TAMO

In this section, we present the part of Thales Avionics Maintenance Ontology (TAMO) coming from the analysis of the .AR files.

The concepts and role definitions of TAMO are represented in Description Logics in form of a T-Box. Once the T-Box is built, the A-Box of TAMO can be populated through instantiation of terms of the T-Box. The T-Box along with the A-Box constitute the *knowledge base KB*.

Figure 4.4 and 4.5 detail the main concept names and role names, respectively, to represent the .AR tests in the ontology.

TAMO CONCEPTS

Concept		**Description**
Test		*Groups all types of tests*
	TestLogLine	*Each line present in a TestMain*
	TestMain	*Each .AR file is a TestMain*
TestPart		*The first level in the structure of a test*
TestFunction		*The second level in the structure of a test*
TestSubfunction		*The third level in the structure of a test*
TestCode		*The code of the TestLogLine*
TestDescription		*A description of a test*
TestDesignation		*Each TestLogLine can have one or two designations*
TestFile		*The .AR file name*
TestProgram		*The program run on the test-bench for this specific TestMain*
TestResult		*The sanction of each TestLogLine*
Justification		*The reason the test was made (intitial test, final test, loop)*
Station		*The Test-Bench where the test was performed*

Figure 4.4: The concepts chosen to represent the tests in TAMO.

As seen before, there are several levels of test in an .AR file. The .AR file itself is a test, denoted by *TestMain* and each individual line in an .AR file is as well a test, denoted by *TestLogLine*. Each TestLogLine can belong to a *TestPart*, a *TestFunction* and/or a *TestSubFunction* within the .AR file. Besides the location of each TestLogLine within the file, we consider its most significant properties: its *TestCode*, the two *TestDesignations*, and the *TestResult*. We do not consider the *measurements* of the TestLogLine, because they refer to numeric real values which would require additional processing (determine if they are in a range, if they are greater or smaller than, etc.) which would require to increase the expressiveness of the language chosen for the ontology. Thus we consider

only if the test was passed or not.

Our approach focuses on the tests made on the ELAC computer. The ontology (Figure 4.10) is able to represent more information than the one considered relevant for the current use case, but we have selected a subset of concepts and roles that are relevant for our scenario. Because of this, there is no need to consider the *manufacturer*, the *part number*, nor the *serial number* of the equipment. We have also omitted the *first entry point* and *last entry point* fields, since the TestPart is already associated to each TestLogLine.

Once the concepts are defined, we need to establish the relations between them. We do this by introducing roles. Intuitively, a role defines a binary relation between two individuals in the form of $R(a, b)$. The domain of the role R establishes the type of individuals allowed as the first component (a) and the range defines the type of individuals allowed as the second component (b). Figure 4.5 presents the role domain, the role name, the role range, and a description for the main roles selected in TAMO.

TAMO ROLES

Domain	Role	Range	Description
TestMain	**hasTestFile**	TestFile	*The .AR file name associated to each TestMain*
TestMain	**hasTestProgram**	TestProgram	*The program run but the test-bench*
TestMain	**hasTestLine**	TestLogLine	*Links a TestLogLine to a specific TestMain*
TestMain	**justifiedBy**	Justification	*The justification of the full test (initial, loop, final)*
TestLogLine	**hasTestCode**	TestCode	*The TestCode of each TestLogLine*
TestLogLine	**hasTestDes1**	TestDesignation	*The first designation of each TestLogLine*
TestLogLine	**hasTestDes2**	TestDesignation	*The second designation of each TestLogLine*
TestLogLine	**hasTestResult**	TestResult	*The sanction of each TestLogLine (GO/NOGO)*
TestLogLine	**lineHasPart**	TestPart	*The TestPart to which this TestLogLine belongs*
TestLogLine	**lineHasFunction**	TestFunction	*The TestFunction to which this TestLogLine belongs*
TestLogLine	**lineHasSubFunction**	TestSubfunction	*The TestSubfunction to which this TestLogLine belongs*

Figure 4.5: The roles chosen to represent the tests in TAMO.

The concept names and role names introduced so far constitute the T-Box of TAMO, that is the terminological knowledge represented in the ontology. The T-Box alone contains no information about any particular .AR test. It defines the vocabulary we can use to describe them, and what form these descriptions should have, but it does not contain the descriptions themselves. To add the .AR tests to the ontology (i.e. populate the ontology) we assert facts about the .AR files in the form of axioms in the A-Box. The A-Box constitutes all the facts we know about our domain.

Since the A-Box contains only unary and binary relations, it can be represented as a graph, which can be a useful representation for visualization and processing. Consider the following A-Box corresponding to some of the lines from the .AR file in figure 4.6.

A-Box$_1$ = {
TestMain(t_1), TestFile($ELAC.AR$),TestLogLine(L_1),
TestDesignation($T1DGIO08$),TestDesignation(080050),TestResult($NOGO$),
TestPart(8: $ARNIC$...), TestFunction($ARNIC\ MESSAGE$...),
TestSubfunction($COM\ ARNIC$...),TestPart(TS), TestCode(180215),
hasTestFile($t_1, ELAC.AR$),hasTestLine(t_1, L_1),
lineHasSubfunction($l_1, COM\ ARNIC$...), hasTestResult($l_1, NOGO$),
lineHasPart(l_1,8: $ARNIC$...),lineHasFunction($l_1, ARNIC\ MESSAGE$...),
hasTestDes1($l_1, T1DGIO08$),hasTestDes2(l_1, 080050),hasTestCode(l_1, 180215)}

Then A-Box$_1$ can be represented as a graph, as shown in figure 4.7.

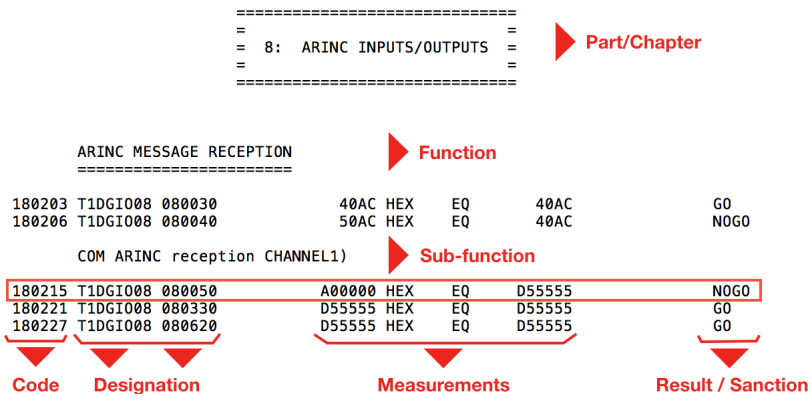


Figure 4.6: The extract of the .AR file corresponding to A-Box$_1$.

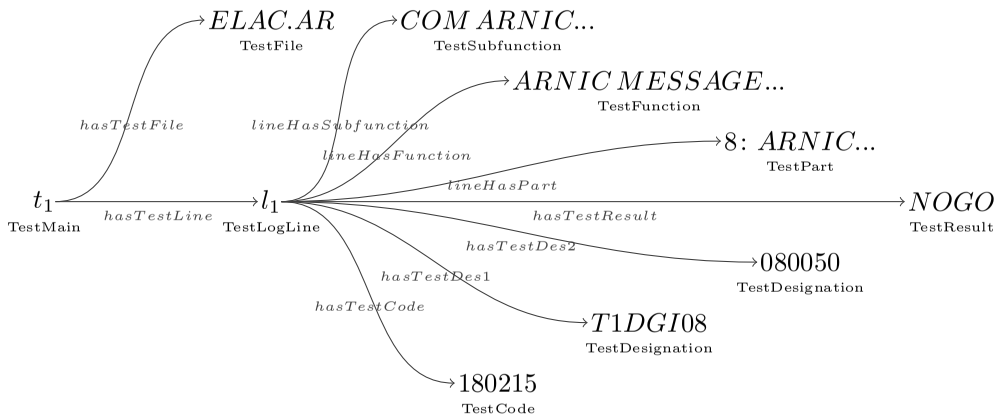


Figure 4.7: The graph representation of a line from A-Box$_1$.

Figure 4.7 shows an example of the extract of an .AR file represented as a graph. The graph represented corresponds to A-Box$_1$. The nodes of the graph are the individuals, the labels below the nodes correspond to concepts names, and the edges between the nodes are the relations between the individuals. The example in figure 4.7 shows a partial representation of the .AR file, where the $TestMain$ t_1 has a single $TestLogLine$ l_1 and seven properties.

4.1.3 The Corrective Actions

On the other hand we have the **corrective actions**. In avionics maintenance there are multiple types of maintenance actions (repair, cleaning, preventive maintenance tasks, upgrades, etc) which we call the *event* of the maintenance action. From these types of maintenance actions, we have selected the *replacements* of the components in the ELAC as the actions to be modeled, since these components have a direct influence in the results of the .AR files. Each ELAC is a computer composed of several *boards* (six plus two interface boards), and each board has hundreds of *components* of different *types* that can be replaced. Figure 4.8 illustrates the structure of an ELAC and a set of corrective actions, as extracted from the workshop systems. The actions are divided per SRU (Shop Replacement Unit), which in our case are each one of the boards in the ELAC. The types of boards are: POWER MON, MPU ANA, MSP DG, CSP DG, CPU ANA, POWER COM, LSP and INTER UNIT.

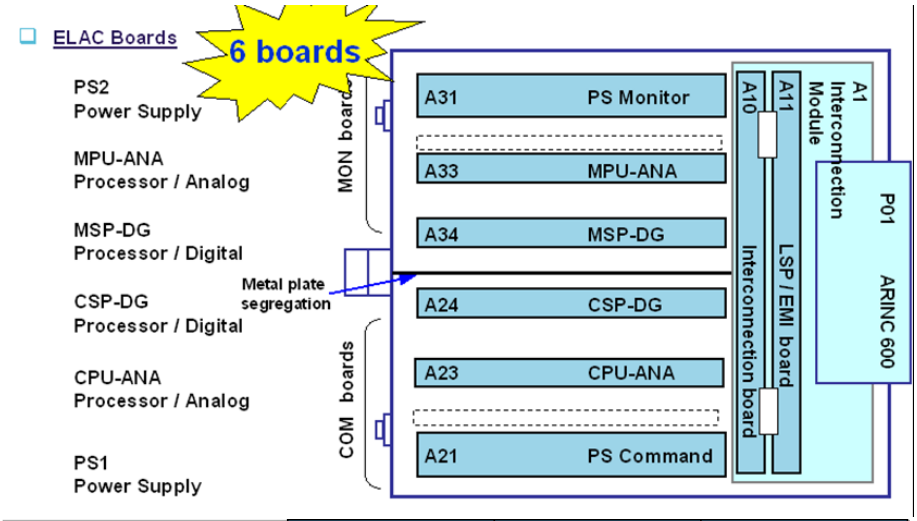
In the sample data in figure 4.8 the corrective action for each file is given by the board (SRU1 Type), the specific position (repere topo) and the type of component (SRU1 component) to be replaced. These three elements are necessary for the technician to identify the replacement. In the example, the corrective action for file 20094-777-777.AR is to replace components U30 and U44 of type AMPLI in the board MPU ANA. Therefore an individual corrective action must denote the board, the position of the component and the type of component to be replaced. Formally:

Definition 4.1 (Individual corrective action)**.** *An* individual corrective action ia *is a tuple:*

$$ia = (board, component, type)$$

where board *is the board in the ELAC computer where the replacement takes place,* component *is the position of the replaced component and* type *is the type of the replaced component.*

Each time a repair is needed several such replacements can take place, in several boards, therefore a complete *corrective action* is a set of individual repairs.



Ar File	SRU1 Type	SRU1 Component	SRU1 repere topo
20094-777-777.AR	MPU ANA	AMPLI	U30 U44
20182-777-777.AR	MSP DG	EPLD	U28
20030-777-777.AR	MSP DG	EPLD|RAM	U25|U35 U36

Figure 4.8: The association between the .AR files and the repair actions. From left to right: the .AR file name, the board, the type of component and the position of the replaced component. Diagram extracted from *ELAC product specification sheet.*

Definition 4.2 (Composed corrective action)**.** *A composed corrective action a is as a set:*

$$a = \{ia_1, ia_2, \ldots, ia_n\}$$

where each ia_i is an individual corrective action.

For example, the file 20030-777-777.AR, has associated a composed corrective action $a = \{ia_1, ia_2, ia_3\}$ where :

$$\begin{aligned} ia_1 &= (\text{MSP DG, U25, EPLD}) \\ ia_2 &= (\text{MSP DG, U35, RAM}) \\ ia_3 &= (\text{MSP DG, U36, RAM}) \end{aligned}$$

4.1.4 Modeling Corrective Actions in TAMO

The model of the actions is simpler than that of the .AR files, since we consider a specific type of corrective action (replacement) and the three elements that identify them: the board, the component position (topo) and the component type.

The T-Box with the concept names and role names to represent the corrective actions is shown in figure 4.9.

TAMO ACTIONS T-Box

Concept			**Description**
ComposedCA			Denotes a composed corrective action
AtomicCA			Denotes an individual corrective action
SRU			The board where the component replaced by each AtomicCA takes place
LocationInSRU			The position (topography) on the board (SRU) of the component replaced by each AtomicCA
ComponentType			The type of the component replaced by each AtomicCA
Domain	**Role**	**Range**	**Description**
ComposedCA	**hasIndividualAction**	AtomicCA	*Associates an individual action (AtomicCA) to a corrective action (ComposedCA)*
AtomicCA	**actionHasCard**	SRU	*Indicates the board (SRU) of each AtomicCA*
AtomicCA	**actionHasComponent**	LocationInSRU	*Indicates the location in the board (SRU), of the component replaced by the AtomicCA*
AtomicCA	**actionHasType**	ComponentType	*Indicates the ComponentType of the component replaced by the AtomicCA*

Figure 4.9: The T-Box to represent the corrective actions: on the top, the concepts names, on the bottom the role names.

Note that a technician might perform several tasks during the investigations of the cause of the failure, and might replace suspicious components, which do not really solve the failure. Thus, if these components were listed in the replacements that solve the failure, some unrelated replacements would be suggested. The data we use for training is filtered, and only the root causes for each failure are taken into account.

The ontology TAMO was designed to provide a generic vocabulary and a basis for domain specific ontologies. TAMO allows to represent information about the tests, the corrective actions, the items associated and contextual information about the maintenance process. Since in our scenario we test a single equipment family (ELAC) and the interest is focused solely on the test results, the information about the type of equipment, the operator that ran the test, the aircraft involved, etc., was not considered a priority [1] and was left as possible extensions for future work. The ontology in Figure 4.10 shows the sub-part of TAMO directly related to the tests and actions previously detailed.

[1] This information might also be subject to restrictions, like the airplane logs, since they belong to the client, i.e. the airline.

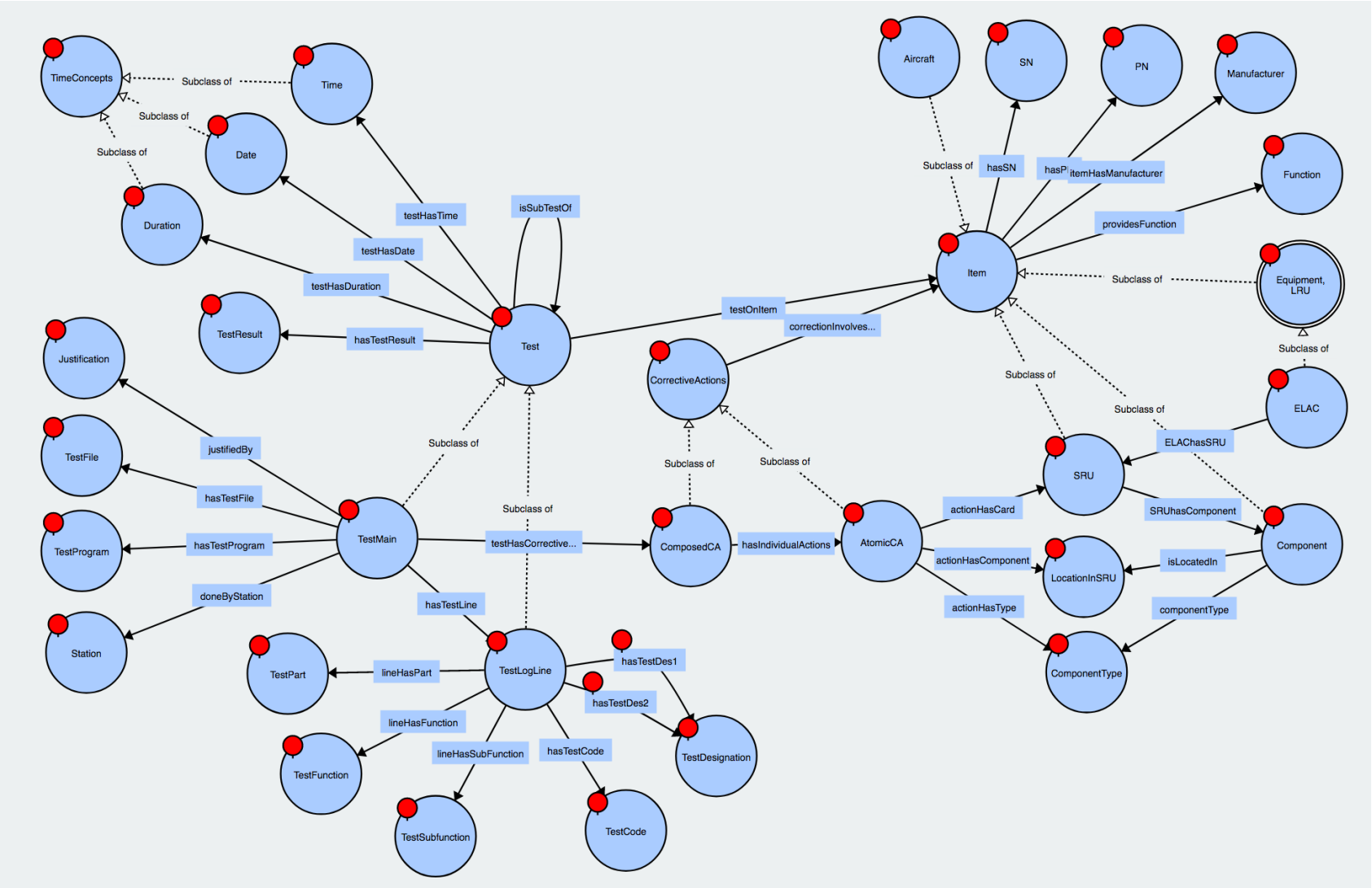


Figure 4.10: The concepts and roles in TAMO selected to represent the tests and the corrective actions. In the figure the concepts and roles directly related to tests and actions are shown.

4.2 Failure Signatures as Situations in $\mathcal{O}$

In avionics maintenance, the signature of a failure is characterized by the properties (results, context) of the test files that detect the failure. Thus what we want to achieve is to extract the relevant properties from the test files, that allow us to define a failure signature. Such a set of properties, enable us to distinguish a set of tests files from the rest, and therefore finding these sets of test, provides the basis to construct a failure signature. Recall that a *situation* in $\mathcal{O}$, is a set of individuals that can be described by a DL concept. The sets of individuals are the sets of tests, and the DL concepts, are the descriptions of the signatures. Under this perspective, a failure signature defines a situation in $\mathcal{O}$, as presented in chapter 3. This section formalizes the relation between failure signatures and situations, and shows how to obtain them.

We start by quoting a definition for the Failure Signature from the Global Standards for the Microelectronics Industry [JEDEC, 2018]:

> The necessary and sufficient information about a failure that establishes a strong relationship between failure characteristics and failure mechanism. This necessary and sufficient information can include emission microscopy results, morphology data, test data, IV-curves, environmental history, etc. and therefore can be either electrical or physical in nature. The scope of application can be time-based, lot-based, package-based, design-based, etc.

Regarding the .AR test files, the information available to define a signature are all the features of the test: the TestLogLines, the TestCode of each line, the TestPart and TestFunction to which each line belongs, etc.

The possible ways in which we can combine the properties of the .AR files grow exponentially with respect to the number of features they present. But only some of the sets of .AR files we find this way, will indeed represent a failure signature. Nevertheless, a failure signature necessarily has to be represented by one or more of these sets of .AR tests. Any such set of .AR tests that *can be* described by a DL concept in $\mathcal{O}$ defines a *situation* in $\mathcal{O}$, thus the situations in $\mathcal{O}$ provide the means to define the failure signatures : every failure signature is a *situation*. Therefore our problem is reduced to finding situations in $\mathcal{O}$.

Nevertheless, not all the situations in $\mathcal{O}$ define a failure signature. Some of them are too general, in that they do not properly separate a set of files from the rest, and can be seen as approximations for the signatures. Thus we look for the more specific description for each situation in $\mathcal{O}$, given by the MSR. The MSR provides the most detailed DL description for a situation in $\mathcal{O}$. If the situation is too general, any refinement of the MSR will lead to the discovery of new, more

specific situations (if they exist). These are the reasons why we search for the MSR: it provides the most precise characterization for each signature, it allows us to define the limit of the current *situation* and enables us to continue the refinement process. Thus every *situation* that represents a set of files $\{f_1, \ldots, f_n\}$ in $\mathcal{O}$ is characterized by its MSR, which we call the signature for $\{f_1, \ldots, f_n\}$. This notion is captured by the following definition:

Definition 4.3 (Signature in $\mathcal{O}$)**.** *Given a set of .AR files $\{f_1, \ldots, f_n\}$ and an ontology $\mathcal{O}$, the* signature $\mathscr{S}$ *of $\{f_1, \ldots, f_n\}$ is their MSR.*

To illustrate this notion consider example 4.1, where on the left of figure 4.11 we have the graph representation of files f_1, f_2, f_3 and on the right the corresponding A-Box.

Given the set of individuals $\Delta = \{f_1, f_2, f_3\}$ the following DL concept definitions are examples of refinements we can obtain using α:

$$
\begin{aligned}
\mathscr{S}_0 &\equiv \exists hasTestLine.(\exists hasTestResult.\{NOGO\}) \\
\mathscr{S}_1 &\equiv \exists hasTestLine.(\exists hasTestCode.\{1234\}) \\
\mathscr{S}_2 &\equiv \exists hasTestLine.(\exists hasTestCode.\{1234\} \sqcap \exists hasTestResult.\{NOGO\}) \\
\mathscr{S}_3 &\equiv \exists hasTestLine.(\exists hasTestCode.\{1234\} \sqcap \exists hasTestResult.\{NOGO\} \\
&\quad \sqcap \exists hasTestPart.\{Part1\})
\end{aligned}
$$

Each of the above refinements is obtained by $\alpha(\top)$ guided by f_1. The concepts are selected to show that the refinements become each time more specific, that many situations can be found guided by one sample, and that for each situation several different concepts might exist. When several concepts represent a situation, the most specific is selected.

From the above concept definitions, we find that $\mathscr{S}_0 = \{f_1, f_2, f_3\}$ and $\mathscr{S}_1, \mathscr{S}_2, \mathscr{S}_3 = \{f_2, f_3\}$, thus these four concepts define two situations in $\mathcal{O}$ (i.e. $\{f_1, f_2, f_3\}$ and $\{f_2, f_3\}$).The signature for $\{f_1, f_2, f_3\}$ is $\mathscr{S}_0$, but we have three candidates for the signature of $\{f_2, f_3\}$: $\mathscr{S}_3, \mathscr{S}_2, \mathscr{S}_1$. Note that $\mathscr{S}_3 \sqsubseteq \mathscr{S}_2 \sqsubseteq \mathscr{S}_1$. Following definition 4.3, we select $\mathscr{S}_3$ as the signature for $\{f_2, f_3\}$ since it is the more specific.

The *signatures* found are added to the KB, so that they can be further used for consulting. The consulting phase, is detailed in the next section.

Example 4.1.

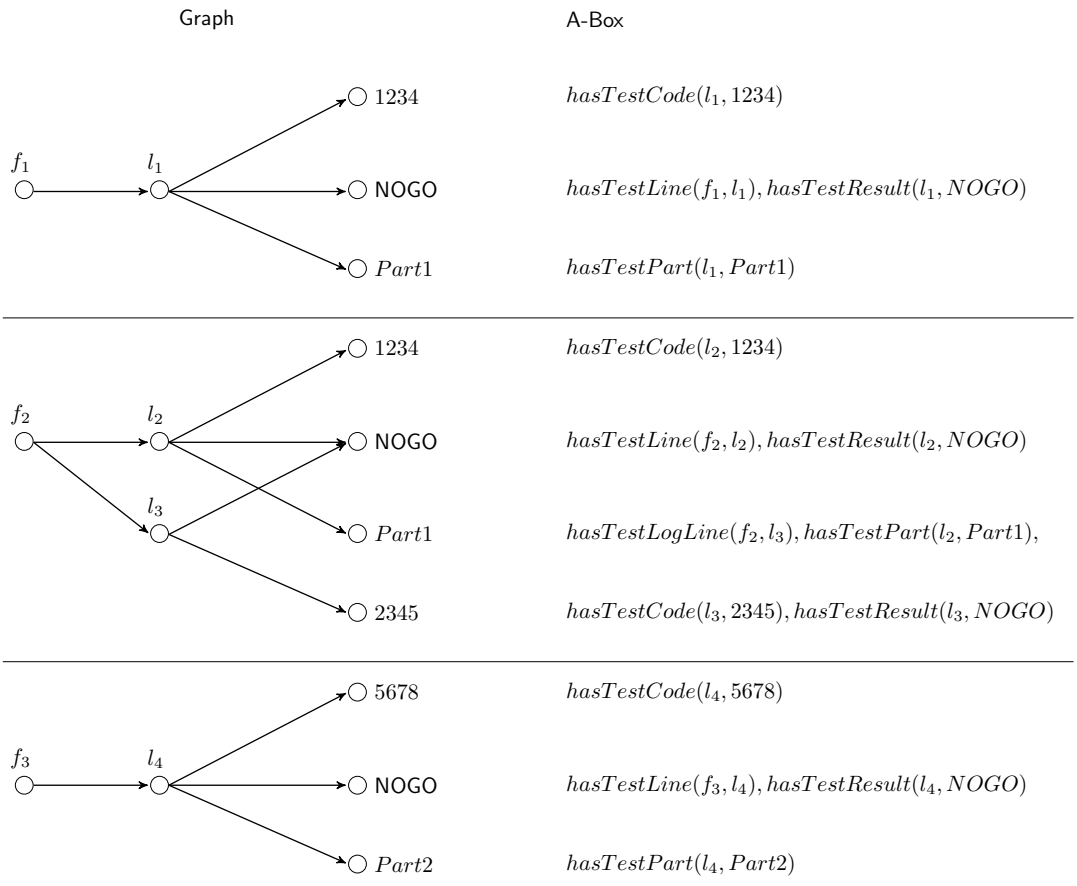


Figure 4.11: Graph representation and corresponding A-Box for files f_1, f_2, f_3.

4.3 Obtain suggestions for a new sample (Consult)

A set of samples of the relation between the tests and the actions is given by an expert. From the technicians experience, we can evidence that for each file, more than one individual corrective action can take place, and in more than one board.

Figure 4.12 shows a subset of the samples of the relation between the .AR files and the corresponding corrective actions:

Ar File	SRU1 Type	SRU1 Component	SRU1 repere topo	SRU2 Type	SRU2 Component	SRU2 repere topo
10094-888.AR	POWER MON	CAPACITOR| AMPLI	C123 C128 | U14 U19 U26 U31	LSP	LOGIC STD| CLEANING	U7
10094-9999.AR	CPU ANA	EEPROM				
10094-1111.AR	CPU ANA	CONVERTER	U75	MPU ANA	AMPLI	U68

Figure 4.12: The association between the .AR files and the repair actions. From left to right: the .AR file, board, type of component, position of component replaced in SRU1, and in SRU 2.

The relation between files and actions is represented by the set FA of pairs (f, a), where each f is an .AR file and each a is a composed action. Thus the actions associated to a specific file f are defined by:

Definition 4.4 (File Actions)**.**

$$A_f = \{a \mid \ (f, a) \in FA\}$$

As an example consider the set: $FA = \{(f_1, a_1), (f_2, a_2), (f_3, a_3)\}$, then $A_{f_1} = \{a_1\}$.

We can extend this definition in a similar way to obtain the actions of a signature $\mathscr{S}$.

Definition 4.5 (Signature Actions)**.** *Let $\mathscr{S} = \{f_1, \ldots, f_n\}$ be a signature in $\mathcal{O}$, and let FA be the set correlating files and actions. Then the actions associated to $\mathscr{S}$ are defined by:*

$$A_{\mathscr{S}} = \{a \mid (f, a) \in AF \text{ and } f \in \mathscr{S}\}$$

Consider again the set: $FA = \{(f_1, a_1), (f_2, a_2), (f_3, a_3)\}$, and let $\mathscr{S} = \{f_2, f_3\}$, then $A_{\mathscr{S}} = \{a_2, a_3\}$.

When the KB is used to obtain suggestions for a new file f_x, we first obtain the most specific signature in $\mathcal{O}$, then we obtain the files already in the KB

that belong to this signature, and finally obtain the actions associated to each file, thanks to the set FA. These actions represent the suggestions for f_x. These three steps are detailed in the following.

Let $\Delta = \{f_1, \ldots, f_n\}$ be the set of all .AR files in $\mathcal{O}$, and assume $\mathcal{O}$ has been enriched (trained) using the refinement process in chapter 3, where all signatures for a situation in $\mathcal{O}$ have been discovered and added to $\mathcal{O}$. Given a new file $f_x \notin \Delta$, our task is to find the set $A_{\mathscr{S}_{f_x}}$ of actions that can be associated to f_x.

Step 1: Obtain the most specific signature for an .AR file The file f_x, might belong to more than one signature in $\mathcal{O}$, thus we select the most specific one, since it provides the most detailed description for the failure signature.

Definition 4.6 (Most Specific Signature)**.** *Given an .AR file f_x and an ontology $\mathcal{O}$ that contains learned signatures. Let $f_x \in \mathscr{S}_1, \ldots \mathscr{S}_n$ be the signatures for f_x. A* most specific signature *for f_x is defined by:*

$$\mathscr{S}_{f_x} = \mathscr{S}_i \mid \nexists \mathscr{S}_j \text{ with } \mathscr{S}_j \sqsubset \mathscr{S}_i$$

Consider again example 4.1, and let $f_x \equiv f_3$ (meaning that both files have exactly the same test results). Then the signatures for f_x are:

$$f_x \in \mathscr{S}_0, \mathscr{S}_3$$

Following definition 4.6, the most specific signature of f_x (in symbols $\mathscr{S}_{f_x}$) is $\mathscr{S}_3$.

Step 2: Obtain all the files in KB that belong to the selected signature Once we have selected the signature for f_x the second step is to obtain all files that belong to the signature. That is the files in $\mathcal{O}$ that are instances of the signature. In our example we have:

$$\mathscr{S}_{f_x} = \mathscr{S}_3 = \{f_2, f_3\}$$

Step 3: Obtain the actions associated to each file in the signature Once we obtain the .AR files that belong to the most specific signature $\mathscr{S}_{f_x} = \{f_2, f_3\}$ for f_x, we use the set FA to obtain its suggestions. Consider again the set: $FA = \{(f_1, a_1), (f_2, a_2), (f_3, a_3)\}$. We have found that $\mathscr{S}_{f_x} = \{f_2, f_3\}$, then according to definition 4.4 we have $A_{\mathscr{S}_{f_x}} = \{a_2, a_3\}$.

Finally, all those corrective actions associated to the signature $\mathscr{S}_{f_x}$ form the suggestions of file f_x:

$$Suggestions_{f_x} = A_{\mathscr{S}_{fx}}$$

Figure 4.13 illustrates the relations between the files, the signatures and the actions from example 4.1 in this section.

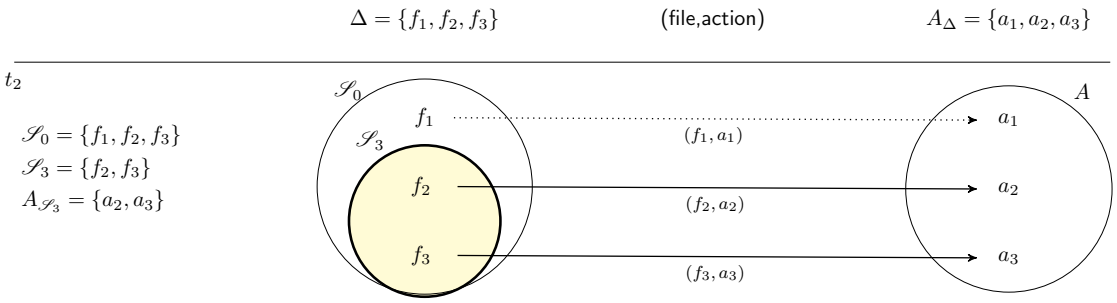


Figure 4.13: Illustration of the relation between signatures, .AR files and corrective actions.

4.4 A more fine-grained KB through feedback (Feedback)

In this section we detail the acquisition and integration of new information into the KB, through the technicians feedback. In the previous section, we have shown how to consult the knowledge base and obtain the suggested actions for a new .AR file f_x. After the file is consulted, and the suggestions are proposed to the technician, an investigations & repair phase follows. During this process, the technician resolves the failure detected by the .AR file. Once the true corrective action is known, we can start the next process.

The feedback process has two main tasks:

1. It aims to integrate and validate the feedback from the technician by recording the true corrective action and associate it to the corresponding .AR file. In this way it is made available for future consultations.

2. The .AR file in the feedback might contain information not seen before in the training stage (other properties). The second task of the feedback is to analyze this new .AR file, in search for new signature descriptions, and, if found, integrate these descriptions in the knowledge base.

As a result of this process, we obtain a more fine grained ontology and a larger set of corrective actions that can be suggested. These two tasks are described in the following.

Integrate the true corrective action This step is simple. The feedback of the user is given by the consulted .AR file f_x and the individual corrective actions

that solved the failure: $ia_1, ia_2, \ldots ia_n$. The full composed corrective action a_{f_x} associated to the .AR file, is the set containing all the individual actions:

$$a_{f_x} = \{ia_1, ia_2, \ldots ia_n\}$$

This information is added to the KB. Finally the set FA (of files-actions) is modified to link the file f_x to the composed corrective action a_{f_x}, by adding the pair (f_x, a_{f_x}) to FA.

Analyze the new .AR file in search for new signatures Since the signatures discovered in $\mathcal{O}$ depend on the .AR file analyzed, it is possible that new signatures are found following the arrival of a new .AR file. This is the case when the technician provides the feedback [2]. Intuitively, this process is a continuation of the learning phase, which is "restarted" from a specific point. The signature learning process explained previously in this chapter, in section 4.2, requires to specify: the file to guide the learning, the initial set of tests and the initial concept to refine. These three parameters are given by:

- Once the new file f_x is presented to the KB, we obtain the most specific signature $\mathscr{S}_{f_x}$ for f_x, where $\mathscr{S}_{f_x}$ is the concept to be refined.
- We obtain all files in $\mathscr{S}_{f_x} = \{f_1, \ldots, f_n\}$, and set $\Delta = \{f_1, \ldots, f_n\}$. This is the initial set of individuals for which we want to find sub-signatures.
- The refinement process is guided by f_x.

[2] We do not integrate new files when they are consulted, because the true corrective actions are not yet known.

Example 4.2. *Assume a new .AR file f_4, detailed in figure 4.14, for which we want to obtain its signature. Continuing example 4.1, if we consult the KB for the*

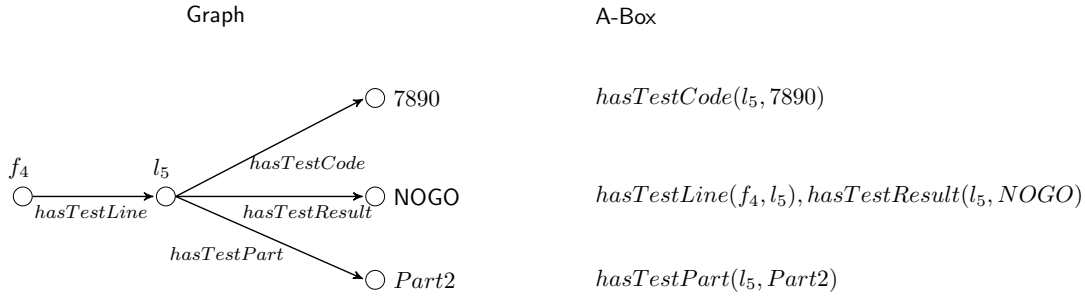


Figure 4.14: The graph representation and the corresponding A-Box of a new file f_4 presented to the KB for learning.

signature of f_4, we find that its most specific signature (out of $\{\mathscr{S}_0, \mathscr{S}_1, \mathscr{S}_2, \mathscr{S}_3\}$) is $\mathscr{S}_0$:

$$\mathscr{S}_0 \equiv \exists hasTestLine.(\exists hasTestResult.\{NOGO\})$$

And thus :

$$\mathscr{S}_{f_4} = \mathscr{S}_0 = \{f_1, f_2, f_3, f_4\}$$

Since f_4 is a new .AR file, we can use it to search for new signatures. If we refine the current signature found for f_4, we obtain a set of refinements of the form:

$$\alpha_{f_4}(\mathscr{S}_0) = \{\mathscr{S}'_1, \ldots, \mathscr{S}'_n\}$$

One of such refinements, $\mathscr{S}'_1$, is defined by:

$$\mathscr{S}'_1 \equiv \exists hasTestLine.(\exists hasTestResult.\{NOGO\} \sqcap \exists hasTestPart.\{Part2\})$$

Where the instances of $\mathscr{S}'_1$ are :

$$\mathscr{S}'_1 = \{f_3, f_4\}$$

Since $\mathscr{S}'_1$ is also the MSR for $\{f_3, f_4\}$, we record this new signature in the KB, making it available for further consultation.

If f_4 would be consulted again, its MSR will no longer be $\mathscr{S}_0$:

$$\mathscr{S}_{f_4} = \mathscr{S}'_1 = \{f_3, f_4\}$$

This example illustrates how new signatures can emerge in the presence of new samples.

From the above example, it is clear that new samples might lead to the discovery of new signatures. This process can go on continuously, and the resulting KB will become each time more "fine-grained", in the sense that more signatures that capture each time a more specific set of files can be made available.

We are interested in this evolution, since the more fine grained the knowledge becomes the more precise the corrective actions related to each signature can be.

The following and final example shows how the more signatures we find, allows us to minimize the set of corrective actions for the suggestions.

Example 4.3. *We are given the set of .AR files* $\Delta = \{f_1, f_2, f_3\}$*, the set of actions* $A_\Delta = \{a_1, a_2, a_3\}$*, the relations between them* $FA = \{(f_1, a_1), (f_2, a_2), (f_3, a_3)\}$ *and a concept that captures all files* $\mathscr{S}_0 = \Delta$*. In time* t_0 *we are presented with a new file* $f_x \equiv f_3$ *(meaning* f_x *and* f_3 *have the exact same results) for which we want to obtain the suggested corrective actions. We find that* $f_x \in \mathscr{S}_0$ *(since* $f_x \equiv f_3$*) and the associated corrective actions are* (a_1, a_2, a_3)*. This is illustrated in Figure 4.15. In time* t_1 *we obtain* $\mathscr{S}_1$ *as a refinement of* $\mathscr{S}_0$*. The operator*

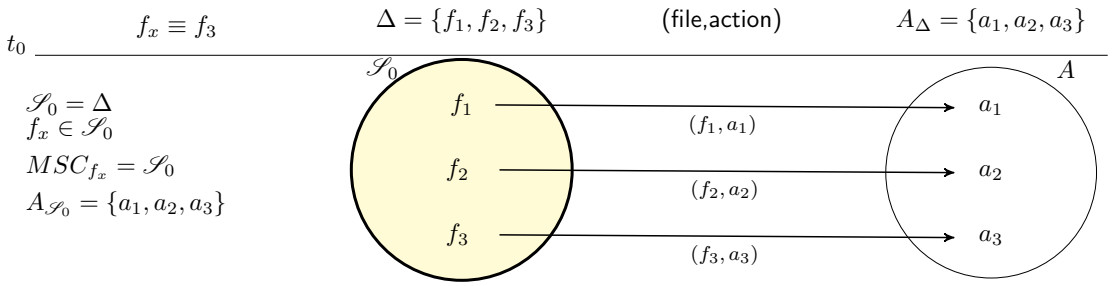


Figure 4.15: The state of the KB in time t_0.

α ensures that $\mathscr{S}_1 \subseteq \mathscr{S}_0$ and thus $\mathscr{S}_1$ can only contain equal or less elements than $\mathscr{S}_0$. Since the actions associated to each signature depend on the files in the signature, the suggestions for a refinement $\mathscr{S}'$ can only be equal or less than the suggestions of $\mathscr{S}_0$. In time t_1 we find $f_x \in \mathscr{S}_0, \mathscr{S}_1$. We select again the MSR ($\mathscr{S}_1$) and thus the suggestions are $A_{\mathscr{S}_1} = \{a_2, a_3\}$ (Figure 4.16). In t_3 the operator α is applied to $\mathscr{S}_1$ and the suggestions for f_x become $A_{\mathscr{S}_2} = \{a_3\}$. The modifications to the knowledge base, the set of actions, and the files in signature $\mathscr{S}_2$ are shown in Figure 4.17.

The next section of this chapter, presents the summary and the conclusions.

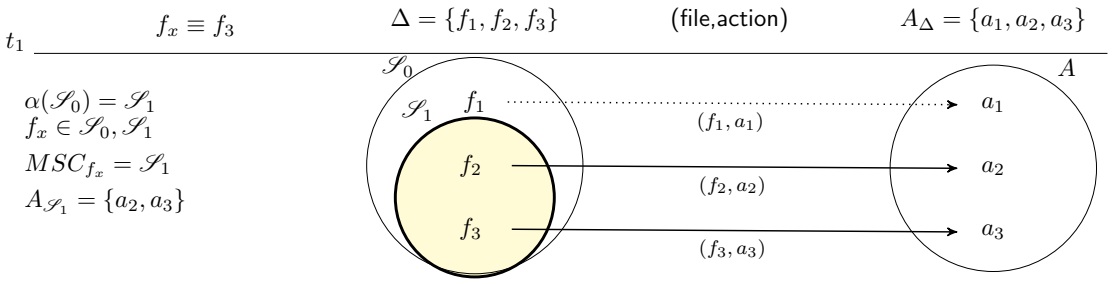


Figure 4.16: The state of the KB in time t_1.

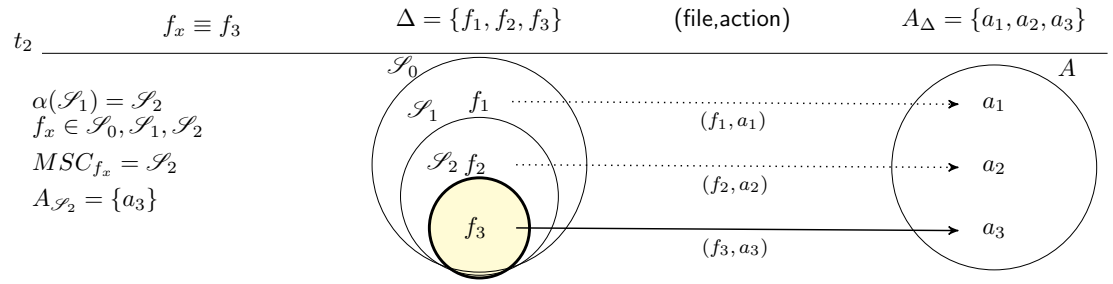


Figure 4.17: The state of the KB in time t_2.

4.5 Summary

In this chapter we have presented the relation between the approach for situation discovery introduced in chapter 3 and the problem of approximating failure signatures stated in the introduction in chapter 1. We have first detailed the .AR files, their sections, properties and the relevant features that are taken into account to model them and provide their representation in the ontology, called Thales Avionics Maintenance Ontology (TAMO). We have next made an analysis of the corrective actions we are required to provide as suggestions, and we have similarly established their main features to be considered and modeled. As explained before, in this work we are focused in the process of diagnosis and repair of the ELAC equipment in the maintenance workshop. Nevertheless, the design of TAMO is conceived to provide the basis to model maintenance for any other equipment, consider more types of tests, more complex corrective

actions and attach contextual information. This has been left out of the scope of the thesis, to establish limits for our problem. Nevertheless, for the sake of completeness, the main concepts and relations related to tests and corrective actions available in TAMO are also shown in figure 4.10.

With the details on the data sources, the mapping to the ontology concepts, and the structure of the ontology, we have proceeded to apply situation discovery to approximate signatures in TAMO, where the .AR tests modeled in TAMO are sets of individuals, and the signatures are DL descriptions for some of these sets. By refining the initial concept *TestMain* from which all .AR files are instances, we obtain the features that allow us to distinguish between them. These features tell us what are the elements of the tests results they have in common, expressed in the language of the ontology. For each set of files that can be described, we select a single concept (the most specific) as the representative for the signature of those files. The signature has in consequence associated the corrective actions corresponding to those files. These actions become, in turn, the suggested actions for any new file that is classified as belonging to the mentioned signature.

We have seen that there are two main tasks we need to consider when new, unseen .AR files are presented to the KB: consulting the KB to obtain the suggestions, and enrich the KB. For the consulting phase, we ask the ontology what is the most specific signature for the new file, and then the associated actions become the suggestions for the new file. For the feedback phase, we use the new .AR file to guide a new refinement process in search for new signatures.

In the next chapter, we specify how these processes (consulting and feedback) are implemented.

Chapter 5

Prototype

In this section we present the implementation of the approach, where a prototype to support avionics maintenance diagnosis has been designed and deployed. To start we briefly recall the maintenance diagnosis process for the ELAC and detail some important aspects. Then we explain the origin of the data, the context in Thales where the implemented prototype takes place, and its requirements. Next, we present the architecture of the system in a distributed environment allowing massive data processing and remote access. Then for the two main functions of the system : consult and feedback, we detail the implemented procedures and show how the final user interacts with the tool. The prototype had several versions, to improve its features. These are outlined before turning to the summary.

5.1 The diagnosis process

This section briefly recalls the diagnosis process for the ELAC equipment and explains the environment in which the implemented tool is expected to be used.

5.1.1 The Maintenance Procedure

Aviation maintenance is a rich, complex and highly technical domain. As such, it involves a diversity of locations, companies, services and actors with tight interaction and interdependence. The diversity of stakeholders and its distributed nature is reflected on the systems that compose it.

Our approach focuses on the maintenance workshop, whose clients are the airlines. When an equipment in an aircraft is found faulty, it is removed and replaced by another unit with the same capabilities. The removed unit is sent to a workshop, where it has to be diagnosed, repaired and certified.

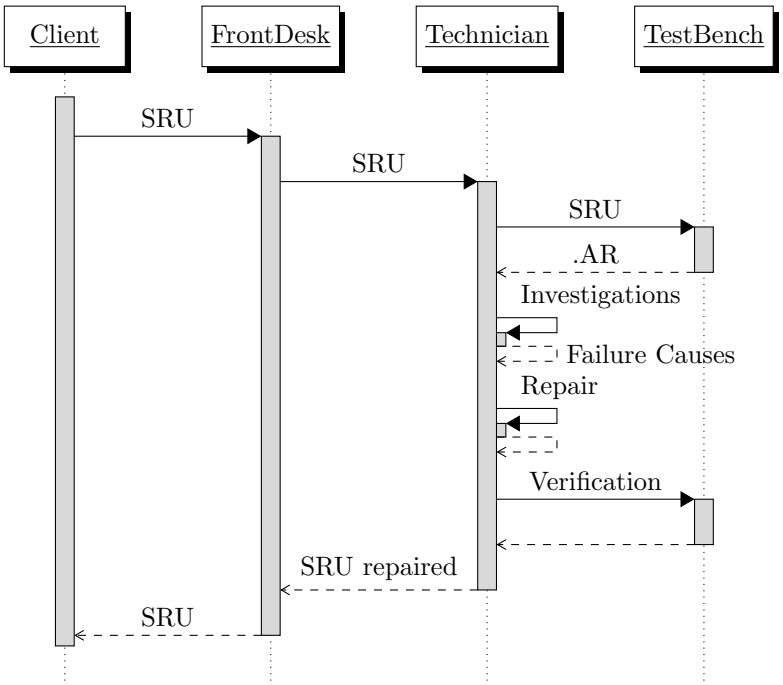


Figure 5.1: Sequence diagram for the repair of a SRU (Shop Replacement Unit).

Figure 5.1 illustrates the process: the *Client* (top left) sends the faulty equipment, called a Shop Replacement Unit (SRU) to the workshop to be repaired. The SRU is received in the workshop via the *FrontDesk*, along with the cause of removal of the SRU from the aircraft and other contextual information, like the aircraft type, the status of the systems of the aircraft, etc. The FrontDesk gathers all this information and creates a Work Order for the equipment. Once received and registered, the SRU is then transferred to the *technician* along with the Work Order, and the technician proceeds to test the equipment in the *Test-Bench* to verify if he can detect the same failure detected by the client, and determine its origin. The Test-Bench outputs the corresponding *.AR file* (All Results) with the test results. Out of these tests, a series of *investigations* take place (not necessarily by the same technician) to determine the exact *failure causes* and the corresponding maintenance actions to return the unit to a fully functional state. Once the *repair* process takes place, it has to be *verified* (again using the testbench) and if all tests are passed it is *certified*, returned to the FrontDesk, and finally returned to the client. All the actions performed by the technician for repair are filled in a report, and the more relevant parts of this report (like the list of components replaced and the root cause of the failure) are registered in a distributed system called ISEDIS.

5.1.2 The Investigations to Diagnose and Repair an Equipment

The investigations involve many processes and vary from equipment to equipment. In this section we explain in more detail how the technician uses the results of the test bench and what resources he has available to determine the failure causes for an ELAC equipment. This provides an insight on the complexity of the task, a better understanding of the procedure, and how exactly the implemented tool is intended to support him.

The investigations start with the analysis of the test results. The Test-Bench is a station designed to test a series of functions of the equipment and the results of the tests are presented to the technician as a text file. Each such file is composed of thousands of lines, where each line tests a specific function of the equipment. (for an illustration refer to Figure 4.2). The level of the tests is very low, in the sense that they test a functional chain, that is a specific circuit inside the equipment. If the circuit is working properly, the corresponding line in the test presents a result of *GO*, whereas if the test is not passed, the line presents a sanction of *NOGO*. Therefore a *test file* is composed of thousands of sub-tests (around 4000 sub-tests) each one with a result of either GO or NOGO.

Once the test results are available (Figure 5.2), the individuals tests (each line in the .AR file) with the result NOGO provides the technician with the references to the *functional bock* involved. The functional block is a logic diagram of interconnected components. In order to verify if they work properly, their physical location has to be established. Once the functional block(s) involved in the failure have been identified, the physical components are identified with the help of some software tools and/or manuals (Orchestra, Individual Drawing, CMM). The physical components provide a reference that allows to establish their physical location in the equipment, thanks to the *physical drawings*. Through this process the technician obtains a list of possibly problematic components, and proceeds to the *repair operation*. Once the replacements and repairs are finished, the technician reports in detail his/her actions, which are recorded in the SAP\ERP [1] system, for reference. The information in the SAP comes from several sources (WorkShop Order, Maintenance Report) and many of its fields are in textual form. i.e. they are not structured. From the records in the SAP module (and other sources) another technician selects the most relevant information regarding the repair process (main cause, components replaced, etc.) and feeds the tool for reliability and technical support ISEDIS. The advantage in ISEDIS is that a bigger part of the information is structured in proper fields (no longer as plain text).

[1] SAP\ERP is an enterprise resource planning software.

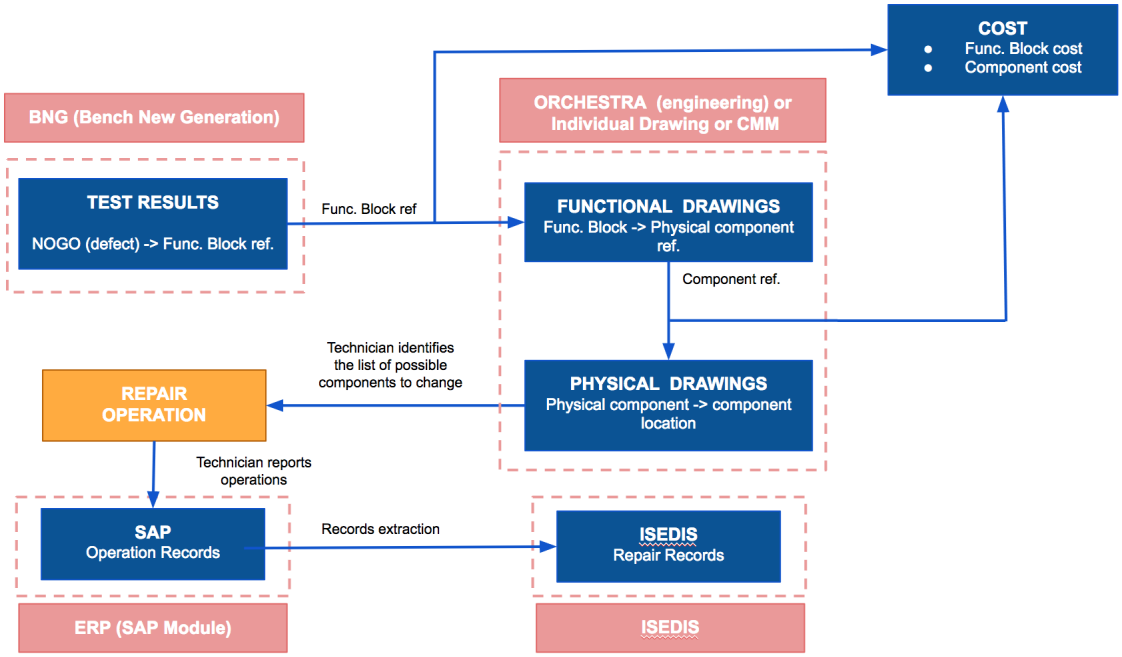


Figure 5.2: The systems and tasks involved in the maintenance process, from the point of view of the technician.

A limitation found in our case was that there is no direct correlation between the test obtained from the test-bench and the final report recorded in the ISEDIS system, that can be automatically exploited. The systems and the procedures are independent from each other, and thus reliable access to correlate both (the .AR files and the replaced components listed in ISEDIS) is not explicitly given. The correlation between the historical data of the tests and the corrective actions in ISEDIS system had to be "manually' established by an expert, and was not possible in all cases.

Reducing the investigations time, to determine the list of possible faulty components, is traduced in more efficiency in the repair process.

5.1.3 The format of the data sources

Regarding the format of the data, the .AR files are presented as plain text, and thus can be accessed through any standard text editor.

To include an .AR file in the ontology we need to map it to the model defined by the ontology. We can not do this directly. First the .AR files are parsed into a structured JSON format, and then a second process creates an OWL version

of the files, to be used by the ontology.

The JSON format is the preferred format to exchange information between the modules of our tool. The files representing the ontology, the T-Box, the A-Box, and intermediary files (temporal representations, discovered knowledge) are all in OWL2 format. The correlation between the .AR files and the corrective actions is given as an Excel file.

Finally, a copy of each .AR file consulted using our tool, and their structured JSON representations, are stored in Cassandra, ensuring we register all information provided to the system.

5.1.4 The e-Diag initiative

In order to support the technician in diagnosis and maintenance, the e-Diag (electronic diagnosis) initiative within Thales was launched. e-Diag aims to provide a knowledge base that allows the technician to obtain suggestions of the components to be repaired, a visualization tool that allows the easy physical location of the component and a dysfunctional model, to determine the mechanism of the failures. Within this context, our approach provides a solution for constructing and exploiting a knowledge base, to provide repair suggestions.

The e-Diag project forms part of the efforts in Thales for digitalization of the information they already hold in their key market areas, such as avionics. This digitalization process is aimed to add value to their current procedures and provide new revenue out of the exploitation of the digitalized information.

5.2 Requirements

The requirements presented in this section are concerned with the expected interaction of the user with the tool, its input and outputs, and the environment in which the system should be deployed.

5.2.1 Use Cases

To support the technician in the diagnosis task, two main use cases have been identified: consult the KB to obtain suggestions, and update/enrich the KB with new .AR files and corrective actions, which we call the feedback from the user.

Use Case: Consult

The tool should enable the user to provide an .AR file as an input, and receive the corresponding suggested actions as output. The suggested actions should be grouped by composed corrective actions, since it is interesting for

the technician to see when repairs that were made together solved the failure detected by the .AR file. Additionally for each suggestion, besides the listing of the components to be replaced, the number of cases seen where this suggestion is related to the failure have to be displayed. This value is also considered as the confidence of the suggestion. This use case of the tool is called *consulting the knowledge base* and is illustrated in figure 5.3.

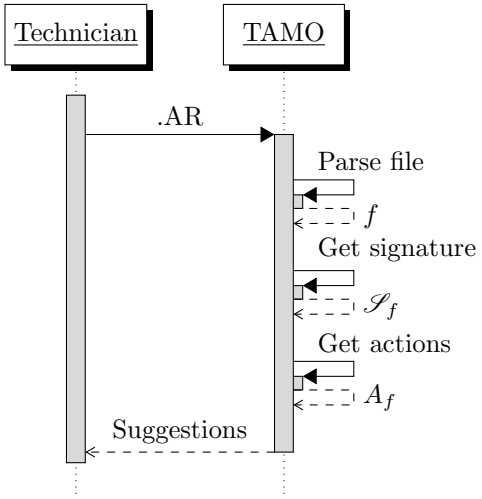


Figure 5.3: Interaction between the end user and TAMO. Case: Consult the knowledge base.

Use Case: Acquire Feedback

On a second phase, once the technician has obtained the suggestions and made the repair, he/she has determined the real corrective action (which might differ from the suggested one) and we can update the ontology using this feedback (figure 5.4) to increase the confidence on the given suggestion, or associate new corrective actions to failure signatures. The technician is to return to the tool, and provide the true corrective action for a previously consulted .AR file. Intuitively, if the suggestion provided when the file was consulted was correct, the confidence of the given answer should increase. If the suggestion was not correct, the true corrective action has to be acquired and integrated into the system. Finally, once the feedback is received, the tool counts with a new .AR file and its corrective actions. The system should analyze this new .AR file, to determine if new signatures can be discovered.

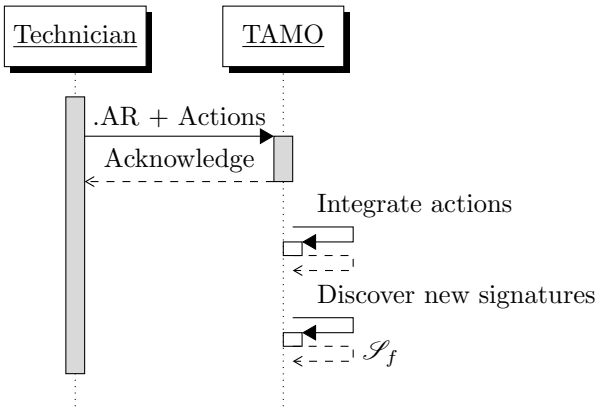


Figure 5.4: Interaction between the end user and TAMO. Case: Acquire users feedback.

5.2.2 Environment

On both previously described cases, the technician is located in his own workshop. Many repair workshops may exist. Thus the knowledge base has to be accessible from all workshops, and should be able to respond to consults and acquire feedback from technicians in any of these locations. Additionally in a full scale implementation, the system should be able to handle a large quantity of data to be consulted. Finally, as the information handled in avionics is sensible and confidential, the system has to be deployed in a secure environment.

5.3 System Architecture

An overall diagram of the system for consulting the knowledge base is shown in figure 5.5, where three different work groups are involved: to the left the end user in **Thales Avionics Workshop** (TAV) who accesses the system functions via the Human Machine Interface (HMI), and two teams to the right in **Thales Research and Technology** (TRT). One team is in charge of the distributed infrastructure and provides the *web services* that allow the end user to interact with the knowledge base, and on the far right another work team is in charge of the development, implementation and monitoring of the *learning and reasoning engine* which directly accesses and modifies the knowledge base.

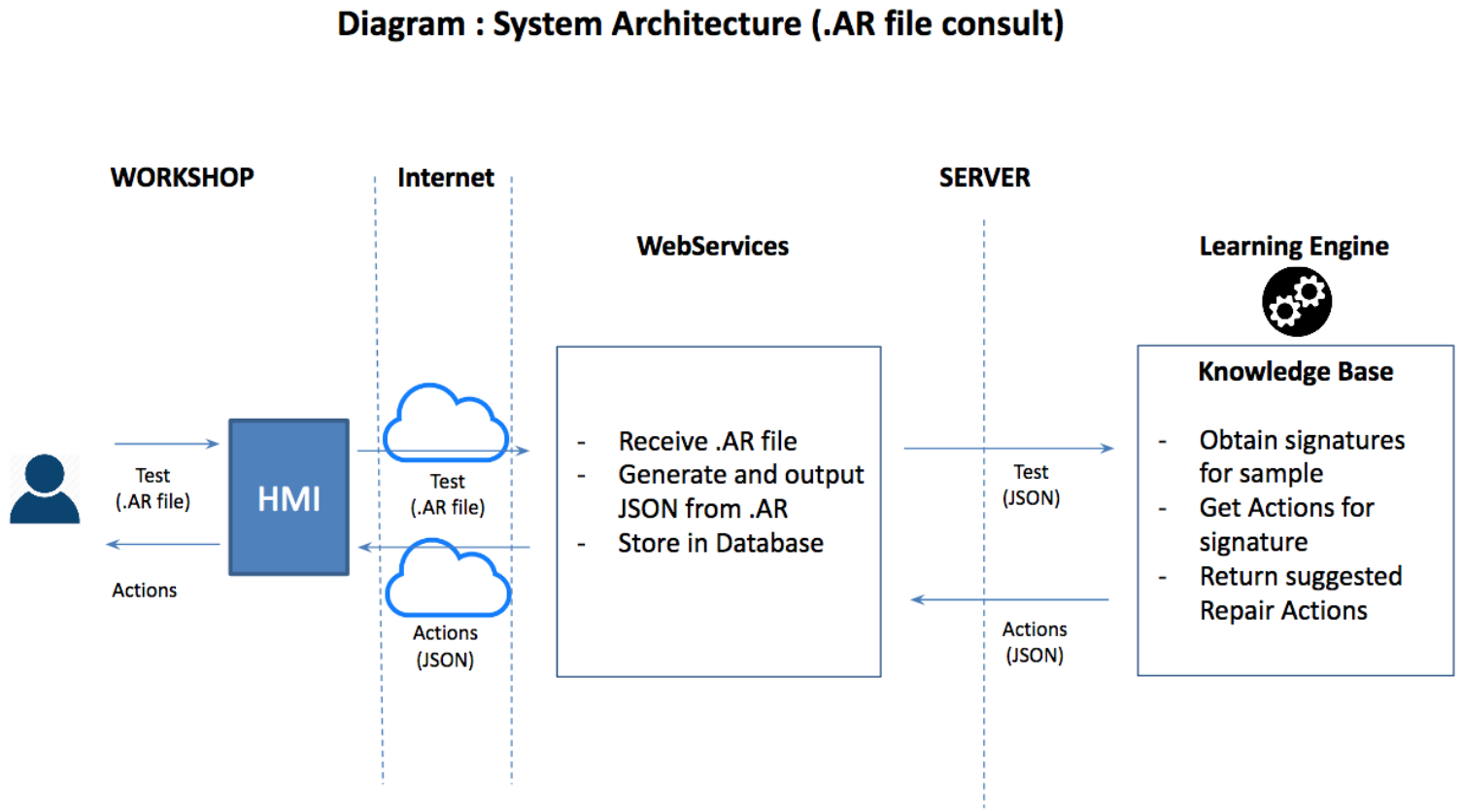


Figure 5.5: The implementation of the prototype to consult the knowledge base. On the left the end user provides the test results. These are passed over a VPN through internet to Thales TRT. In TRT, the web services provide the file to the KB, and transmit the answer back to the user.

In figure 5.5 we show the end user (technician) on the left who provides a new .AR file (test) for which he/she requires a suggestion. The HMI is a web-based interface, built using HTML and Javascript mainly. The technician uses the HMI to upload the file, which is received and pre-processed in TRT. The server in TRT is a Tomcat Server which allows for the Java coded web pages to be served. The received .AR files, need to be transformed into a structured format to enable the mapping of the information of the file carried into the ontology. We have selected JSON as the format to structure the .AR file, because it is a well known, simple format which can be easily exchanged and exploited through several applications. Then, a reasoner is used (Hermit/JRDFox) to decide if the uploaded .AR file is an instance of a known signature, in which case we can obtain the associated repair actions. Once the analysis is ready, the answer of the suggested repair actions is returned back to the web services, again in JSON format. This information is finally returned to the end user via the web service.

The TRT cluster

The distributed system is mounted over a BigData architecture to allow massive processing and distributed access. The platform is composed of a cluster of five computers, plus a dedicated server to host the HMI, and the corresponding proxies and firewalls required by Thales. In this setting, each request from the HMI to the server in TRT, is associated to a stream of messages managed by Kafka. Kafka is a messages manager which allows for distributed processing. Intuitively, each request to the server is considered a message and a list of messages is maintained by a central manager (broker). Each node in the cluster is capable of requesting the central manager (broker) a message to process, and once the processing is finished, the message is erased from the list. This allows for parallel and distributed processing when the number of messages are massive. Additionally, each message processed, involves the reception and processing of an .AR file. The information generated in this process is registered in Cassandra, which is a database with distributed capabilities. In Cassandra, the data is distributed among the nodes in the cluster allowing for redundancy and replication of the data. Redundancy refers to the existence of several nodes in the system, thus if one node fails, another can continue to perform the required tasks. Replication involves sharing information so as to ensure consistency between redundant resources (nodes), this involves the synchronization of the nodes in a cluster.

5.4 Technical specifications

The user interface is designed using HTML (V4) and JavaScript (1.7), and is fully operational using any up-to-date web browser. In the implementation of the prototype we have opted for FireFox (V 50 +) as the default browser. It is to note that, since we are in a controlled and secure environment, the tool is only accessible using a computer connected to the Thales internal network.

The learning engine of our approach, as well as the web service are implemented in Java (V. 1.7). As reasoners we have used Hermit 1.3.8 and JRDFox v6.

The system uses Cassandra as the Data Base Management System, and the web server is provided by TomCat.

5.5 System Functions

In this section we detail the main functions of the system. We start by explaining the initial setup of the system, and how the sample data was used to construct a trained knowledge base. Once this knowledge base is available, we show the implementation of the two main functions of the tool: consulting and feedback.

5.5.1 Access the system - Initial setup

Before deployment, the knowledge base has been trained using our approach and the available training samples. In this initial setup the descriptions (signatures) are discovered.

In the first connection to the system, the welcome screen appears. Once inside the system, the main screen (figure 5.6) provides access to the two main functions: Upload a file to consult the KB and obtain the suggestions, and provide feedback for an already consulted file.

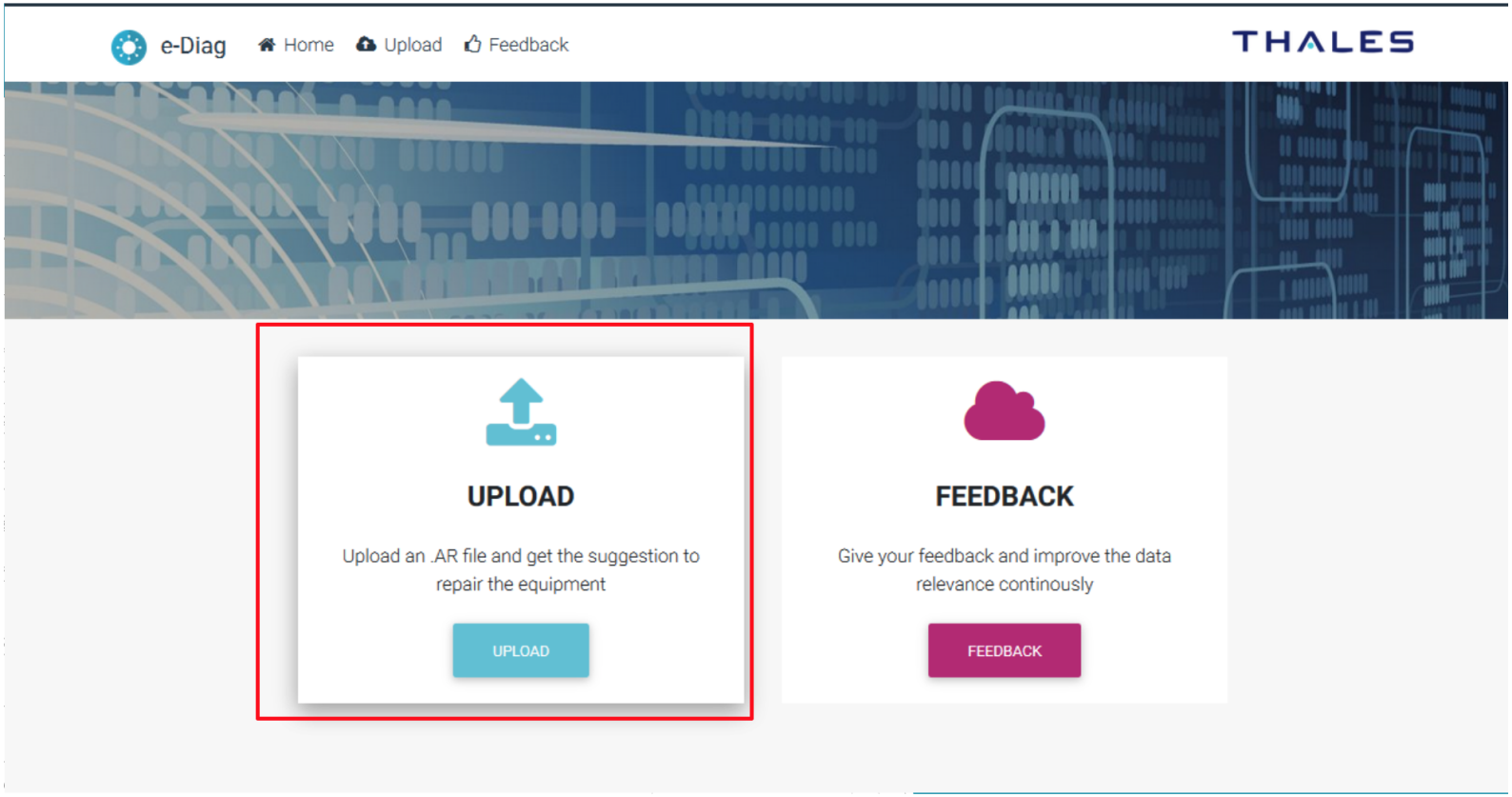


Figure 5.6: The initial screen.

5.5.2 Consult the KB

The main use of the system is to consult the knowledge base to find the failure signature that best suits an .AR file test, and obtain the suggestions. This process is detailed in figure 5.7.

In the top left of diagram in figure 5.7 we have the Human-Machine Interface (HMI) and the *preprocessing* module. The HMI is an HTML-Javascript based interface to allow the technician to interact with the system. It connects the client machine with the web service coded in JAVA, and managed by a tomcat server. The HMI enables the user to select and upload an .AR file which is then transmitted to the preprocessing module in the server.

Once the file has been received, it is first parsed to JSON to provide a structured and shareable representation of the .AR file. The structure of the parsed file corresponds to the model for the files defined in the ontology. Once the JSON file is available, a second module maps each entry in the JSON file to an axiom that can be added to the ontology. This provides with an A-Box representing the file, stored in OWL format in memory. The DL representation of the file is represented by f in the diagram. The preprocessing module also stores in the database (Cassandra) a copy of the file received in both: raw format and in JSON.

Once f is available, it is transmitted to the *classify* module, where we use a reasoner to consult the knowledge base, and obtain all the signatures $\mathscr{S}$ known for f. Then, according to our preference criteria, we select the most specific signature $\mathscr{S}_f$ of f. Once the signature $\mathscr{S}_f$ is selected, we consult again the knowledge base to obtain all its instances $\mathscr{S}_f = \{f_1, \ldots, f_n\}$, that is all known files that share the same signature. The output of this module is the selected signature along with the .AR files that comprise it.

The *get suggestions* module, consults the historical data in the database, to retrieve the corrective actions associated to each file in $\mathscr{S}_f$, thus constructing the set of corrective actions for the signature. The duplicates are eliminated and the actions are formatted into JSON. The JSON file containing the suggestions is returned to the HMI, which decodes the file and provides a visual representation for the actions.

The next section shows the user interface for the consulting phase.

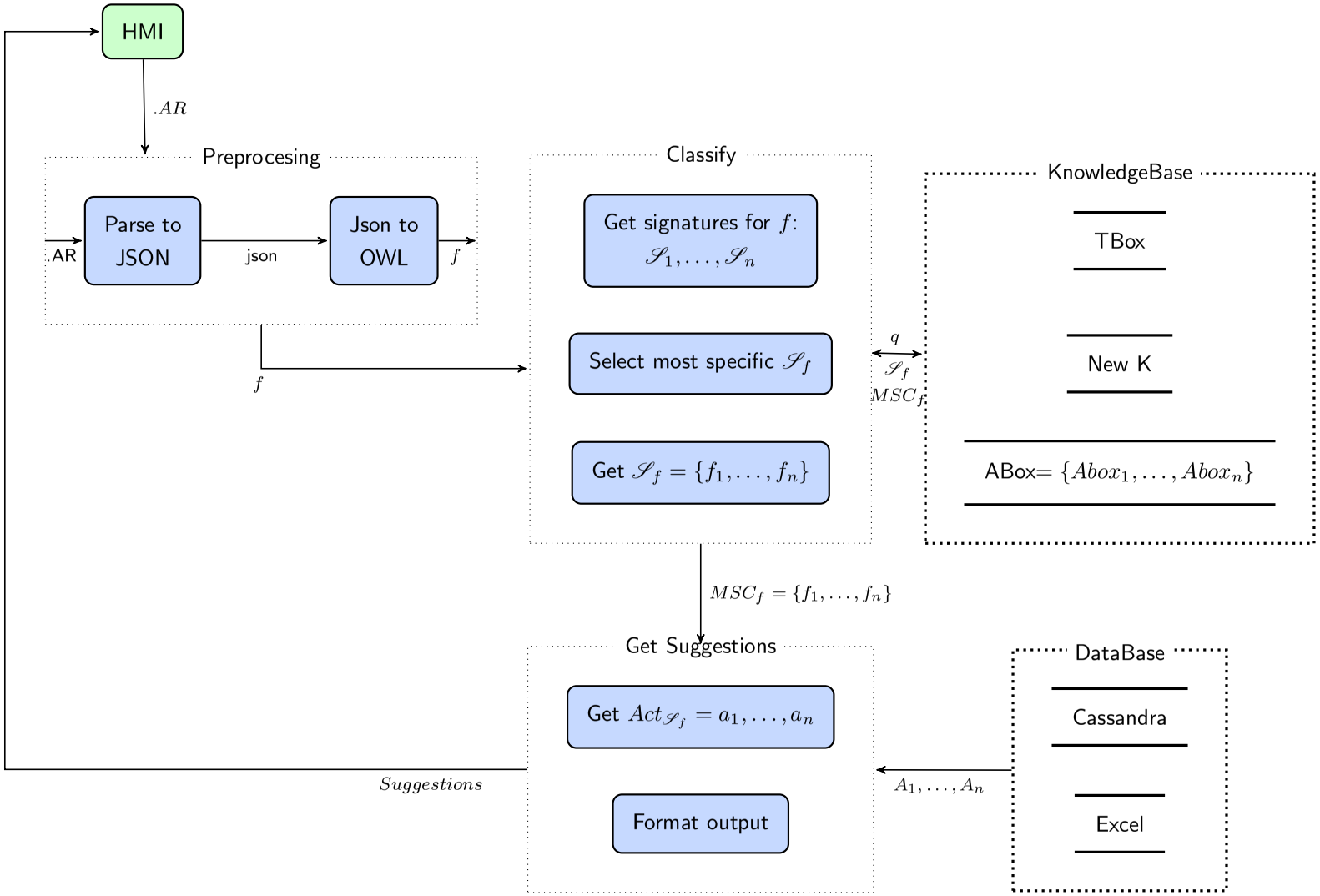


Figure 5.7: Modules for consulting the knowledge base.

HMI

In the consulting phase, first the technician uploads an .AR (All Results) file, which is the output of the test-bench. Figure 5.8 shows the interface through which the technician can upload the .AR file.

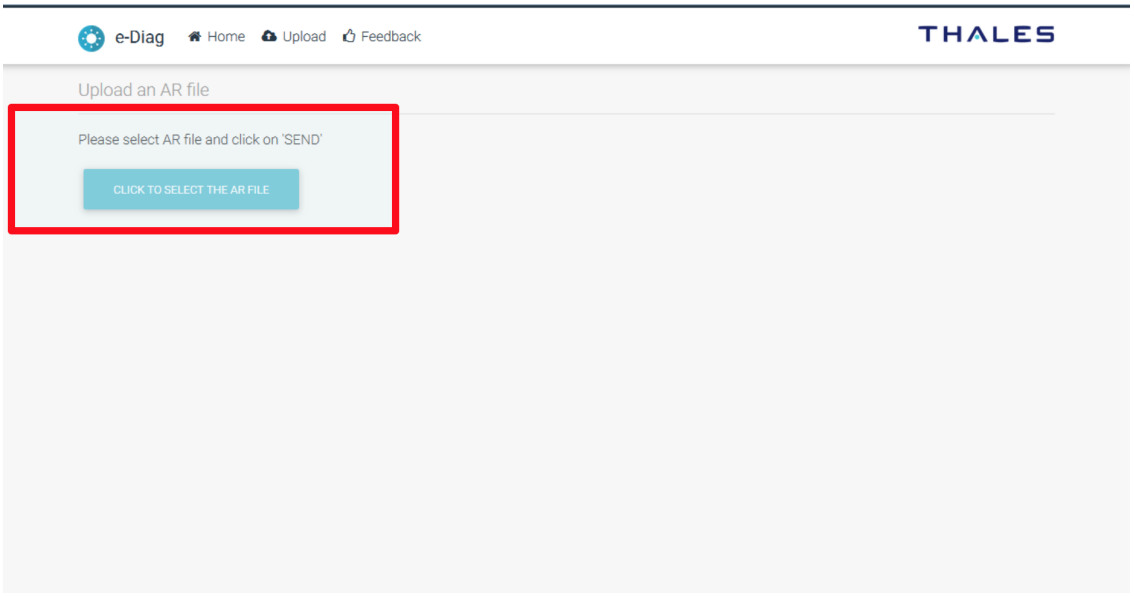


Figure 5.8: The upload file screen.

The "CLICK TO SELECT THE .AR FILE" button, opens a pop-up window to select the file from the desired location (hard disk drive, usb key, etc.). Once the file is chosen, its name will be displayed on the top of the screen and a "SEND" button will appear to start the analysis.

Depending on the type of file and the size of the knowledge base, the system will take between a few seconds (20s) to several minutes (+5m) to analyze the file, and retrieve the suggested actions. Figure 5.9 shows the suggestions provided to the technician by the tool.

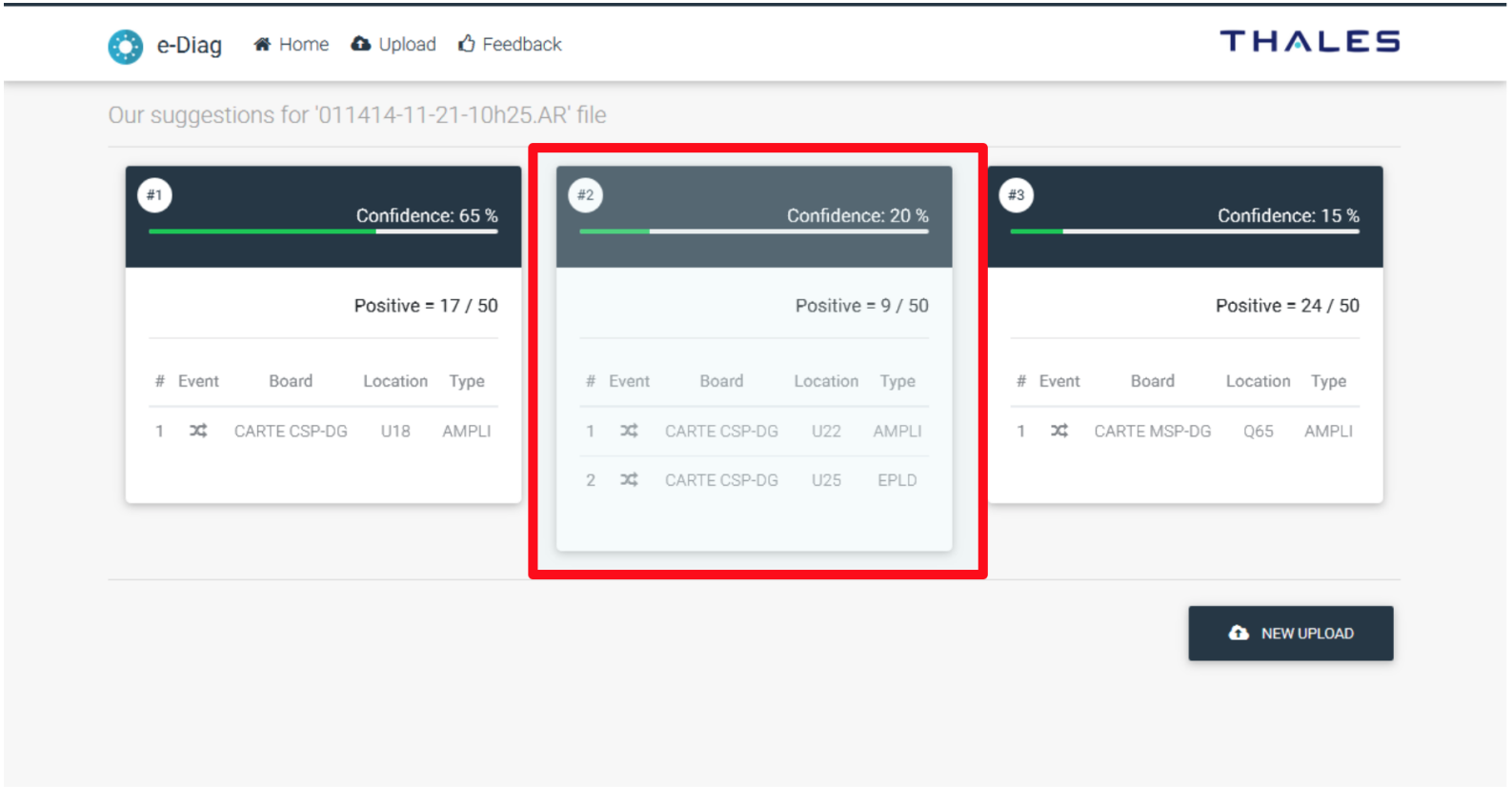


Figure 5.9: The suggested corrective actions for the consulted file.

Several suggested actions might correspond to a single file. In Figure 5.9 we see that three suggestions were found for this case. Each suggested action displays:

- Number of suggestions.
- Confidence: the confidence assigned to this particular suggestion to repair the failure, given by the number of similar cases found.
- Number of positive cases found: How many cases similar to the failure found, were solved by this corrective action (proportional to the confidence).
- Replacements. These are the components to be replaced, identified by: their board, location and type.

The consulting phase ends here. The technician may use the suggestions to help him diagnose and repair the equipment.

5.5.3 Integrate Users Feedback

Once the technicians have made the repair, based or not on the proposition(s) given by the consult phase, the right corrective action to repair the equipment is known (which might differ from the suggested one). This feedback is used to increase/decrease the confidence of the given suggestions, add new corrective actions to the system, and analyzed the consulted .AR file to search for new signatures. The implementation of this process is detailed in the following.

Figure 5.10 illustrates the process for acquiring the technicians feedback. On the top left, we have the HMI as in the consulting phase, which enables the user to select the previously consulted .AR file, to validate the given suggestions.

Since we already treated the file before, we already have a structured (JSON) version of it and the file does not need to be uploaded again. Once its unique identifier is selected through the HMI, the *preprocessing* module, retrieves its JSON version from the database (Cassandra) and passes it to the OWL parser, to obtain f, where f is the OWL representation of the .AR file. The file f and the validated *actions* are then send to the *add* module.

In the *add* module, a first stage is concerned with *adding* the actions and the .AR file permanently into the knowledge base and the database, where the set AF of files-actions is stored.

The corrective action is split into its atomic components, i.e. the individual corrective actions, and they are added to the database, each one as a pair (file, actions). Next the file f is permanently added to the knowledge base's A-Box. Note that in the consulting phase, we did not add f to the KB, because we didn't know the corrective action and thus we could not associate it to any suggestions.

It should also be noted, that the A-Box of the KB (at the bottom right of the diagram) is composed of several sub A-Boxes. This is necessary to reduce the load imposed to the reasoner when querying a large database. The intuition behind this split A-Box structure is to provide to the reasoner with the minimal necessary information, so it can provide an answer faster. Because of this, the *add* module has to handle to which A-Box it is going to add the current .AR file.

Once the information is stored, a background learning phase takes place, which no longer involves the user. The *learn* module receives the new file f and uses it to discover new signatures. We first obtain the current signature $\mathscr{S}_f$ and all its instances. This is done to avoid to use all files in the analysis. If f belongs to $\mathscr{S}_f$ and there exists a more specific signature, then this signature can be found by refining $\mathscr{S}_f$. We obtain the refinements of $\mathscr{S}_f$ using the operator $\alpha_f(\mathscr{S}_f)$. All signatures found in this process are then added to a separate OWL file "New K" which can be seen as an extension of the T-Box in the knowledge base. This structure allows to separate the original terminology (T-Box) from the acquired knowledge (New K). Once the signatures learned, they are available for further consulting.

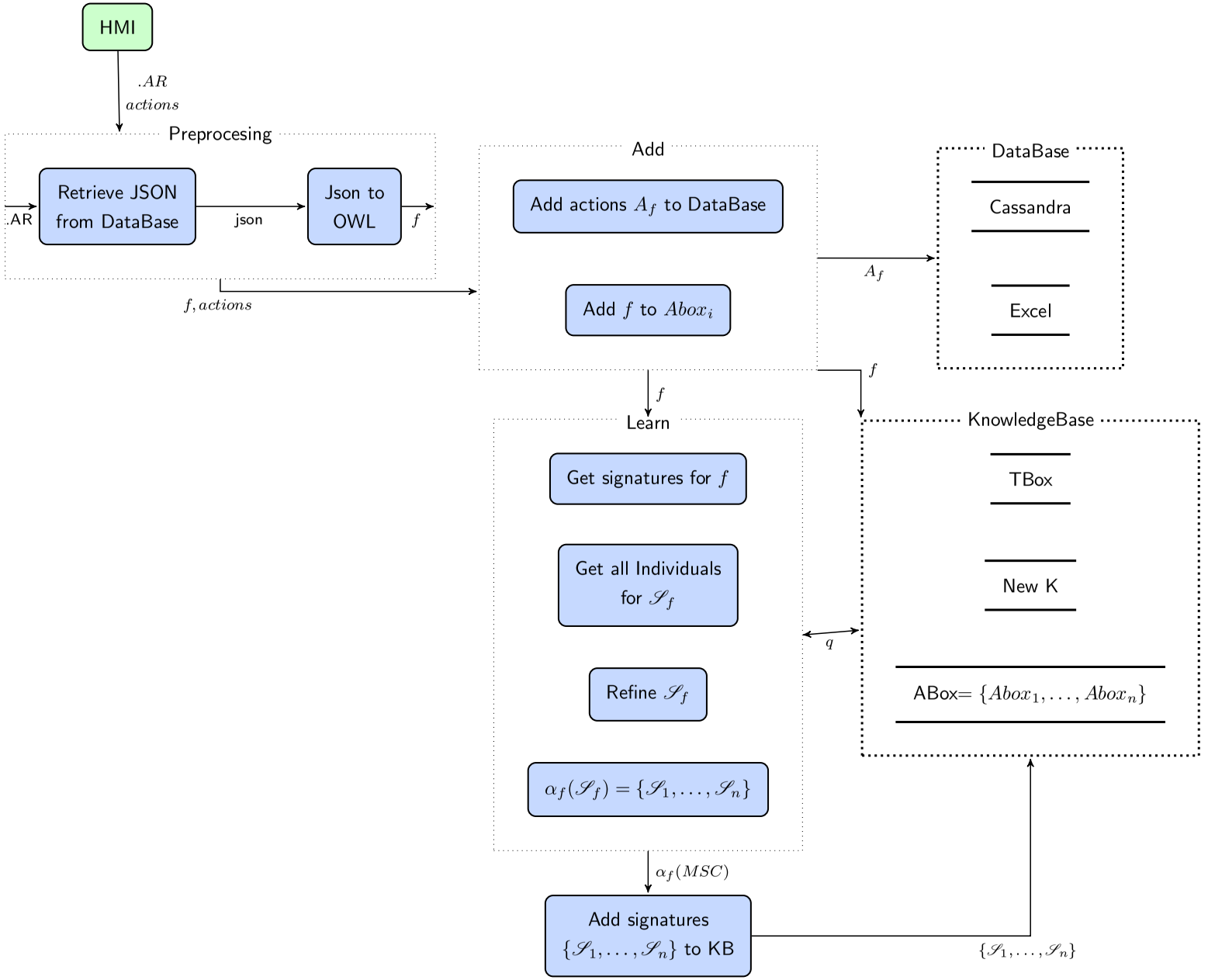


Figure 5.10: Flow Diagram for the Feedback process.

HMI

In this section we show and explain the user interface for the feedback process.

Once feedback has been selected in the initial screen (Figure 5.6, the system displays to the technician a list of all the consulted files, for which the feedback has not yet been given (figure 5.11), the user must select the correct file, and the system will show the previously found suggestions, so that the user can validate them.

The validation screen (figure 5.12) is similar to the one obtained in the consulting phase, but this time the tool gives the user the chance to validate the suggestions (meaning the suggested replacement action indeed solved the failure) or to turn to manual feedback if no proposition was correct. Each proposed suggestion presents a "Validate" button, and if none of the suggested actions is correct, the HMI provides a "SWITCH TO MANUAL FEEDBACK" button, on the bottom right of the screen. Manual feedback implies that none of the given suggestions were correct or that they were not accurate.

If manual feedback is selected (figure 5.13), the user is presented with a form to fill in the components replaced. For each component replaced he/she must fill in: the board, the location and the type of each replaced component. Next, an additional learning phase takes place to search for new signatures that could be discovered, thanks to the new .AR file. All this process runs in the background and is transparent to the user.

Finally, either through the automatic or manual feedback, the system will acknowledge it has properly received the information from the technician with a "Thank you" message. This ends the feedback phase of the prototype.

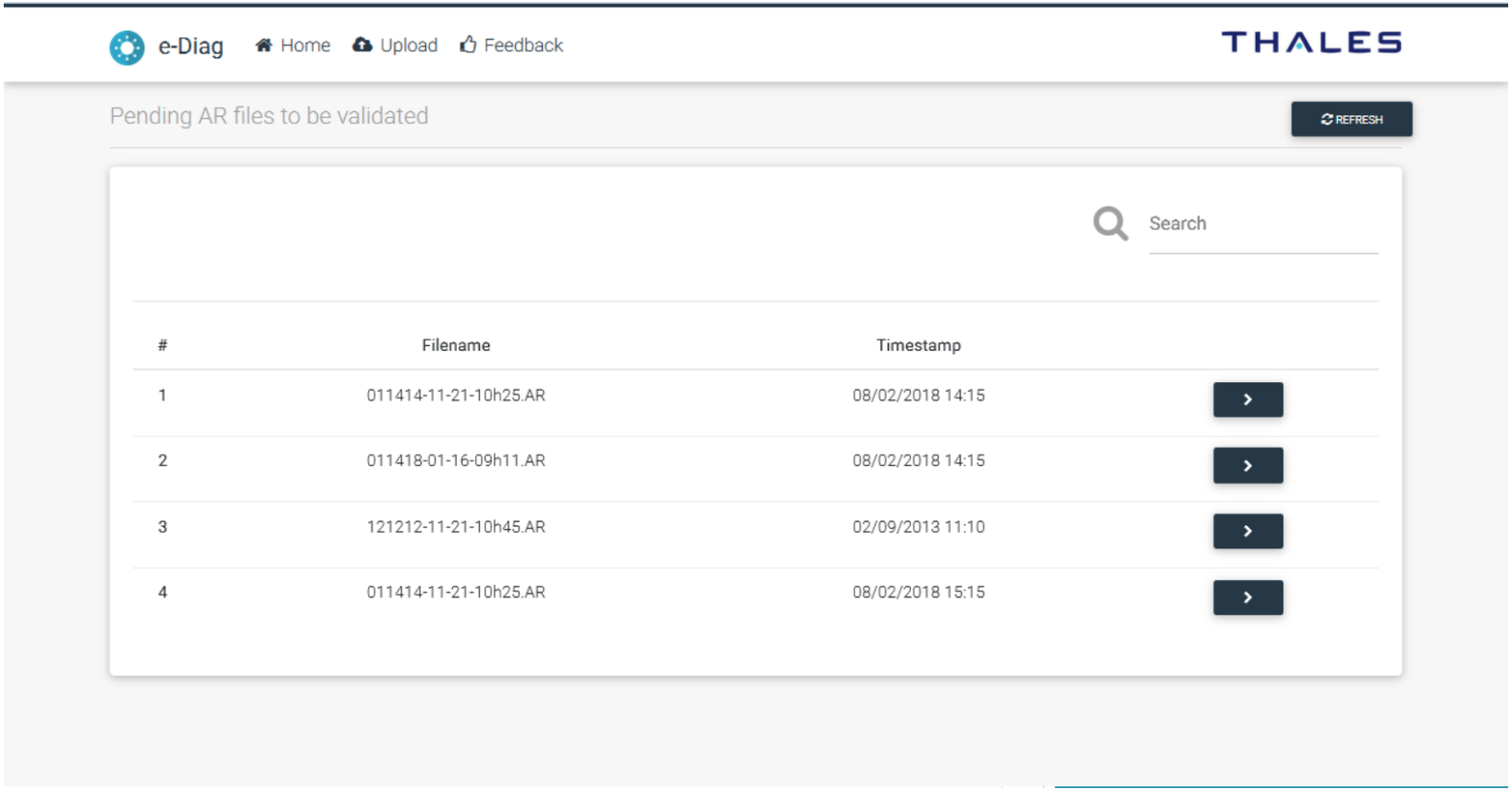


Figure 5.11: The consulted files awaiting validation.

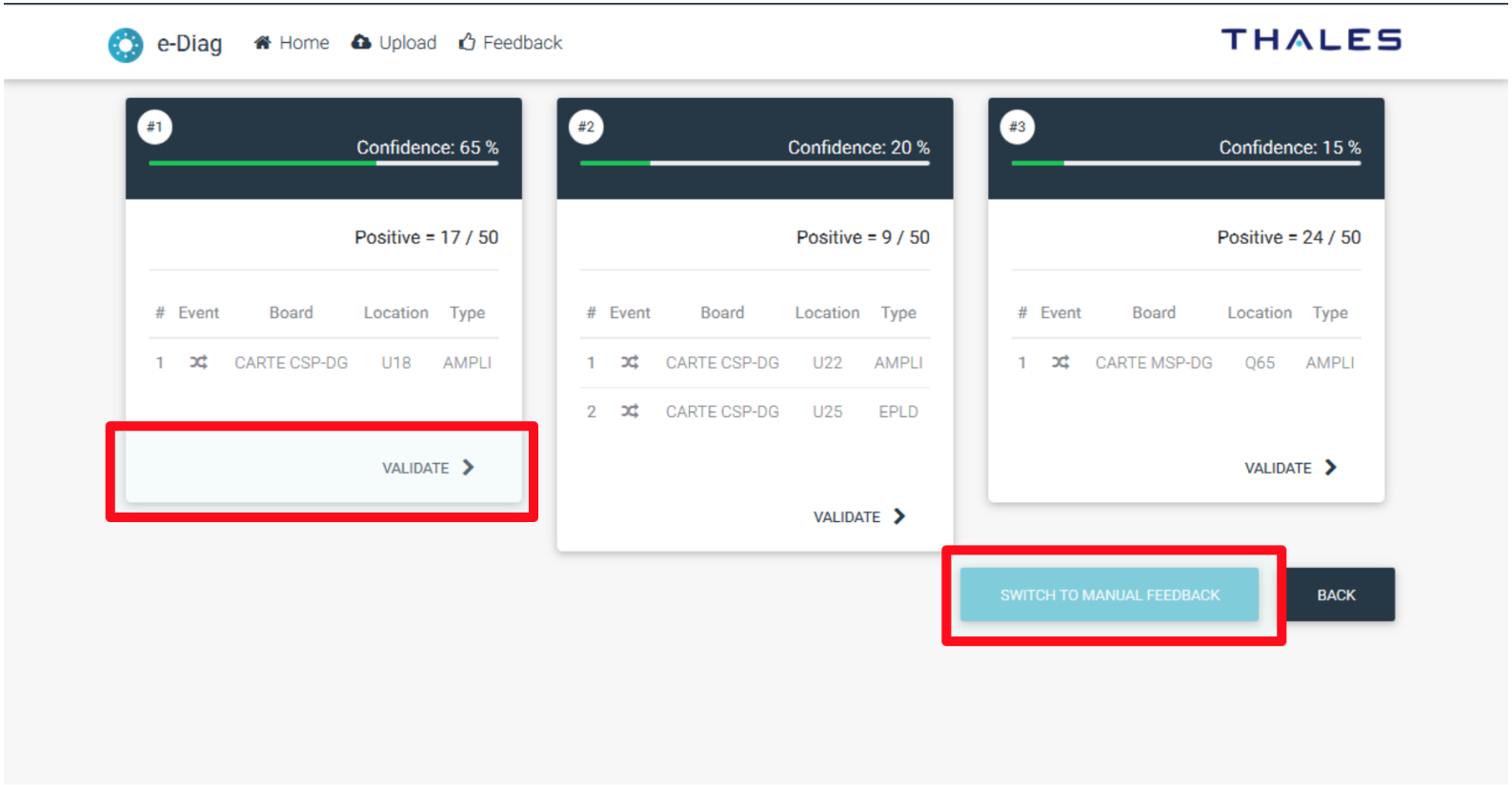


Figure 5.12: The validation screen.

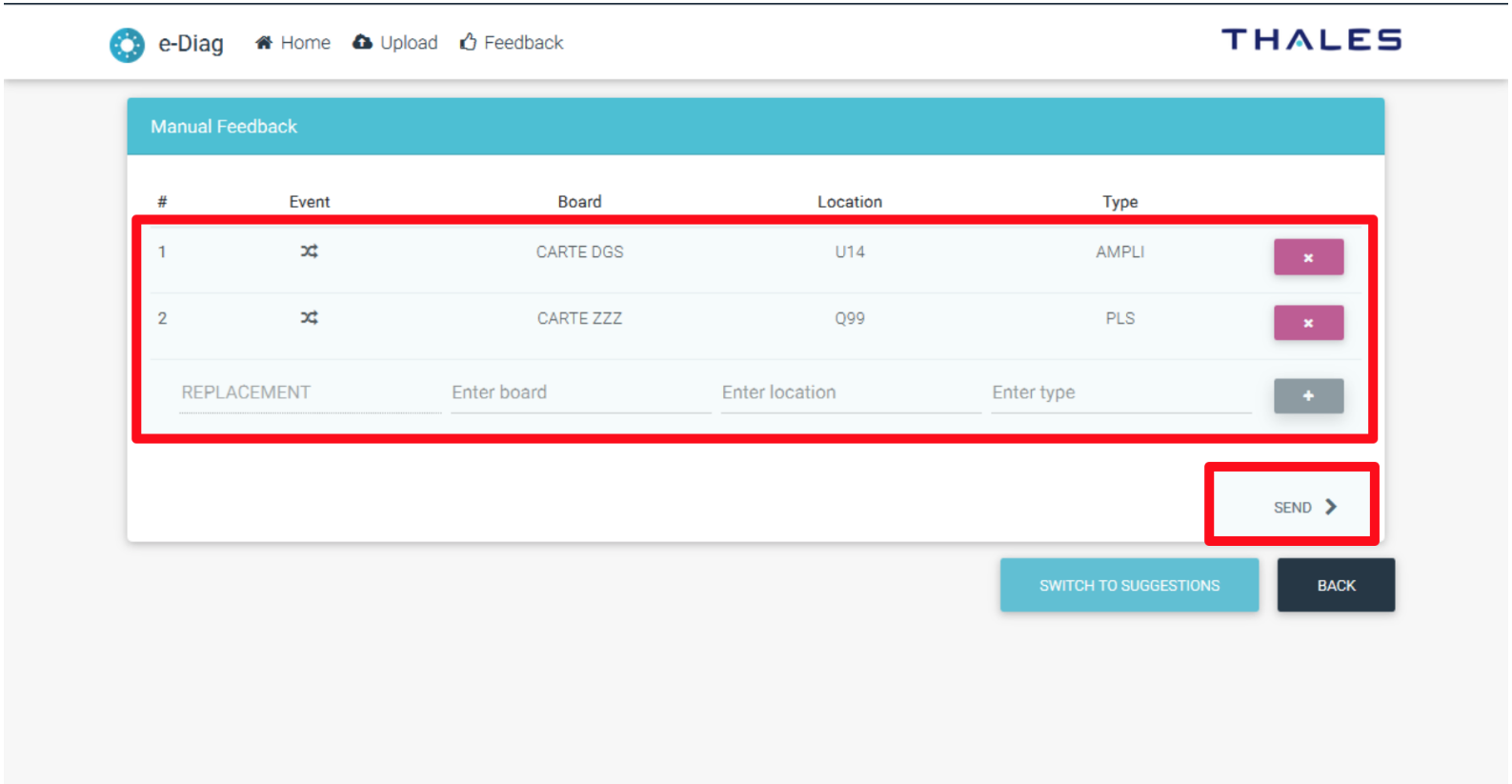


Figure 5.13: The manual feedback screen.

5.6 The different versions of the prototype

The prototype has passed through a series of iterations in its development which can be summarized as follows.

- **V1** Implements the algorithms and specifications of the approach. Serves as a proof of concept and to determine the limits of the implementation.
- **V2** Improves version one in the amount of data it can handle and in the time needed for the consulting and feedback phases.

It is important to outline that the resulting trained KB from V1 and V2, follow the same specification.

5.6.1 Version 1

The first version of the prototype (V1) serves as a proof of concept. This version takes as input the ontology with an empty A-Box, a set of file samples with their corresponding actions, and finds the signatures that can be derived from each sample. This version is mainly intended to: verify that the results are inline with the specifications of the algorithms, to select and format the output presented to the end user, and to evaluate the scalability and limits of the solution. Efficiency at this stage was not a main objective, but instead to evaluate the feasibility of the implementation and identify its limits. The consultation times under this version would vary from some seconds up to several (15+) minutes, and the learning phase of one file could take between 20 minutes up to one day.

5.6.2 Version 2

Out of the tests and usage of the first prototype, we found several important points to take into account for a better evaluation of the approach. First, we found that the knowledge base grew very fast with a very small number of .AR files and that it became challenging for the reasoner to provide answers after ˜20 files were analyzed. Second, we identified some procedures, like the computation of the minimal A-Box and the iteration process, which could be optimized. We also found that it was challenging for the reasoner to handle concept definitions that involved equivalences within the sub-concepts that conformed it. To overcome these limitations, the second version (V2) considers the following extensions.

- We have analyzed up to which level we can split the knowledge base, in order to provide to the reasoner only the amount of information strictly necessary to perform the computations. Out of this analysis we have split the ontology in several files. One file contains the original T-Box, another

file contains the discovered knowledge given by the *signatures*, and the A-Box is split in several files, each one containing a limited number of .AR files. Out of the usage and experiments with version V1, we have found that a limit for the reasoner arrived when the A-Box reached about 20 000 axioms. Thus the current implementation takes up to 5 .AR files in one A-Box, before creating a new file. In this manner, we have increased the limit size of the knowledge base, from 25 files on version one, to 50, 100 and 150 files in version two. An important feature of this division, is that some of the processes, notably learning and consulting the knowledge base, can be parallelized up to certain point. Implementing parallel computing requires high expertise at a conceptual level, at an implementation level and for the consideration of the underlying architecture of the system, thus it is out of the scope of this thesis. Nevertheless, the analysis has been made on the feasibility, and the second version of the prototype opens the doors for implementing some of these processes in parallel.

- Out of the analysis made in version V1, we have evidenced that the calls to the reasoner were the most time consuming tasks. We have reduced the calls to the reasoner where possible in version V2.

- To evaluate whether the limitations for the reasoner were given by the types of task given, or by the reasoner itself, we have developed two sub-versions of V2. The first sub-version V.2.1 uses HermiT[2] (v. 1.3.8) as the reasoner and the second sub-version V.2.2 replaces the main calls to the reasoner with RDFox[3], which is a parallel, RDF[4] based reasoner, for which Oxford Labs has provided us with a research license for the experiments.

5.7 Summary

In this chapter we have specified the two main uses of the implemented system: consult and feedback. We have started by describing the context under which the diagnosis support tool takes place within Thales, and by describing how the technician currently analyses, repairs and reports faulty ELAC equipment. These considerations establish the requirements for the prototype. We have shown the designed architecture for the system to overcome the distributed access, and to leverage on BigData architecture, to enable scalability of the system. Within this context, we have specified the modules that carry on each of the main functions of the system, along with the interfaces that allow the user to interact

[2]http://www.hermit-reasoner.com/
[3]http://www.cs.ox.ac.uk/isg/tools/RDFox/
[4]Resource Description Framework graphs

with the tool. In the specification of the modules of the system, it is shown the correspondence with the definitions and specifications made in Chapter 3 and Chapter 4, where the approach and its application to our problematic is detailed. Finally we have provided an insight on the different versions of the prototype and the reasons why these versions took place.

This chapter allows not only to establish the characteristics of the prototype, but enables to appreciate the overall path from the conception of a solution, through the specification of the background theory and the development of the necessary algorithms, to the design of a system and its implementation. This chapter closes the gap between conception, formal specification and implementation of a feasible solution.

Chapter 6

Evaluation

In this chapter we present the evaluation of the approach through the implementation of the prototype. It is divided in three sections. Section 6.1 evaluates the relevance and the number of suggestions proposed by the approach. Our hypothesis here is that the more fine grained the knowledge base, the more specific failure signatures[1] we can find and therefore we can minimize the number of suggested corrective actions. Next, in Section 6.2 we provide measurements and analysis on the efficiency of the implementation, given by the response times. In Section 6.3 we compare our approach with DL-learner, which is a similar tool used to discover concepts. Additionally, in Section **??** we present a survey designed for user feedback [2].

Finally, in section 6.4 we present a summary of the obtained results and our conclusions about the experiments.

6.1 Evaluation of the Relevance and the Number of Suggestions

In this section we evaluate a) how pertinent the suggestions are with respect to the given sample data, and b) the number of responses obtained using different versions of the knowledge base. These experiments aim to test our assumption that the more files we analyze, the more specific failure signatures we can find (traduced in minimizing the suggested actions) and the better the results we can provide (whether the given suggestions suit the problem or not). To this end we consult several versions of the knowledge base (trained with 25, 50 and 100 samples) and for each version we analyze the proposed suggestions versus the

[1] Failure Signatures in defined in chapter5

[2] At the moment of the redaction of the thesis this evaluation was ongoing.

real corrective actions, and the number of suggestions proposed.

6.1.1 Key Performance Indicators

To evaluate the relevance and the number of suggestions, we have defined a series of key performance indicators detailed in figure 6.1. Our approach aims to discover failure signatures from a set of given samples, which are later used to classify new unseen .AR tests. Our assumption is that the more knowledge that is analysed, the richer the resulting knowledge base, which should traduce in improving the quality of the discovered failure signatures. The quality of the discovered signatures is given by the relevance of the associated suggestions, and the specificity of the signatures.

Criteria		Objective
Relevance of suggestions	**Composed actions**	This indicator tells us if the suggested corrective actions are relevant to solve the failure detected by the .AR files. An .AR file may have several suggestions associated, we consider that the suggestions were **relevant** if among them there is one that **solves** the failure. A **composed action solves** a failure if all individual actions in the composed action are required for the failure to be solved. (A composed action is a set of individual actions)
	Individual actions	This indicator is similar to the previous one, but considers **individual actions**. An individual action is **relevant** if it is an element of the composed action that solves the failure.
Specificity of discovered failure signatures	**Number of files in failure signature**	Our hypothesis assumes that the more information that is presented to the knowledge base, the more signatures we can specialise, thus improving the model accuracy and increasing its quality. For each discovered failure signature a number of .AR files in the KB are associated. The more specific the signature, the less files that belong to it. To verify if this evolution takes place, we analyse the discovered signatures of different versions of the knowledge base.
	Number of suggestions per consultation	This is another perspective from the same principle in the previous indicator, that is more directly related to our objective of minimising the amount of suggestions provided to the technician. Some consulted files will have associated too many suggestions because a proper signature is not known. If the suggestions are too many, they become useless even if among them there is a relevant one. This happens when a very general signature has been associated to a file. We expect that the richer the knowledge base, the more signatures we can specialise, and that less files will be associated to a very general signature. This should in turn translate in the number of suggestions being reduced. To verify if this evolution takes place, we analyse the discovered signatures of different versions of the knowledge base.

Figure 6.1: The Key Performance Indicators (KPIs) for efficacy.

Each signature has associated a set of suggested corrective actions. To evaluate the relevance of the suggestions, we have established the first two indicators in table 6.1: **Relevance of composed actions**, and **Relevance of individual actions**. Recall that to return an equipment to a fully functional state, several component replacements might take place. Each replacement is called an *individual action*, and the set of all individual actions required to solve a failure is called a *composed action*. Note that a composed action could also be comprised of a single repair [3]. These first two indicators allow to evaluate the relevance of the suggestions for both cases. We assume that individual corrective

[3]Composed and individual actions are defined in chapter 5 The Prototype

actions are more frequently shared by .AR files than fully composed actions, which would enable us to provide partial relevant suggestions, when no exact composed actions can be found. The reason to evaluate both types of answers is to asses if the rate of relevant suggestions increases when we consider as well the individual actions.

The last two indicators in table 6.1: **Number of files in failure signature** and **Number of suggestions per consultation** deal with the aspect of how specific are the discovered signatures. When few information is available, we expect that the discovered signatures would be too general. That is, too many files will be associated to such signatures. When a file is consulted, and the signature found is too general, it means that the knowledge base does not know a better (more specific) signature for the consulted file. Better in this case is a signature which is *more specific* since it should provide less suggestions, and to which less files should be associated.

Before we introduce the methodology to evaluate these KPIs, let us present some of the characteristics of the data available for our experiments.

6.1.2 The Characteristics of the Data

The data available for our experiments imposes several challenges. It is relatively small, when considering the number of samples versus the number of features the samples have, and when considering the number of possible resolutions (corrective actions) that exist. We find that most of the files, do not share a corrective action with another file. This implies that for some failures (and their failure signatures) at most one sample can be used to learn the corresponding signature. This fact is one of the factors that motivates an analytical based learning, in contrast to a frequent pattern or statistical learning process. This also implies that the prediction that can be achieved by a model based on the experience of the technicians, reflected in the sample data, can not be 100 percent accurate: if an expected corrective action is not present in the sample data, it is not possible for the model to predict that action. Nevertheless, the motivation to implement a prototype and its evaluation is to show up to which degree a model based on our approach can provide satisfactory answers (if any), and to estimate the applicability, benefits and challenges of such systems. These challenges reflect the complexity of dealing with real data and provide on-filed evaluation of the prototype.

In the following we highlight the main properties of the data that should be considered when interpreting the results of the evaluation.

A summary of the information from the .AR files used in the evaluation is given in figure 6.2 where it can be seen that there are only 15 composed actions

Measurement	Value	Unit
Number of files	**150**	Files
Individual actions associated to a file (min)	1	Individual actions
Individual actions associated to a file (max)	23	Individual actions
Number of different composed actions	120	Composed actions
Number of different individual actions	239	Individual actions
Composed actions present in more than 1 file	15	Composed actions
Individual actions present in more than 1 file	58	Individual actions
Number of files that share a composed action with another file	45	Files
Number of files that share an individual action with another file	90	Files
Total files that share a composed action with more than one file	25	Files
Composed actions present in two or more files	5	Composed actions
Individual actions present in two or more files	31	Individual actions
% of files that can be given a composed answer	30,0 %	
% of files that can be given a partial answer	60,0 %	

Figure 6.2: A summary of the properties of the data to evaluate efficacy. The table shows a summary of the number of corrective actions (composed and individual) associated to each file, and the amount of files that share corrective actions.

that are shared by two or more samples (files). On the other hand, these 15 actions are associated to 45 files.

Any approach that uses this data set, will obtain the suggestions from the correlated samples in the historical data given by the experts experience.

If the full set of 150 files is divided into a training set and a validation set, to cross validate a model, a specific suggestion will only be available if the corresponding file is within the training set. In this setting, from figure 6.2 it can be seen that since only 45 files share composed actions, the remaining 105 files (approx 70% of the 150), have no possibility to be given a correct suggestion. Indeed, for a suggestion to be available, the associated file has to be in the training set.

To further increase the difficulty of the problem, from the 45 files that do share a composed action, 20 of them share the action with only one file. And thus only 25 files share actions with more than one sample, which in turn corresponds to only 5 corrective actions.

An alternative to overcome these figures is to consider partial corrective actions (individual actions). Since our goal is to provide suggestions of repair actions, these suggestions can be completely correct (for which few samples exist), partially correct or not relevant at all.

If we consider individual actions, as partially correct suggestions, in the sense that they give a hint or a partial resolution for the failure, these numbers change.

There are 90 files that share an individual corrective action with another file, representing 58 individual actions. Thus, we would expect that the number of relevant suggestions increases if partial suggestions are made available.

To be able to compare the use of composed and individual actions, both are considered in the evaluation of the relevance of the suggestions.

Our problem has an additional complication: multiple signatures may correspond to multiple actions and different actions might solve the same test result. This impacts the meaning of the signatures computed and the learning process.

If we know that for the set of tests $\{t_1, t_2, t_3, t_4\}$ action $\mathcal{A}_1$ follows. Then we would consider these tests as the positive samples of the signature $\mathscr{S}$ we are looking for. Then, the task is to obtain the most specific signature such that $\mathscr{S} = \{t_1, t_2, t_3, t_4\}$. But we could find that t_1 and t_2 have one test result, and t_3, t_4 have another, and that all of them are still solved by the same action $\mathcal{A}_1$. Therefore we are looking for two signatures $\mathscr{S}_1$ and $\mathscr{S}_2$ "hidden" in the set of samples. Since we can not know this in advance, a strategy needs to be implemented to consider these cases.

Considering the other direction: when a signature $\mathscr{S}$ is solved by multiple actions $\mathcal{A}_1, \mathcal{A}_2$. We will have two sets of samples $\{t_1, t_2\}$ for $\mathcal{A}_1$ and $\{t_3, t_4\}$ for $\mathcal{A}_2$, thus we will find two signatures $\mathscr{S}_1$ and $\mathscr{S}_2$. But if all samples belong to the same signature, it should be the case that $\mathscr{S}_1 \equiv \mathscr{S}_2 = \{t_1, t_2, t_3, t_4\}$, which is the intended answer. This property has another implication in the learning process: the fact that test t_3 is not in the set of samples for $\mathscr{S}_1$ does not imply it has a different signature, and we can not consider t_3 as a negative sample for $\mathscr{S}_1$. This is why we do not use negative samples to guide the search.

6.1.3 Methodology

To evaluate our previously defined KPIs, we need to establish how the set of samples will be used. Cross-validation offers a methodology to select which parts of the data that will be used for training and which ones for validation. Next, to estimate how well the discovered signatures behave, we defined some metrics over the defined KPIs. Both, cross-validation and the selected metrics are introduced next.

Cross Validation

When developing a model for classification, the accuracy of the model depends on the samples that are used for training it. Once the model is trained, its accuracy can be verified by using a second set of validation data. Out of this validation, the generalization error can be determined (i.e. how well unseen samples can be predicted).

The objective of cross-validation is to estimate the generalization error, and ultimately minimize it.

Given a set of samples, it is divided in several partitions and some of these partitions are used as *training data* while others remain as *validation data*. In this way we can estimate the error of the model with respect to the partitions. It is evident that depending on the selected partition the results may vary, and thus there are several ways of implementing cross-validation, like k-fold cross validation and leave-one cross validation [James et al., 2013]. The main difference lies on how the training and validation sets are constructed, and on how many times/partitions are tested. In this thesis we focus on k-fold cross validation.

k-Fold Cross Validation

In k-fold cross-validation, the original sample set is randomly partitioned into k equal sized subsamples. A single subsample is retained as the validation data for testing the model, and the remaining k-1 subsamples are used as training data. The cross-validation process is then repeated k times, with each of the k subsamples used exactly once as the validation data. The k results can then be averaged to produce a single estimation.

Assume we have 100 data points. For k-fold cross validation, these 100 points are divided into k equal sized and mutually-exclusive *folds*. For k=10, we might assign points 1-10 to fold #1 , 11-20 to fold #2, and so on, finishing by assigning points 91-100 to fold #10. Next, we select one fold to act as the validation set, and use the remaining k-1 folds to form the training data. For the first run, we might use points 1-10 as the test set and 11-100 as the training set. The next run would then use points 11-20 as the validation set and train on points 1-10 plus 21-100, and so forth, until each fold is used once as the validation set.

Metrics

In pattern recognition, information retrieval and binary classification, precision (also called positive predictive value) is the fraction of relevant instances among the retrieved instances, while recall (also known as sensitivity) is the fraction of relevant instances that have been retrieved over the total amount of relevant instances [Olson and Delen, 2008, Duda et al., 2012]. Both precision and recall are therefore based on an understanding and measure of relevance, and are applied to the first two KPIs defined in figure 6.1 (Relevance of composed actions, and Relevance of individual actions).

These metrics are based on the notions of true/false positives and true/false negatives. Before presenting how to compute these indicators, we first define

how the positive and negative samples should be understood in our context.

Definition 6.1 (Positive, Negative, False Positive and False Negative Samples)**.** *Given and .AR file f, a failure signature $\mathscr{S}$, the correct corrective composed action A_f for f, and the set of actions $Actions_{\mathscr{S}} = \{A_1, \ldots, A_n\}$ associated to $\mathscr{S}$, we say that file f is a:*

- True positive (tp) *for $\mathscr{S}$, if $\mathscr{S}$ is the MSR for f and $A_f \in Actions_{\mathscr{S}}$.*
- False positive (fp) *for $\mathscr{S}$, if $\mathscr{S}$ is the MSR for f but $A_f \notin Actions_{\mathscr{S}}$.*
- True Negative (tn) *for $\mathscr{S}$, if $\mathscr{S}$ is not the MSR for f and $A_f \notin Actions_{\mathscr{S}}$.*
- False negative (fn) *for $\mathscr{S}$, if $\mathscr{S}$ is not the MSR for f but $A_f \in Actions_{\mathscr{S}}$.*

Precision is defined by:

$$\mathrm{P} = \frac{\mathrm{tp}}{\mathrm{tp} + \mathrm{fp}}$$

Whereas recall is defined by:

$$\mathrm{R} = \frac{\mathrm{tp}}{\mathrm{tp} + \mathrm{fn}}$$

The two measures can be used together in the F1 Score (or f-measure) to provide a single measurement for a system. The f-measure combines precision and recall as their harmonic mean. It is defined by:

$$\mathrm{F} = 2 \cdot \frac{\mathrm{P} \cdot \mathrm{R}}{\mathrm{P} + \mathrm{R}}$$

The above measures are defined at the level of the individual signatures in the knowledge base. In our case, each discovered signature can be seen as a binary classifier of which an .AR file is or is not an instance, but the tool as a whole is a system composed of multiple binary classifiers. Once we obtain the precision and recall for each relevant signature, we need to aggregate them since we are interested on evaluating the system as a whole. This can be achieved by the micro and macro average of the precision and the recall. Given n = number of all signatures we want to evaluate, the Micro Average Precision (MicroAP) and Micro Average Recall (MicroAR) are defined by:

$$\mathrm{MicroAP} = \frac{\sum_{i=1}^{n} \mathrm{tp}_i}{\sum_{i=1}^{n} \mathrm{tp}_i + \sum_{i=1}^{n} \mathrm{fp}_i}$$

$$\mathrm{MicroAR} = \frac{\sum_{i=1}^{n} \mathrm{tp}_i}{\sum_{i=1}^{n} \mathrm{tp}_i + \sum_{i=1}^{n} \mathrm{fn}_i}$$

Similarly the Macro Average Precision (MacroAP) and Macro Average Recall (MacroAR) are defined by:

$$\mathrm{MacroAP} = \frac{\sum_{i=1}^{n} \mathrm{P}_i}{n}$$

$$\mathrm{MacroAR} = \frac{\sum_{i=1}^{n} \mathrm{R}_i}{n}$$

6.1.4 Experiments

The relevance of the provided suggestions and the specificity of the signatures found, can be evaluated in different contexts. First we evaluate how well a model trained using 100 samples performs (leaving a third of the samples to validate it) and then we evaluate whether the results improve when more/less information is used for training.

Thus this section is subdivided in two experiments:

- **Relevance and Specificity of the Suggestions** For this experiment, we use cross validation. We partition the set of all samples in several sub sets, and use each test as a validation set once.

- **Evolution of the KB** To estimate if the performance increases with the evolution of the knowledge base, we obtain the KPIs for knowledge bases of different sizes (25, 50 and 100 samples).

Experiment 1 - Relevance and Specificity of the Suggestions

In this first experiment, we evaluate the relevance of the returned suggestions and the specificity of the discovered signatures. For both evaluations we use k-fold cross validation with a size of $k = 3$, which represents a third of the samples. This means that the full set of 150 samples is divided in three partitions p_1, p_2 and p_3, each one containing 50 .AR files. Each partition is used once as the validation set, while the other two are used for training the knowledge base. The size of k is selected considering that there are few samples that share composed corrective actions ($\backsim 30\%$) and the time necessary to train each knowledge base. The low rate of shared composed actions means that roughly, 1 out of 3 samples has a chance to be correctly classified by a model. If we select a partition size that is too small, it is possible that the validation set contains no classifiable files ($\backsim 70\%$ of files), and if the partition is too big it is possible that all classifiable files are in the validation set, leaving the training set with only unclassifiable files.

Using $k = 3$ provides us with three versions of the knowledge base: $KB_{100}^{p_1 \cup p_2}$, $KB_{100}^{p_1 \cup p_3}$ and $KB_{100}^{p_2 \cup p_3}$. When the knowledge base $KB_{100}^{p_1 \cup p_2}$ is obtained using the partitions $p_1 \cup p_2$, each file in p_3 is consulted against $KB_{100}^{p_1 \cup p_2}$. For each such consultation, the file is assigned a signature (given by the MSR). Each signature has associated a) a set of corrective actions, that represent the suggestions, and b) a set of files. Under these considerations, when consulting the samples in p_3 we analyze:

- Per each file consulted:

  - the number of relevant composed actions suggested vs. the relevant individual actions suggested, in figure 6.3 ,
  - the relevant actions (composed and individual) suggested vs. the total number of suggestions in figure 6.4, and
  - the number of files per suggestion, and number of suggestions, in figure 6.5.

- Additionally, per each signature found we analyze:
  - the precision, recall and f-measure, shown in figure 6.6.

In the following, we provide the results and analysis corresponding to the knowledge base $KB_{100}^{p_1 \cup p_2}$, and the aggregate results for the three versions of the knowledge base. The full results for each version ($KB_{100}^{p_2 \cup p_3}$ and $KB_{100}^{p_1 \cup p_3}$) are available in the Appendix B.

Results of experiment one for $KB_{100}^{p_1 \cup p_2}$

Figure 6.3 shows the number of relevant composed actions and the number of relevant individual actions for each consulted file in partition $p3$. Each file has been consulted against the knowledge base trained with partitions $p_1 \cup p_2$, consisting of 100 samples (KB_{100}^{p1+p2}). A correct composed action, means that one of the suggestions proposed by the tool contains all the individual actions (replacements) that are required to solve the failure detected by the consulted .AR file. From the figure, we can see that this is the case only for 2 out of the 50 files in p_3. In the analysis made in figure 6.2 we showed that only 30% of the files have the possibility to be correctly classified if the set of samples is partitioned into training and validation sets [4] and that only 15 composed corrective actions are shared by more than one file. Thus assuming a "lucky" partition where for each predictable action, we have one sample on the training set, and one sample in the validation set, at most 22 files could have a correct answer. And this would only be the case if the model is able to perfectly learn with one sample only, and if two samples having the same corrective action have the exact same features. As we have seen, this is not the case (different samples may have the same corrective action, and *vice versa*). This shows the low rate of full correct answers expected. Nevertheless, in the awareness of these figures, the objective of these experiments is to show if these estimations really hold, and if the model can provide valuable suggestions even under these circumstances.

[4] Most of the files (70%) are the only representatives for their corresponding composed corrective action.

To this purpose, we also consider partially correct suggestions, or correct individual answers. These are given by the individual replacements in any of the suggestions, that are inline with the expected results. From figure 6.3 it can be can seen that the number of partial suggestions is much higher than considering fully composed corrective actions only. The model has given a partial answer for 14 out of the 50 files. This partial answer can either be a single replacement or up to five replacements. This shows that if partial answers are considered, more relevant information can be provided. This makes sense in the context of suggested corrective actions, which do not intend to impose a repair, but to give hints on its resolution.

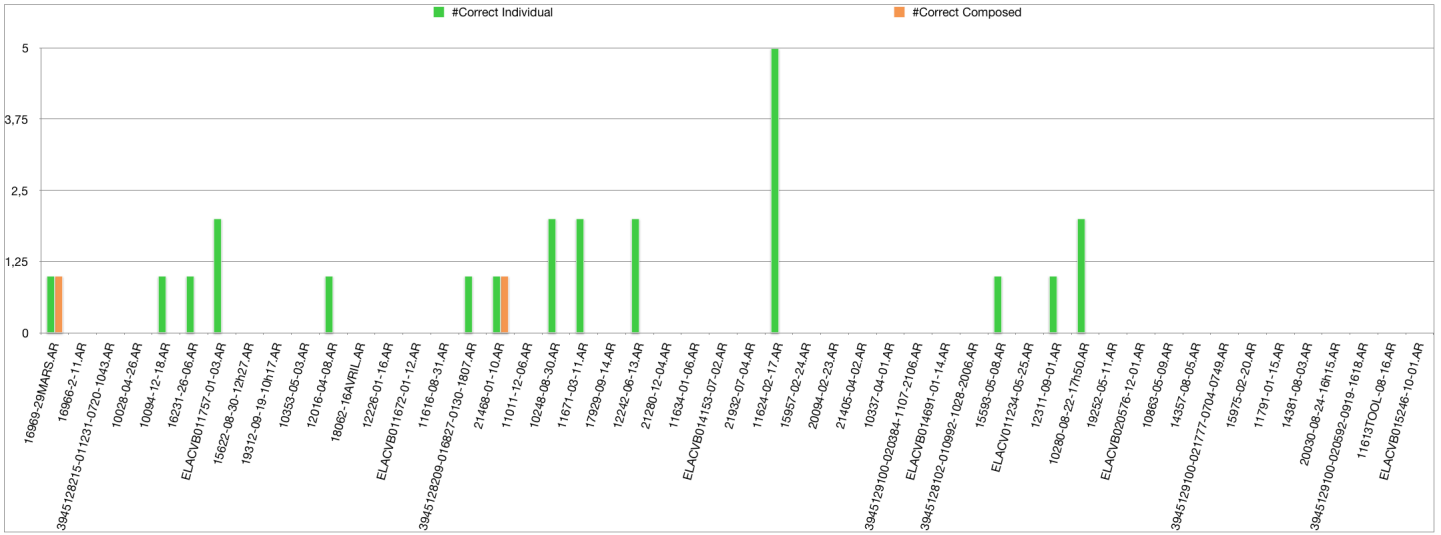


Figure 6.3: The suggestions for the 50 files (x-axis) in partition $p3$ when consulting the knowledge base KB_{100}^{p1+p2}. In the figure the y-axis shows the correct atomic actions (green) and the correct composed actions (orange).

Figure 6.4 is built over the same graphic of figure 6.3 but adds the total number of suggestions. For each file consulted, we first show the total number of suggestions (corrective actions) found by the tool, and next to it we show the relevant composed and individual actions suggested. A few files have associated a large number of suggestions, thus the figure is limited to show only 25 files. Nevertheless, on top of the total number of suggestions, figure 6.4 shows the quantity.

Intuitively, if the suggestions are too many (more than 20) it is possible that among them we find a correct or a partially correct answer, but to determine which is the correct one the technician would have to test all of them. Thus a reduced number of suggestions is acceptable and desirable, but more than 20 suggestions is considered useless.

There are two main results to note on this figure: the composed corrective actions and the individual actions.

With respect to the composed corrective actions, in one case 5 suggestions were given and in the other case 28, thus we consider the the first case is relevant, and the second case needs further refinement.

Regarding the individual corrective actions, 8 out the 14 correct partial suggestions have 20 or more suggestions, and thus only 6 of them are considered relevant.

Although these numbers are relatively low, one has to bear in mind the small amount of total samples, the small amount of shared actions between the files, and the fact that the same type of file can be solved by different corrective actions. The figure shows that the correct and partial suggestions can be found by the approach, and that some of the suggestions may require further refinement, to make them relevant.

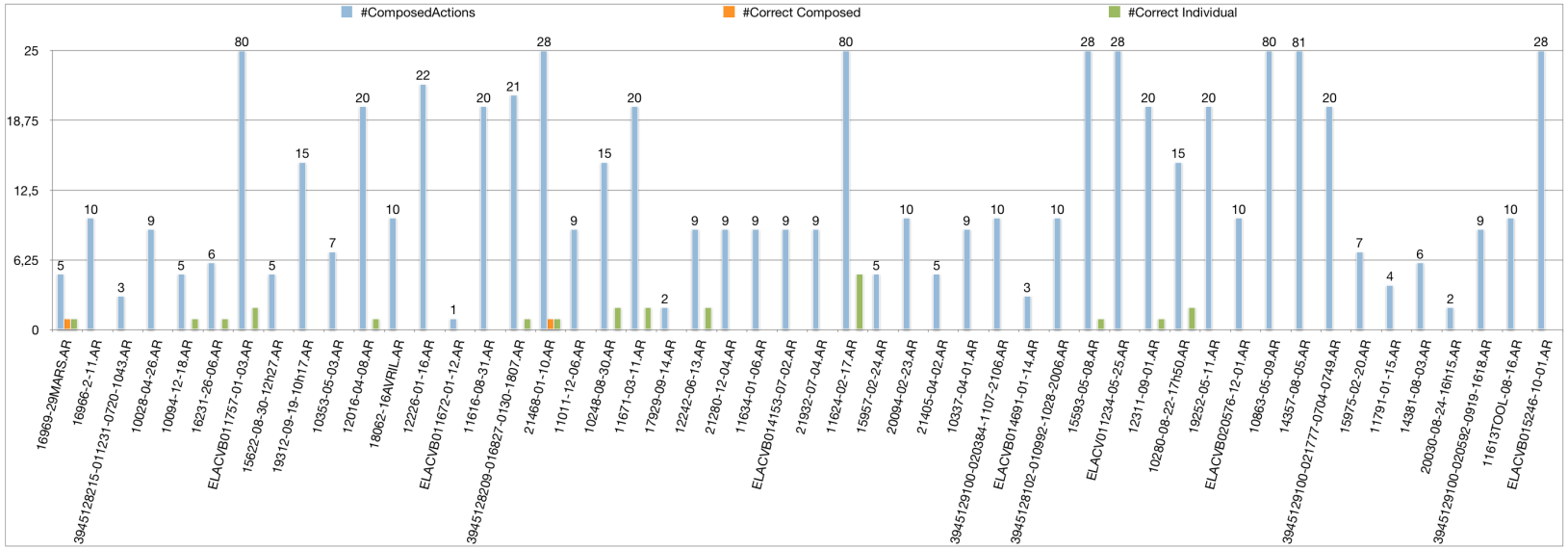


Figure 6.4: The suggestions for the 50 files (x-axis) in partition $p3$ when consulting the knowledge base $KB_{100}^{p1 \cup p2}$. In the figure, for each file are shown : the composed suggested actions (light blue) vs. the correct composed actions (orange) vs. the correct atomic actions (green). For visibility, a cut on 25 suggested actions limits the y-axis.

Figure 6.5 shows the number of files and number of suggestions (composed corrective actions) associated to each file consulted. This aims to establish how general or how specific are the signatures found.

The analysis made in Section 6.1.2 and figure 6.2 show that the types of actions associated to each test (.AR file) are very different. Most of the files (70%) have associated a unique corrective action. Thus, intuitively, the number of files and the number of suggestions provide a measure on how general the signatures are. We would expect that a very general signature, will capture a large number of files and that will have a large number of corrective actions associated, whereas more specific signatures will show these numbers reduced.

From figure 6.2 it can be seen that 10 out of the 50 consultations provided more than 20 suggestions, thus the remaining 80% of the files had assigned an acceptable number of suggestions. We can also see that for 31 files the suggestions are 10 or less. Thus approximately (62%) of the signatures found, are specific enough.

In figure 6.2 it can be seen that 4 consultations found very general signatures, with 96 or even 98 files associated (out of 100). Unequivocally these signatures are too general, in the sense that they do not properly separate a group of .AR files from the rest. This is as well reflected in the high number of suggestions for these consultations.

Note however, that the total files associated to each signature found in figure 6.5 has to be interpreted carefully. The fact that a large number of files is associated to a signature, does not necessarily imply the signature is too general. It could be the case that we count with many samples which present the exact same results, and if a proper signature is discovered for these files, all of them are expected to be associated to the signature. The relation between a large number of files/actions and the specificity of the signatures, assumed in this experiment, comes from the previous analysis made in Section 6.1.2 from which we would expect a large number of files in general signatures.

In conclusion, we can see that with respect to the specificity of the signatures, the results of the experiment provide desirable signatures.

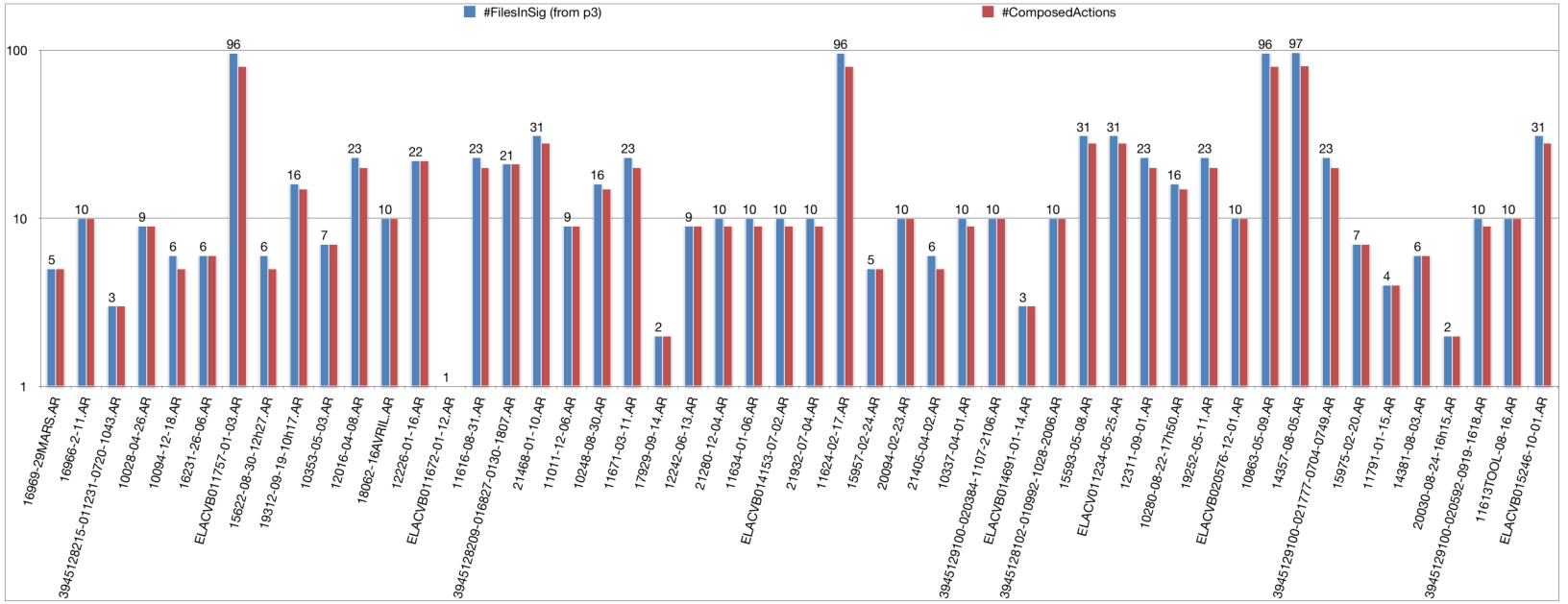


Figure 6.5: When each of the 50 files in partition $p3$ were consulted, the knowledge base $KB_{100}^{p1 \cup p2}$ selected the best signature for them. For each .AR file consulted (x-axis) the figure shows the total number of files that belong to the same signature (y-axis), and the total number of suggestions associated (y-axis). The scale is logarithmic to represent those consulted files whose signature captures a very large number of files/composed actions.

Figure 6.6 is divided in two parts. The top section shows the precision, recall and f-measure for the partial suggestions (individual actions) whereas the bottom section shows these results for the composed actions.

The 50 files consulted (belonging to partition p_3) were assigned to 25 signatures by the knowledge base KB_{100}^{p1+p2}. Figure 6.6 shows the recall, precision and f-measure for each one of these signatures. As we have seen in figure 6.3, 14 out of the 50 files were given a partially correct suggestion. These 14 files are captured by 9 signatures. This is shown in the top part of figure 6.6, where it can be seen that 4 of those signatures (2,4,5 and 13) have a precision of 1, meaning that they capture only .AR files for which the signature has the correct partial answer associated. We can also see that only one signature (19) has a precision below 0.5. Thus most of the signatures have good precision. The recall of the signatures is not so good, but this has to be properly understood. We can see the recall varies from 0.1 to 0.5, and that it does not get beyond these values. It is expected that partial answers are shared by several different signatures. This implies that different files, share some partial corrective actions. Because the files are different, they should not belong to the same signature. Nevertheless, given that they share some individual replacements, they are considered as false negatives: a false negative is a file for which a partial suggestion can be given, but it does not belong to the signature. This explains the low recall, and shows that the results are inline with the expected behaviour of the system and the analyzed data. In this setting, the most relevant metric is the precision, which is high. Since the recall is low, and the f-measure provides the relation of the precision and recall, the f-measure is also expected to be low.

On the bottom section of figure 6.6 , we see the same analysis, but this time for composed corrective actions (full suggestions). As said before, only 2 .AR files got relevant full suggestions. This is represented by signatures 2 and 24. We have also seen that only one of them is relevant (6), since the other one (24) is too general (it captures 31 files and 28 suggestions).

When considering composed corrective actions, the recall is more relevant (although still there could be files with different results and exactly the same correct actions). We can see that the score of signature 6 is perfect, since out of the 50 files in p_3 it captures only 1 file, and no other file in p_3 is seen as a false negative (no other file is expected to be captured by this signature). For signature 24 it is the contrary, the precision is low (0.25) since it captures 4 files, and 3 of them are not provided a relevant suggestion. The recall is even lower (0.14), meaning that other files (6) were expected to be captured by the signature. Both values are reflected in the low value of the f-measure. In short, the explanation for these low scores is that the signature is too general.

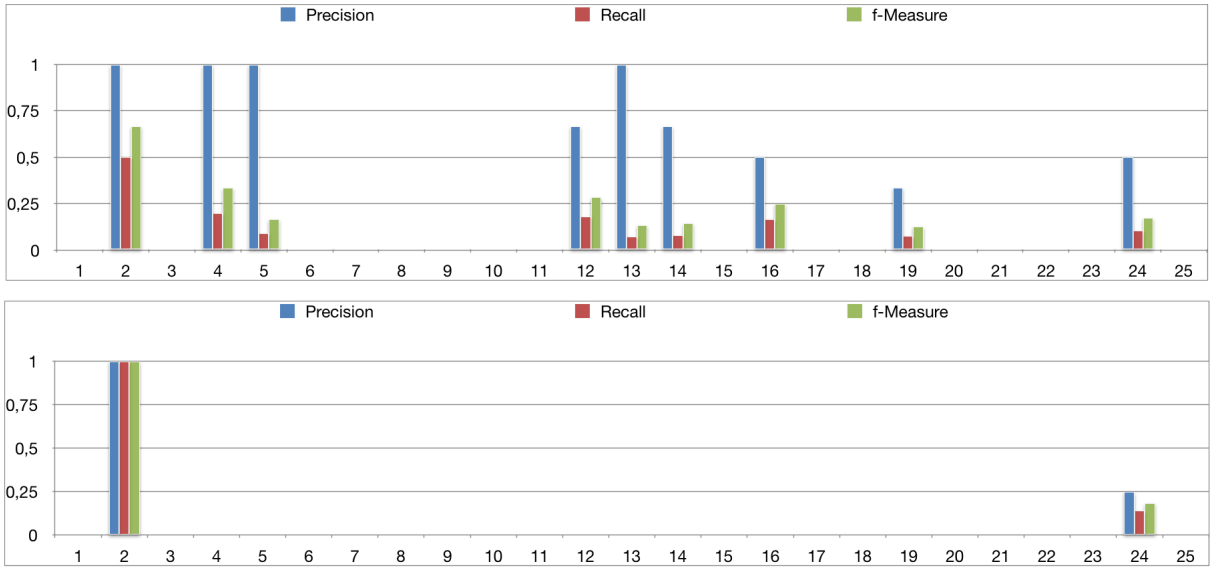


Figure 6.6: The 50 files in partition $p3$ were classified under 25 signatures when consulting the knowledge base $KB_{100}^{p1 \cup p2}$. In the figure, for each signature (x-axis) we show the precision, the recall and the f-measure. On the top of he figure the results for individual suggestions, and on the bottom the results for composed suggestions.

Macro and Micro Precision and Recall

Let us now provide the analysis of the micro and macro precision and recall from figure 6.7. The low average micro precision is explained by the fact that the true positives are always a few (1 to 4) as opposed to several cases where the false positives can be many times more (8 or 9). Since the micro average precision puts all these results together, without regard of the signature to which they belong, the numbers of false positives easily override the number of true positives. This is similar for the micro average recall.

On the other hand, the macro average precision and recall, take into account the signatures where the results are obtained from. And thus a 'bad' performance of a single signature, will not easily override the good performance of other signatures. From figure 6.7 it can be seen that with respect to the macro measures, KB_{100}^{p1+p2} has a good precision performance for both, individual and composed actions, and it has an acceptable recall, for composed actions.

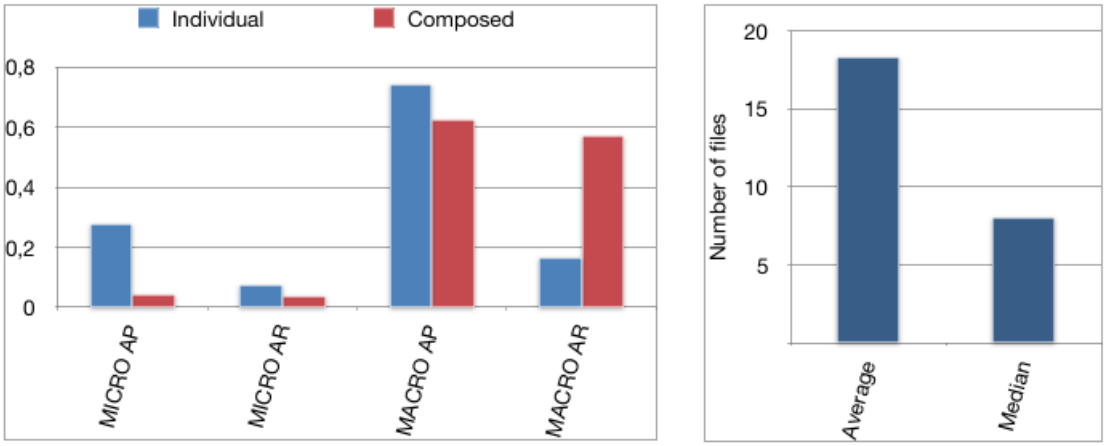


Figure 6.7: Evaluation of the relevance of the suggested actions. On the left: aggregated results of macro and micro recall, and macro and micro precision. On the right: the specificity of the discovered failure signatures, given by the average and median files for all 50 consultations.

The experiments so far, show the relevance of the responses with respect to one knowledge base KB_{100}^{p1+p2}, where we can see the relation between the specificity of a signature and its score, as well as the usefulness of providing partial answers. The experiments also show that the results are inline with the initial analysis of the spare data and its high heterogeneity. Nevertheless, the selection of the partitions can greatly influence the results obtained.

To reduce the bias of selecting one of the partitions (p_3) as validation data instead of any of the other two (p_1, p_2) we have made the same experiments with the three versions of the knowledge base. These are included in the Appendix B.

We next present the aggregated analysis of the results of the three versions available.

Aggregated results of experiment one for $KB_{100}^{p_1 \cup p_2}$, $KB_{100}^{p_2 \cup p_3}$ and $KB_{100}^{p_1 \cup p_3}$

The results presented so far concern only one of the three versions of the knowledge base. The objective of cross-validation is to make multiple experiments in similar conditions with different partitions of the data set, to estimate how and if these results vary.

In experiment one, two main aspects of the trained knowledge bases are evaluated: the relevance of the suggestions and their specificity. For the relevance, we have provided an analysis of the number of relevant signatures found, and for each of these, the precision, recall and f-measure were given. Finally, the average macro and micro precision for the entire knowledge base was calculated and explained. The next step is to obtain the same measurements for all three versions of the knowledge base, and compare them.

Figure 6.8 shows, on the top the values of the Micro/Macro Average Precision and the Micro/Macro Average Recall of the three KB versions. On the bottom, the figure shows a graphic of such comparison. Each indicator is given for the individual vs. the composed actions. From figure 6.8 it can be seen that the micro average precision and the micro average recall are smaller for both types of suggestions (individual and composed) than their macro versions.

From figure 6.8 it can be seen that in all three versions, the precision is in general good (>0.5), and is always better when considering individual actions. A better precision implies less false positives. The graphic shows that when considering individual actions most of the files captured by the signature are indeed (partially) related to the signature's corrective action. In contrast, many files that share some individual replacements with those proposed by the signature, are not captured by the signature. This means that they are considered as false negatives, and thus the recall is lower than the precision.

If we consider only full suggestions (composed actions) as relevant answers, these figures change. In all three cases, the recall is higher for composed actions, meaning that most of the files for which the suggestion is relevant are indeed captured by the signature. This translates in a low rate of false negatives, increasing the recall. The precision for composed actions is not as high as the precision for individual actions, but the macro results are, in all three cases, much better than the micro average ones.

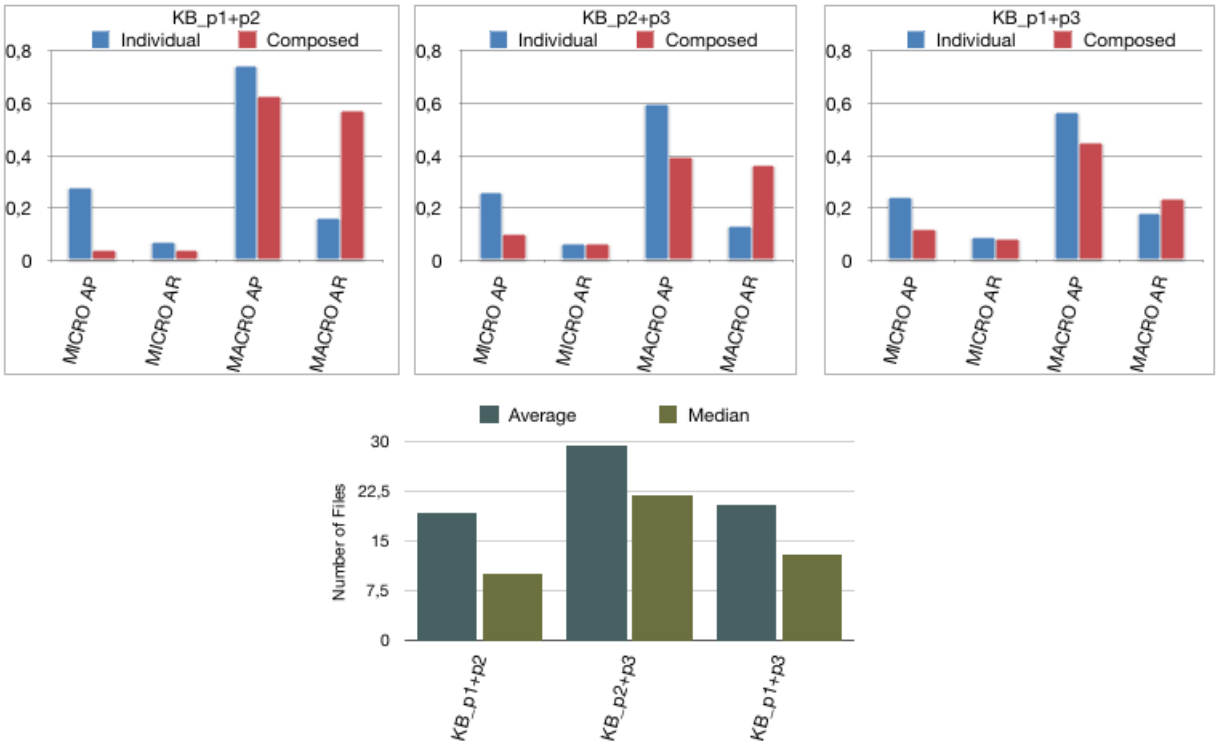


Figure 6.8: Evaluation of the relevance of the suggested actions, showing aggregated results (through macro and micro recall, and macro and micro precision) and the specificity of the discovered failure signatures, given by the average and median files for all 50 consultations.

As the final results for this experiment, figure 6.8 shows the aggregated results for the specificity of the signatures. We have seen that given the high disparities between the files in the data set, we can rely on the number of files that belong to a signature as a measure on how general/specific the signature is. Figure 6.8 shows the average and median number of files captured by each signature, for all 50 consultations made to each KB. The average value can be influenced by non frequent but very high results. In figure 6.8 we can see that the average files is above 20 files per consultation, which is undesirable. On the other hand, the median provides the middle value over which half of the measurements are encountered. This is a better estimator for our case (since it is less sensitive to extreme and rare values) which shows that the median is way below 15 files that belong to each signature found. These results are promising, since we have seen the importance and the influence of the specificity of the signatures, and where having more specific signatures is desirable.

Experiment 2 - Evolution of the KB

This second set of experiments, has the objective of evaluating whether there exists an improvement in the quality of the knowledge base as more information is presented to it, and to estimate the evolution of this improvement.

To evaluate the quality of the knowledge base we use two criteria: first, precision and recall based on the ratio of positive and negative samples of the signatures found, and second, the average and the median related to the specificity of the signatures.

Three versions of the knowledge base are constructed: KB_{25}, KB_{50} and KB_{100} using training sets of size 25, 50 and 100, respectively, where each knowledge base doubles the size of the previous one.

The knowledge base KB_{100} is selected as the best performing knowledge base, from the three versions obtained in experiment one (cross-validation). The best performing version is $KB_{100}^{p_1 \cup p_2}$. Since our goal is to assess if an evolution takes place, it does not make sense to take any of the other two versions in experiment one, as they underperform the former. This selection also provides us with the evaluation set p_3.

The knowledge base KB_{50} is obtained by randomly selecting 50 files from KB_{100}, ensuring that no file in KB_{50} overlaps p_3. We do not cross-validate these sub-versions, because computing many versions of the knowledge base is expensive (in time and computer resources). Thus this second experiment is not intended to provide an exhaustive test of all the possible versions of the knowledge base, but to provide a trend, for which random selection of sub sets of KB_{100} is fair.

The knowledge base KB_{25} is also obtained by randomly selecting 25 files from KB_{100}.

An additional motivation for selecting a knowledge base of size 25, comes from the fact that the first version of the prototype had a limit of around 25 files for the training and consulting processes (depending on the files presented to the KB). Above this number, the reasoner would timeout. Thus, this is the bigger knowledge base we could handle before improving the code. We are interested in evaluating if improvements on the quality of this KB can be achieved.

The three knowledge bases are consulted using partition p_3, which is the validation set for KB_{100}. That is, all three KBs are consulted the same files.

Results - Evolution of the relevance of the knowledge base

To detail the behaviour of the KB_{100}, KB_{50} and KB_{25} we first show the recall, precision and f-measure for each relevant signature in each knowledge base. This comparison is shown in figure 6.9. Then, we compare the macro and micro precision and recall, along with the average number of files, for each version of the KB. This second comparison is shown in figure 6.10.

We start the analysis of the results in figure 6.9 by the number of signatures used for the 50 consultations to each KB. From all the signatures each KB has discovered in the training phase, only some one of them are selected when consulting a file, ie. the most specific. We can see that for the 50 files consulted in each case, KB_{25} has only selected 10 signatures, compared to 21 signatures selected by KB_{50} and 25 signatures selected by KB_{100}. Evidently, the more files used to train a KB, the more signatures it has available. From these, we would expect that those found by smaller training sets (KB_{25}) are more general than those found by larger training sets (KB_{50}, KB_{100}).

If we focus on the individual actions (top graphic of each pair of results in figure 6.9) we can see that for the 50 files, KB_{25} has been able to classify them in 5 signatures, whereas KB_{50} had only selected 4 signatures. Thus it would seem that KB_{25} can provide more relevant answers than KB_{50}. Even though the results from KB_{25} are very good for its size, its has to be noticed that from the 10 files that are given a partial suggestion, for half of them (5 files) the signatures are too general and provide all possible suggestions in KB_{25}. This is why "so many" files are provided a "correct" partial suggestion. This can be partially seen by the precision, which would be much lower for the "irrelevant" signatures, if the size of KB_{25} would be bigger. In such scenario more undesired files would belong these signatures, increasing the number of false positives. The generality of the signatures in each KB will be shown in more detailed in the evaluation in figure 6.10.

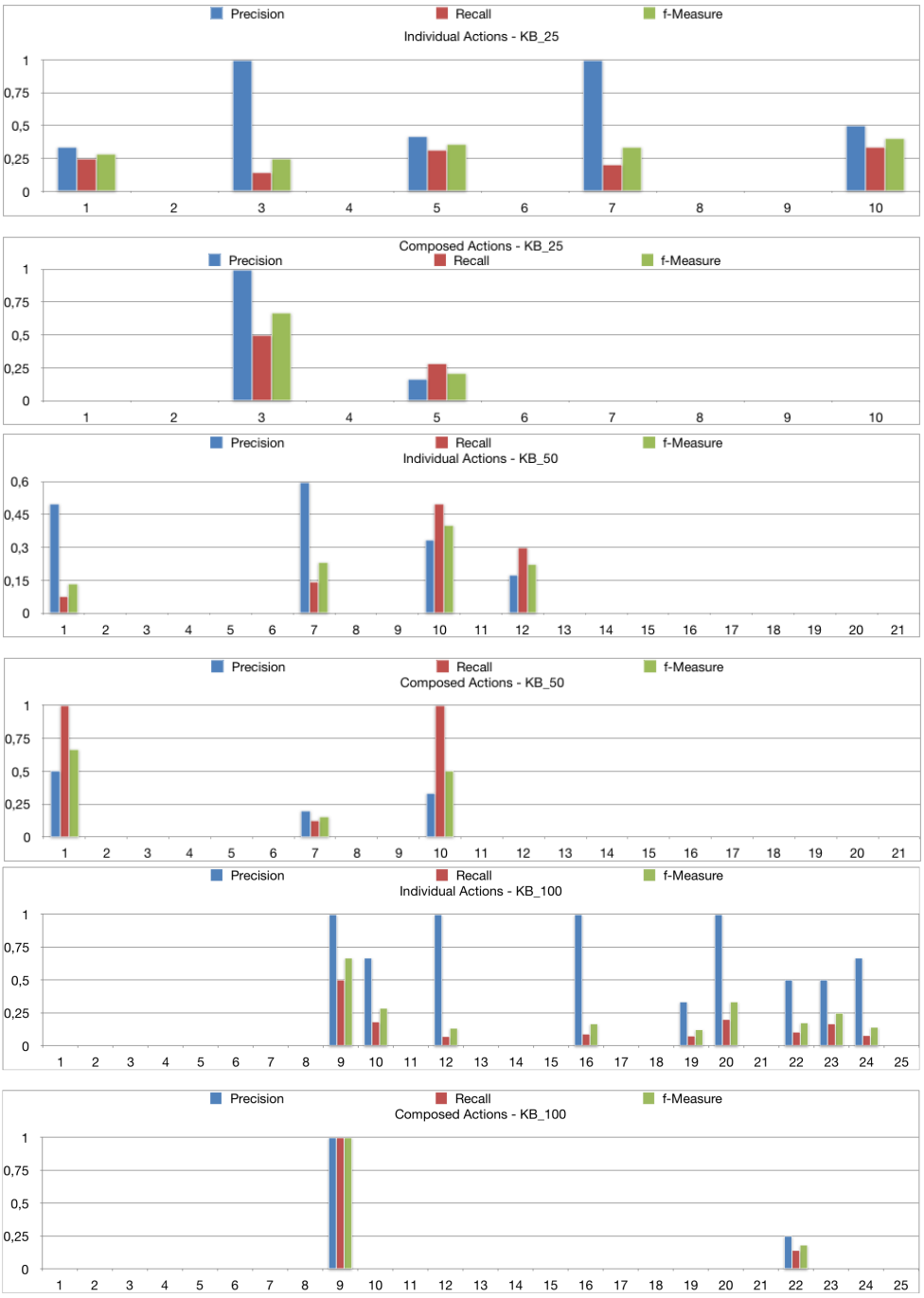


Figure 6.9: From top to bottom, the comparison of the individual and composed actions for three versions of the KB: KB_{25}, KB_{50} and KB_{100}. Each pair of figures shows the precision, recall and f-measure for the individual (top) and composed (bottom) actions for a version of the knowledge base. In all figures the x-axis shows the signatures selected in the 50 consultations.

The relation between the precision and the number of signatures selected is more evident in KB_{100} where it can be seen that most of the partial suggestions have a high precision, making them relevant. With respect to the full suggestions, in all cases their scores remain low, this is explained by the analysis made in figure 6.2 where we have seen that very few files in the full set of samples share composed corrective actions. This is also true for the individual actions, but they are more frequently shared than full corrective actions, which is reflected in all three KBs in that more relevant suggestions are made at this level. Finally, one should notice that for several files (all those signatures with no score) no relevant suggestions were found. Regarding this last point, the figures show a tendency of having each time more relevant suggestions as the KB training sets increase.

Results - Evolution of the specificity of the knowledge base

The aggregated results of the evolution on the relevance of the suggestions are shown in figure 6.10. Regarding the aggregated precision and recall, we have seen that the most indicative measure is the macro average precision. We can see in all cases that this measure performs well, above 0.6 for KB_{25} and KB_{100}, and that this value is always higher for individual actions, highlighting the relevance of providing partial suggestions, and the associated low rate of false positives. On the other hand, the macro average recall for composed actions performs well in all cases, and is always higher that its individual actions counterpart. We have seen that few files share fully composed actions, a higher macro average recall indicates that when a composed action is detected, most of the files that share this action are captured by the selected signature.

Finally, on the right of each figure an analysis of the average and median number of files for each KB is provided. These measurements are slightly different than those shown in cross-validation, where the average and median were calculated with respect to the files from each KB. In the evolution analysis, each KB has a different set of files and of different sizes, nevertheless, all of the KB's are consulted the same set of files, i.e. p_3. Thus the average and median are calculated with respect to this set, to provide a fair estimation.

From figure 6.10 it can be seen that both, the average and median files associated to each version of the knowledge base decrease as the size of the training set grows. In average KB_{25} associated 4.2 files to each selected signature, whereas half of the signatures selected by KB_{100} have associated exactly one file. This shows that indeed, the more information that is presented to a knowledge base, the more specific its signatures can be.

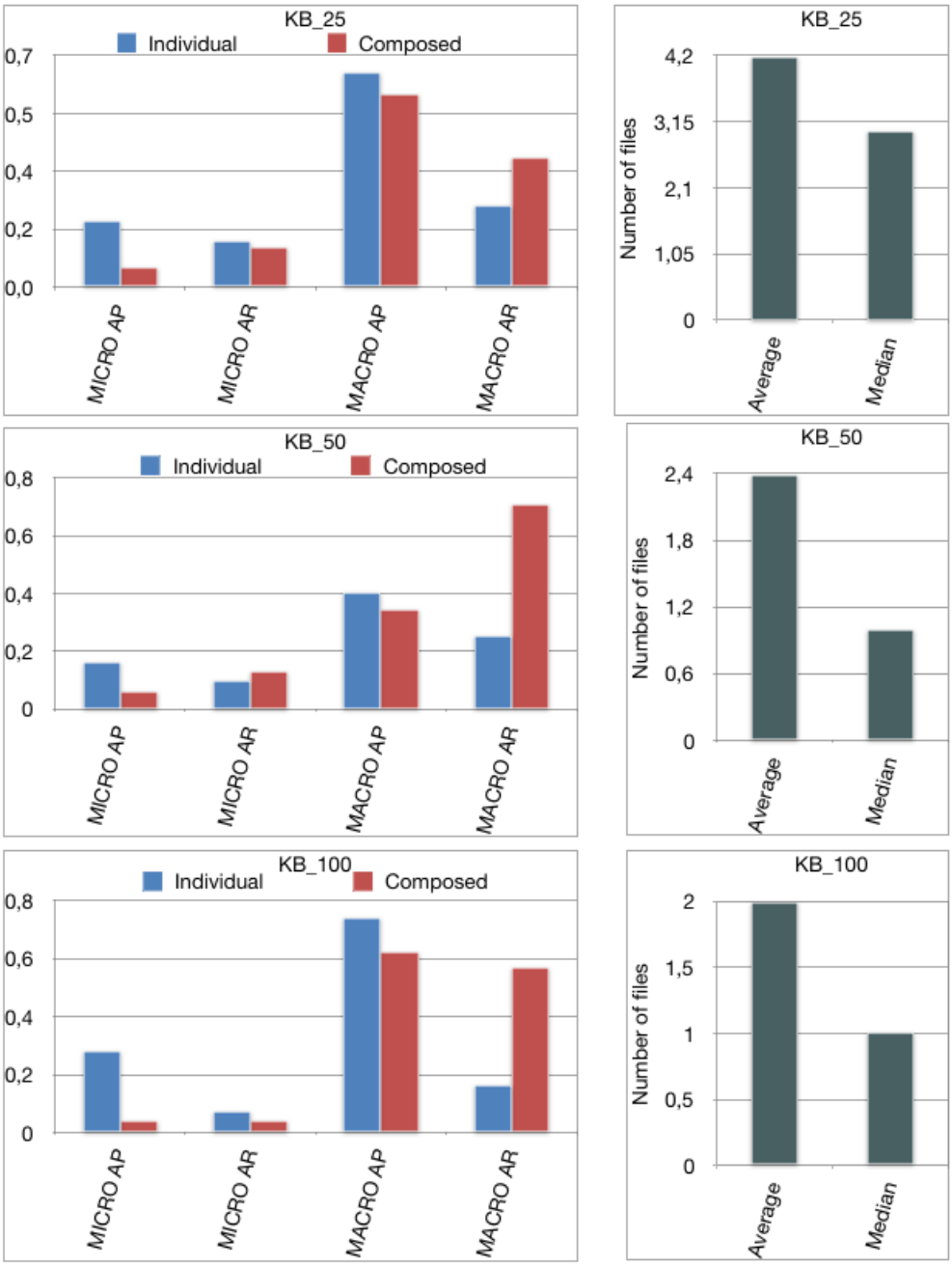


Figure 6.10: Experimentation on the evolution of the KB. The figures show on the left, the Micro Average Precision, the Micro Average Recall, the Macro Average Precision and the Macro Average Recall. On the right, the Average and Median number of files, out of the 50 files consulted. Both types of evaluation are shown for each version KB_{25}, KB_{50} and KB_{100} of the knowledge base.

6.2 Evaluation of the Response Times and Scalability

The second type of evaluation concerns efficiency. We evaluate V1 vs V2 of the prototype to determine the impact of the modifications in the efficiency. The limit of the first version (V1) regarding the number of files that could be used in a single KB was 25 files. The changes made in the second version (V2) allowed to handle all the files that in the sample dataset (150 files). These experiments provide an insight on the efficiency of the implementation and the scalability of the approach, which is also relevant for systems with similar capabilities.

Objective

The main objective of our work is to provide solid foundations on the techniques and algorithms used, and then to provide a prototype that serves as a proof of concept. The prototype is meant to validate the hypothesis, and help identify the key elements that allow or prevent the approach from being implemented. The objective in this experiment is to determine such limits. To this end we evaluate the consulting time between the first and the second version of the prototype.

In the consulting phase the technician uploads an .AR file and the system proposes the suggested corrective actions. It is important to determine the response time, for several reasons: first, there is an evident limit: it should not take more than the time the technician takes to solve the failure (the technician can take a up to a few days to determine the failure). Second, consulting a larger KB should take more time, therefore there is a limit on the amount of information that can be handled in an acceptable time, and even a limit where no suggestions can be given, thus the response times allow us to estimate these limits. Finally, this provides us and the end user with the nominal waiting times for a response. The user will expect a response in a given interval of time considered normal.

The learning phase is by far the most expensive task, computationally speaking. In the development of the prototype V1, the first tests for learning took several days (2-6), and the larger the KB became the more time this task required. In fact this was the main motivation to implement a second version: to overcome the limits of the KB that can be used.

The first version allowed us to identify key limits to the feasibility of the approach. These are related to: the number of calls made to the reasoner, the split of the knowledge base, and avoiding equivalent definitions to be added to the knowledge base. Version two of the prototype has split the A-Box of the knowledge base to avoid considering the whole information at the same time, and

opens the possibility to parallelize consulting the knowledge base. We have also included check on already learned equivalent signatures or parts of signatures.

These changes have allowed us to learn from more files, but this process is still slow (we exhaustively analyze 100 files in approximately 7 days). Consulting the knowledge base, is a sub process of learning, and is the main functionality of the system (learning can be made offline). Thus this experiment shows the consulting times for p_3 with respect to the above mentioned versions of the prototype.

The limits on the consulting times, and ultimately the size of the knowledge base that can be handled impacts the feasibility of a large scale implementation, and enable us to verify if the changes made have had the desired effect.

6.2.1 Key Performance Indicators

To evaluate the efficiency of the implementation we evaluate the consulting time, detailed in figure 6.11

Criteria	Description
Response time (Consult Ontology)	The prototype provides two main functionalities: consulting and learning. In the consult phase the user uploads an .AR file and receives the suggestions from the system. This KPI shows the time needed for this task, and does not take into account the time needed to upload the file, nor to send the response to the end user, but the time the knowledge base takes to provide the suggestions for a given .AR file.

Figure 6.11: Key Performance Indicator for efficiency.

The consulting response time, is the more visible feature of the system and assessing the time it takes is important for the implementation and for the expectations of the end user. Consulting the ontology is also a sub-task of the learning phase, since each learning cycle requires querying the knowledge base. Thus improvements in this process improve as well the learning time.

The bigger the knowledge base, the more time consulting it should take. We would like to evaluate or approximate the limits of this task in the current implementation. Indeed, the first version of the prototype could only be trained with up to 25 files, and consulting the knowledge base was in the order of minutes. This indicator also allows us to assess if the changes made to the first version of the prototype do impact the performance of the system.

6.2.2 Experiments

Experiment 3

During the implementation of the first version, which is the more limited, we evidenced that in both: the consulting phase and the feedback phase, the most time was invested in calls to the reasoner. To overcome these limitations, the first version was refined to reduce the reasoner calls and avoid equivalent signature definitions. This allowed to handle up to 42 files in the learning phase, but this limit could not be passed using a single knowledge base. This last implementation of version 1 is used in this experiment. This knowledge base in denoted by KB_{42}.

The second version of the prototype, has divided the A-Box in several consistent sub-sets, so that each call to the reasoner has to work with a reduced size of information. When using all the information at the same time (all data in a single A-Box) in version one, we saw that consulting the KB could take more than 20 minutes, and in some extreme cases answers wouldn't be given for several hours. The second version further reduces the calls to the reasoner, splits the A-Box and is designed to be used in a server environment. To evaluate the impact of these changes we show the consult time for each one of the files in p_3 against KB_{42} and KB_{50} and KB_{100}.

Figure 6.12 gathers the consult times of p_3 against KB_{42}, KB_{50} and KB_{100}, where KB_{42} is obtained and consulted with the last implementation of version one of the prototype, and KB_{50} and KB_{100} are obtained and consulted using version two of the prototype. The background surface represents the consulting times for KB_{42}, where it is evident that these times are several times longer than those obtained by KB_{50} and KB_{100}. The times on the y-axis are shown in milliseconds, the figure has a limit of 25 000 milliseconds, or 25 seconds. Not only a more than 50% of these files (29) are above this limit, but 13 files had a consulting time longer than one minute (60000 ms) of which 9 had a timeout, where the timeout was set to five minutes. In contrast 90% of the files consulted for KB_{100} has a consult time below 5 seconds, with only 5 files above this mark, where the highest time was 21 seconds. Additionally, one has to bear in mind that these results are for a knowledge base twice the size of KB_{42}. If we turn our analysis to KB_{50} (the smaller surface) we can see that the times for KB_{100} are almost cut by half. The same files that take more time in KB_{100} take more time as well in KB_{50}.

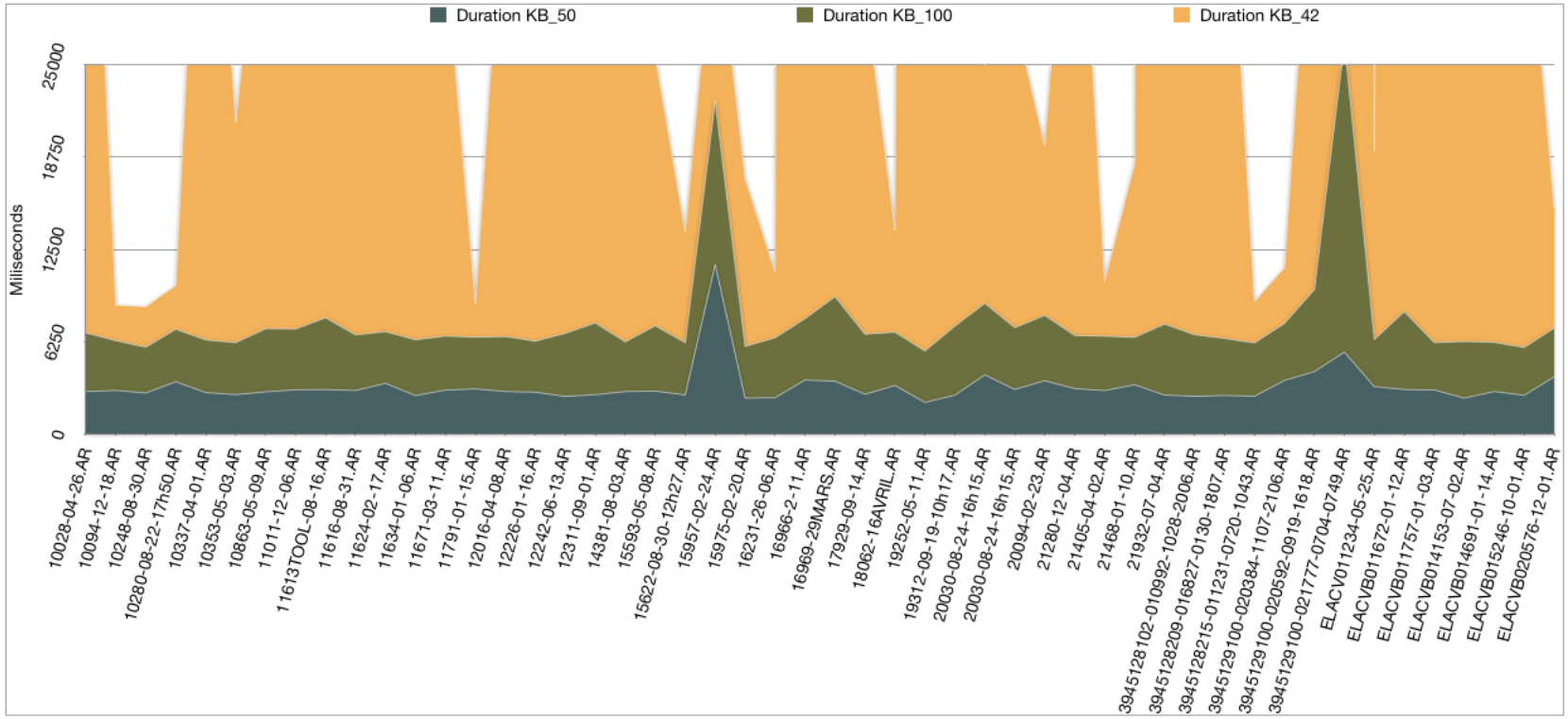


Figure 6.12: The consulting times for the 50 files (x-axis) in partition $p3$. In the figure three surfaces are plotted. In the background the consult times for KB_{42} (version one), next the consult times for KB_{100} and the front most surface shows the consult times for KB_{50}. Times are given in milliseconds.

In this section we have seen the relation between the consulting times, the size of the knowledge bases and the improvements on the implementation of the prototype.

These elements are important to estimate the feasibility of the implementation of the approach. It is not sufficient to establish whether or not the suggestions can be relevant, but also real processing limits to the proposed solution. Special attention should be put in the cases where the reasoner timed out, and in the ability to split the A-Box.

Thanks to these experiments, we have seen that the identified improvements indeed greatly upgrade the performance of the prototype, in that they reduce the consulting time and allow to handle bigger knowledge bases. Files that time out, do not only depend on the size of the consulting knowledge base but as well on the files complexity, and the equivalences found among the signatures in the KB, which make the reasoner task harder. This is evidenced by high reduction of timeouts in the second version. Finally the results show that the limits on the size of the KB can be further pushed, and the split of the A-Box shows that implementing parallel processing is worth.

6.3 Comparison with similar Approaches

In this section we compare our tool to a state-of-the-art concept learning tool: DL-learner. We use both tools to learn concepts from the same training set, and evaluate the number of suggestions obtained.

Objective

We want to evaluate our approach versus a state of the art concept learning tool. Our objective is to obtain the amount of suggestions that each tool provides, to determine which one best minimizes the number of suggestions. Additionally, as we have imposed a limit on how specific the concepts returned by our tool (TAMO) should be, this experiment also allows us to asses if this limit is too low to properly classify the set of samples.

6.3.1 Key Performance Indicators

The indicator for the comparison is the number of suggestions proposed by each tool. We want to asses if our approach is more successful in minimizing the proposed suggestions, than those obtained by DL-Learner.

6.3.2 Methodology

For this experiment we have used DL-learner[5], since it is ont of the most complete and up-to-date tool for concept learning.

DL-Learner is a machine learning tool, designed for supervised learning over DL-ontologies. It supports an extensive number of parameters, algorithms, metrics and languages [Lehmann and Hitzler, 2010]. DL-Learner is designed for supervised machine learning. Thus we argue that it does not exactly suits our case. Nevertheless, the way it constructs concepts is similar to ours. Thus we would like to compare how relevant are the concepts found by DL-Learner, compared to our approach. To enable this comparison, we can regard our training set as partially-labeled data, and DL-Learner can be used to approximate the sets (signatures) we are looking for. DL-Learner provides a set of the best n concepts that describe the samples having a particular corrective action. These concepts depend on the underlying language, the time allowed to run the tool, the type of problem (positives only or positives and negatives), etc. In DL-learner, class expressions that are shorter are preferred.

On the other hand, using our approach in order to find the most specific class for each failure signature, a limit in the length of the discovered concepts has been imposed to be up to 35 subconcepts. The experiment also allows us to determine whether this limit is enough to properly distinguish between the tests. In other words, we consider up to 35 properties common to all tests to form a failure signature, and we evaluate if this suffices to arrange the number of samples we handle.

Experiment 4

For the sake of clarity, in this section we refer to our approach as TAMO, to differentiate our results from those obtained with DL-Learner. For this experiment we have selected a random subset of 25 files out of the total files available (150). For each file, we have obtained the number of corrective actions assigned, both with DL-learner and TAMO. Both DL-learner and TAMO, were trained with a set of 50 files, obtained from cross validation, in Section 6.1.

Results TAMO vs DL-Learner

Since our goal is to minimize the number of suggested actions, we want to evaluate how many actions are proposed by each tool to each file. In Figure 6.13 we show the number of actions returned using the concepts learned by

[5]http://dl-learner.org

DL-Learner and the concepts learned by TAMO, for each of the 25 selected files. Each file may belong to one or more DL-Learner concepts, and therefore it will be associated to all the actions those concepts represent. The concepts that are too general, capture most/all instances. From the figure we can see that most of the concepts from DL-Learner will associate around 20 actions to each file, whereas in our case, most of the files are associated to 3 or less actions. There are also a few cases where we associate more than 30 actions to a file, this is mostly because those files were not related to the set of files we used, to create our classes (learning phase).

The low precision of the DL-Learner concepts, can be explained by the fact that the tests that are solved by the same action might be not only very different from each other, but they might not even share anything in common among them. Given that the underlying language is $\mathcal{ELO}$, no disjunction is allowed (which would help to capture files that are different by a single concept), there is no single representation for all those tests in $\mathcal{ELO}$ and DL-learner returned short but very imprecise concepts.

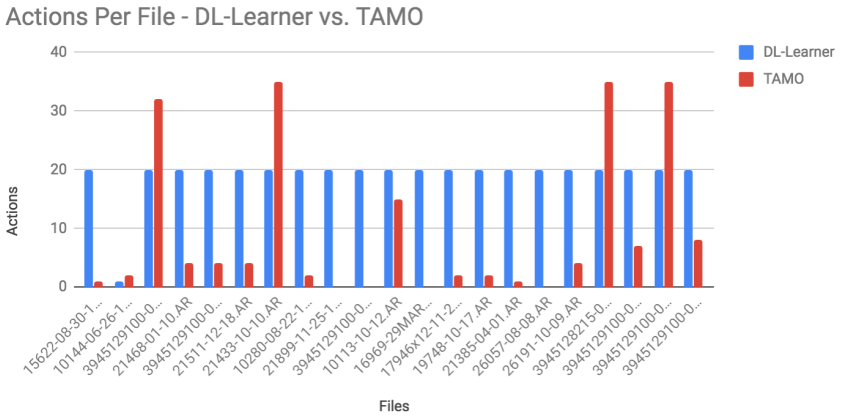


Figure 6.13: Number of actions suggested for each file. DL-Learner vs. TAMO.

6.4 Summary

In this chapter we have presented the evaluation of the approach, aided by the implementation of a prototype.

We have provided an analysis on: the relevance and specificity of the signatures found using cross validation to reduce the bias on the partitions selected. We have seen that relevant answers can be given using our approach, even under the high constraint the data impose: high variability and sparse data. We have also

seen the benefits of considering not only full composed actions as suggestions, but that considering partial suggestions increases the relevant answers we can provide. Out of these experiments, we have obtained a performing knowledge base trained with 100 samples. In a second set of experiments, we tested whether there is an evolution in the knowledge base, as more information is presented to it. By evolution we mean the KB is more fine grained, and that its answers are more precise. We have seen from our experiments with 25, 50 and 100 samples, that this is indeed the tendency. Then, we have proceeded to evaluate the efficiency of the implementation given by the response times. A high-performance system is not the goal of this thesis, but identifying the main challenges and risks for an full implementation are of high interest. Thus we have outlined the weak point we have detected on the several versions of the implementation, provided guidance on how these could be overcome, and evaluated the chosen improvements given by the consultation response times. In a final set of experiments, we have compared the concepts we learn with respect to those learned by a similar tool: DL-Learner. Even though they are designed for different tasks, DL-learner could be used to obtain concepts that approximate the signatures. We have seen that the precision of the concepts obtained by our approach is better adapted to our needs than those obtained using DL-learner. Finally, we have provided the designed survey to obtain the users feedback on the experience of the tool. This last experiment is ongoing at the moment of the writing of this thesis.

Chapter 7

Conclusions and Further Work

In this chapter we summarize the main contents of this thesis and the conclusions regarding the most relevant features. Finally, we present directions for further work.

7.1 Conclusions

We have presented an approach to provide concept definitions in DL for those sets of individuals in an ontology $\mathcal{O}$ that can be distinguished between them. Each one of these sets can be seen as a semantic cluster of individuals, since for each of them there exists a concept expression (at least one) that describes the cluster. In this thesis, each concept describing a set of individuals in $\mathcal{O}$ is called a *situation*. We have provided upper bounds for computing a situation in $\mathcal{O}$ and for computing all situations in $\mathcal{O}$, which are exponential in the size of the concepts. Nevertheless, these problems become polynomial if the size of the concept expressions is bound by a constant. Since each set of individuals can be described by more than one DL concept, we have proposed to prefer the most specific among them. Thus for each representable set of individuals in $\mathcal{O}$ a single concept representing the set is selected. This concept is called the MSR of the set. Once these notions are defined, we have provided a sound and complete algorithm to compute the situations. We have shown how to compute the MSR for an single individual (which corresponds to its most specific concept), how to compute the MSR of a set of instances, and we have provided a strategy to obtain all situations in a set of individuals along with their correspondent MSR's. These algorithms allow us to solve the situation discovery problem.

Our approach for concept discovery can be used to identify "interesting" sets of individuals of a given domain, and provide a meaningful description of the main features shared by the individuals in each such set. The intuition behind is that, if a set of individuals share some properties expressible in DL, a DL definition for such a set of individuals can be provided by solving the situation discovery problem.

In the avionics maintenance domain, the set of individuals to analyze is the set of .AR files resulting from the Test-Bench, and the situations found to describe sub-sets of .AR files serve to approximate failure signatures, thus discovering situations in this specific ontology (TAMO) amounts to discover failure signatures.

Once the signatures are available, we have shown how they can be used by a diagnosis support tool. We have specified the context on which such a tool should be deployed, explaining how the technician currently diagnoses equipment, which are the resources he/she possesses for the maintenance operation, the systems involved, and the format and origin of the data sources. Out of this specification, the requirements of the prototype were established, resulting in two main functions: a) consult the knowledge base to obtain suggested corrective actions, and b) integrate the technicians feedback on the proposed suggestions to enrich the knowledge base. A specification on which results these process should yield, and how the system was implemented was also given.

As the prototype was implemented, we have evidenced limitations in the amount of data it could handle, on the speed of the consultation and learning phases, and constraints on the access and scalability for an industrial implementation. By reducing the calls to the reasoner, avoiding equivalent definitions in the knowledge base and more importantly, splitting the A-Box of the ontology, we have overcome many of these constraints. Besides the modifications made to improve these processes, the prototype was adapted to be deployed under a BigData platform, enhancing the scalability since it enables for future implementation of massive data processing and parallel processing. These improvements have resulted in a second version of the prototype, which was being evaluated at the moment of the redaction of this thesis.

Finally, with the implementation of the approach, we have run a series of experiments that mainly evaluate the relevance of the proposed suggestions and the specificity of the resulting knowledge base. We have additionally provided an evaluation of the performance on the two versions of the prototype, given by the response times when consulting the knowledge base. A final experiment, compares the signatures found using DL-Learner and our approach.

In the evaluation of the relevance of the approach, we have seen that taking into account partial suggestions can significantly increase the relevant proposed

suggestions. Even though for many consulted files, no relevant suggestions were found, due mostly to the sparse data in the sample set, we have seen that the quality of the ontology shows an improvement as more knowledge is analyzed. Moreover, given that the possible suggestions are exponential in the number of failed tests in an .AR file, the evaluation shows that an implementation of such a system is able to provide relevant suggestions, and that these would increase with time.

To conclude, in this thesis we have provided a solution to approximate failure signatures in avionics maintenance, using an ontology based approach. The work in this thesis has a multidisciplinary background, since it is directly related to system diagnosis, knowledge representation and reasoning, machine learning, and description logics, among others. This broad reach is reflected in the stages and results detailed throughout the thesis, where we have passed from the conception, design, implementation, deploying and evaluation of the proposed ideas. This is also a consequence of the nature of the thesis, which takes place under a CIFRE convention, involving both: academia and industry.

In research and innovation there exits several levels of maturity of a project, given by the Technology Readiness Level [Héder, 2017] starting from the very idea in level 1 to the full industrialization of a product, in level 9. The works in this thesis, have lead to an "Initial Gate" approval in TRT, which means that part of the technology developed in TRT has been transferred to another entity, in our case Thales Avionics. Thus the work hereby presented involves levels 1,2,3,4 and 5 of the TRL scale.

Finally, the thesis had lead to several internal (Thales and U-PSud) and external presentations for the work, the development of a screencast (video presentation) of the prototype, the proposal for a patent, and a review of the process involved in maintenance by the experts and technicians involved in the project.

7.2 Further Work

In the short term, the evaluation of the usability and acceptability of the prototype is a main objective.

The tool that implements our approach has an impact on the work the technicians do, and the way they interact with the knowledge base. Their expectations and the acceptability of such a tool play a very important role on estimating the feasibility, benefits and risks of stepping into a industrial implementation.

At the moment of the redaction of this thesis, the last version of the prototype was implemented and a group of ELAC technicians were granted access to it.

The evaluation of the usability and acceptability, is a rich field, and we do not intend to make a full analysis of it in this thesis, nevertheless we have developed, with the help of colleagues in Thales Avionics, a survey designed to test the usability and acceptance of the implemented system.

A sample of the designed survey is shown in Figure 7.1.

As future work we envisage the following directions:

- **More expressive DLs** The current approach is restricted to the $\mathcal{ELO}$ language of DLs. Natural extensions to be considered are to allow the use of disjunction ($\sqcup$) and negation ($\neg$) as constructors in the DL language. One implication of allowing disjunction regards the exponential grow on the number of concepts that have to be explored. Even though this can be limited when the search is guided by the instances, as in our approach, the full implications and considerations remain to be studied. The case of negation is similar, in the sense that to ensure that a negated argument holds, all the cases where it does not hold have to be explored.

- **Parallel Processing** The modifications made in the second version of the prototype, have split the A-Box in several consistent partitions, where each of these A-Boxes is consulted independently by a sequential process and the results aggregated in the end. Since the results of each the consultations are independent from each other, each A-Box can be consulted by a separate process (i.e. parallel processing) and then the results aggregated to provide a single consolidated result, in a Map-Reduce fashion.

- **Distance between the signatures** To the moment, the signatures have no further relation between them than that of subsumption. A distance can be defined, with regard to the specific features considered by each signature, to provide more flexibility on the results. Such a distance would also allow for individuals that do not belong to a signature, to be closely related to the signature, which can improve the suggestions provided.

- **Model Extension** Finally, the model considered in this thesis focuses on the ELAC equipment, on the replacement of components, and on the technician's diagnosis process. The model can be extended to consider more equipment, more maintenance tasks and more actors involved in the maintenance process.

All these aspects, remain as further work.

Retour sur une expérience d'utilité

1 = Pas du tout d'accord
5 = Tout à fait d'accord

French	1	2	3	4	5	English
Vous utilisez le système depuis						**Type de réponse du système (experience 1 semaine / 2 mois)**
moins d'une semaine	x					
moins de deux mois						
deux mois et plus						
Rapidité						
Le temps d'attente pour obtenir une suggestion est trop long.			x			The waiting time to obtain a suggestion is too long.
Pertinence						
La plupart du temps, les suggestions aident à résoudre la panne.			x			Most of the time, the suggestions were related to the cause of the failure.
La plupart du temps, les suggestions sont partiellement correctes.						
Les suggestions avec la confiance la plus haute, son toujours les meilleurs sugestions.			x			When one suggestion was correct, it also had the highest confidence.
Même si aucune réponse n'est correcte, les suggestions aident à résoudre la panne.						Even if no answer is correct, the suggestions are useful to repair the equipment
Impression générale du système (pour une expérience de 1 semaine à 2 mois)						
Ce type de système peut aider à l'efficacité du diagnostic (reduccion de ttemps de diagnose, moins de taches nécessaires).			x			This kind of system can help me reduce the diagnosis time.
Avec un outil de ce type, on peut détecter des cas exceptionnels.			x			With this kind of tools we can detect exceptional cases.
Le nombre de suggestions de réparation est trop grand.			x			The number of suggested repair action is too big.
Évaluation qualitative						
Évolution du système (pour une expérience > 2 mois)						**Evolution du système (experience 2 mois)**
Le nombre de bonnes réponses augmente avec le temps.			x			The correctnes of the answers has been imrpoved with time/usage
Le système devient plus lent avec le temps.			x			The system has become slower over time
Le nombre de suggestions diminue avec le temps.			x			The number of suggestions has decreased with time. (more accurate)
Je pense que le système s'améliore avec le temps.			x			I think that the system is getting better with time.
Questions ouvertes						
Y a-t-il des suggestions plus pertinentes que d'autres? Pourquoi?						Are there suggestions more useful than others? why?
Y a-t-il des suggestions qui ne sont pas du tout pertinentes? Pourquoi?						Are there suggestions that are not useful at all? why?

Figure 7.1: The survey designed to obtain the technicians feedback on the usability and acceptance of the system.

Appendices

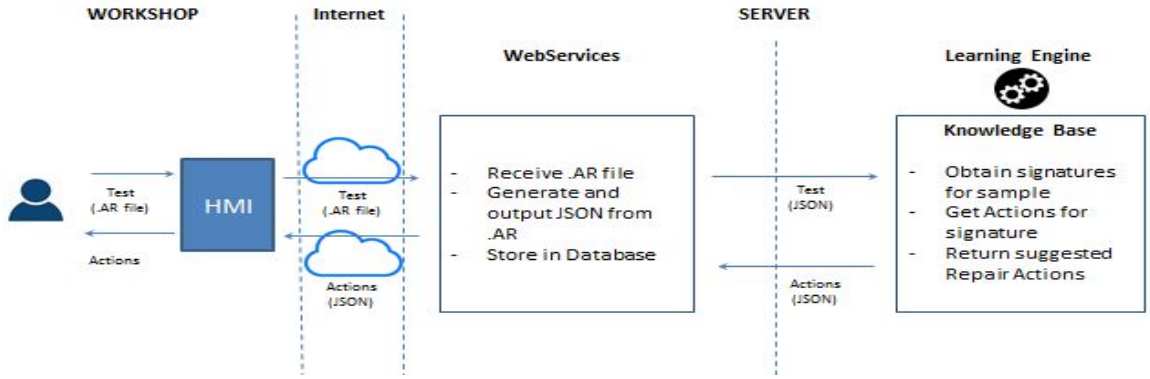
Appendix A

User Manual

e-Diag User Manual

The e-Diag initiative has the goal of supporting diagnosis in avionics maintenance by identifying equipment failures and propose suggestions to experts on the corresponding repair actions. The prototype is implemented in a distributed environment involving a Thales Avionics Workshop in Châtellerault, and the BigData platform in Thales Research and Technology, in Palaiseau, both interconnected through a secure VPN (Virtual Private Network).

On the side of the workshop we have the technician that performs the tests of the faulty equipment, and in Thales R&T we count with a cluster of servers that run several web-services, host the database (Cassandra3), the knowledge base (OWL files) and the learning engine (implemented in Java).These interactions are illustrated in the following figure:



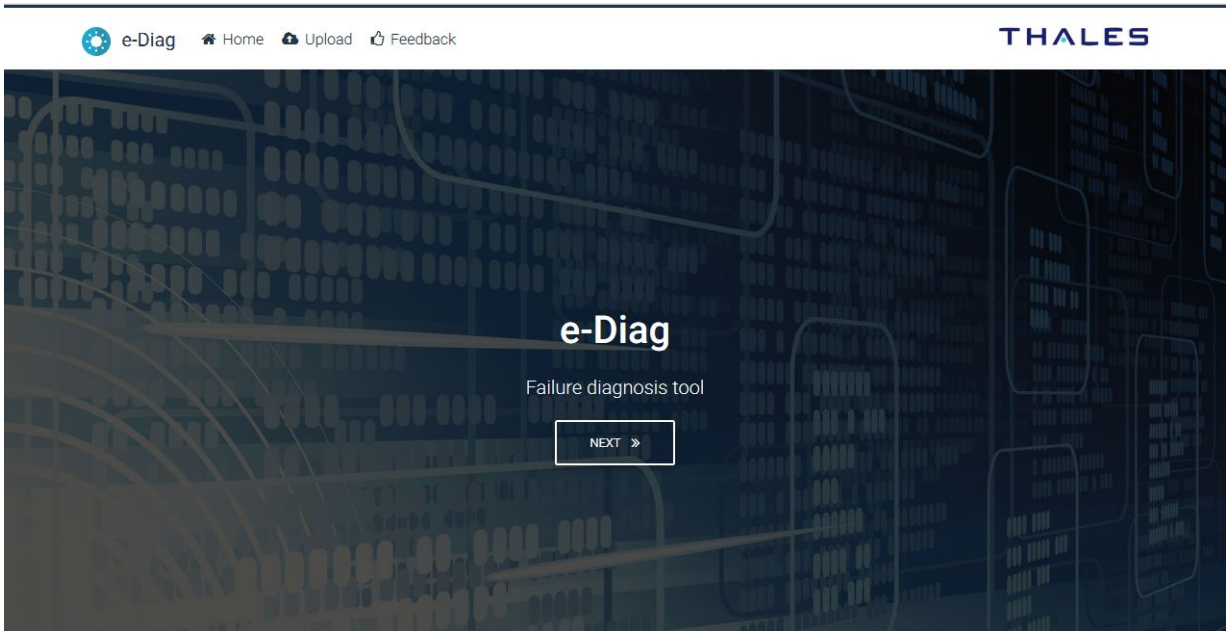
Before the prototype can be used for consulting, a preliminary phase of training the knowledge base has taken place. Once the training is complete, the system is ready to classify unseen tests, to provide suggested actions and to enrich its knowledge base through feedback.

This version of the prototype is limited to the **ELAC** (Elevator and Aileron Computer) test results as input, and **component replacements are provided as output.** . These component replacements are characterized by a board, a position and a component type

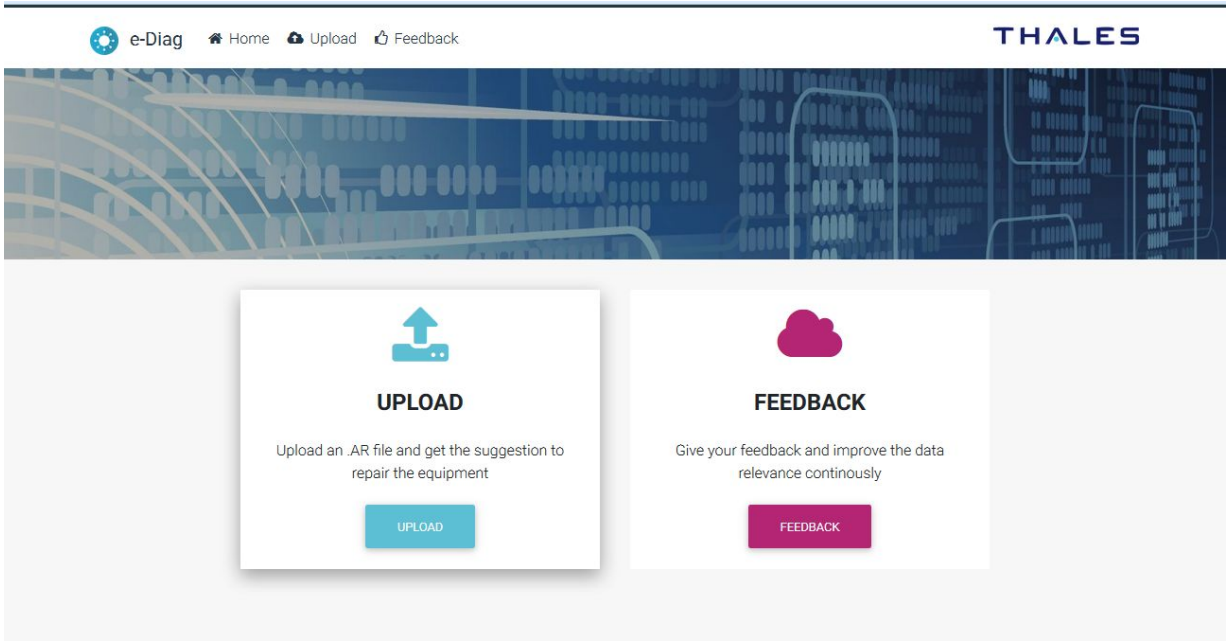
Access the system

The user interface is designed using HTML (V4) and JavaScript (1.7) , and works with any up-to-date web browser. We have used FireFox (V 50 +) for the tests. Nevertheless, since we are in a controlled environment, the tool is only accessible using the station placed in the Châtellerault workshop connected to the BigData network in TRT.

In the first connection to the system, the welcome screen appears:



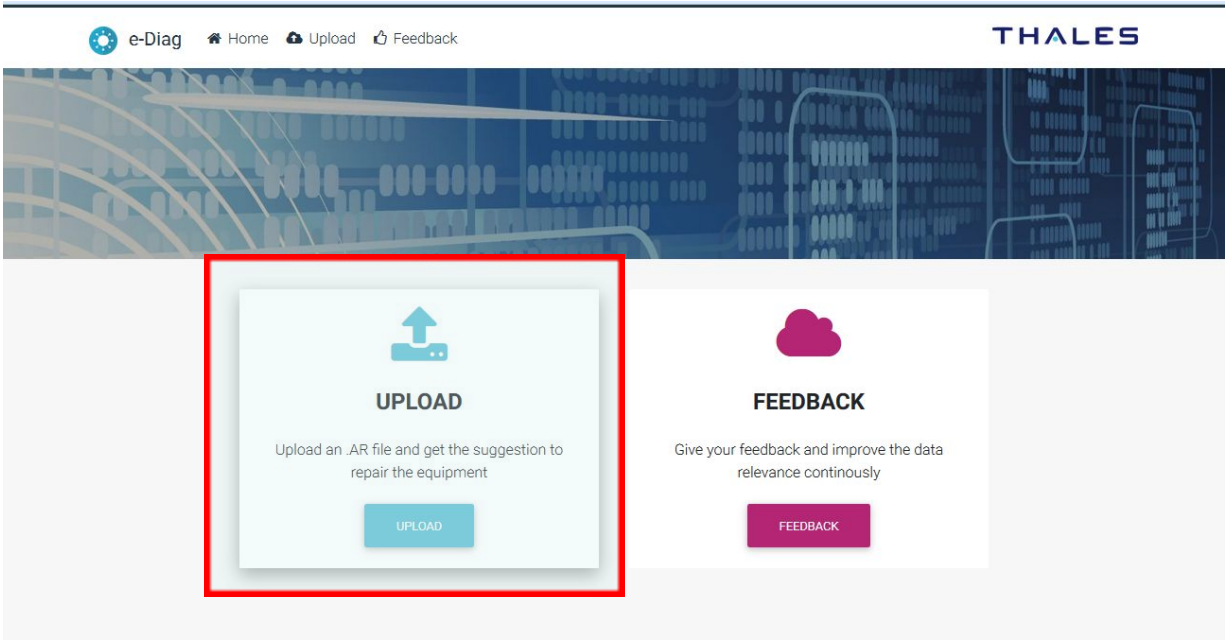
Once inside, the prototype provides two main functionalities: **consulting** and **feedback**.


e-Diag
Home
Upload
Feedback
THALES
UPLOAD
Upload an .AR file and get the suggestion to repair the equipment
UPLOAD
FEEDBACK
Give your feedback and improve the data relevance continously
FEEDBACK

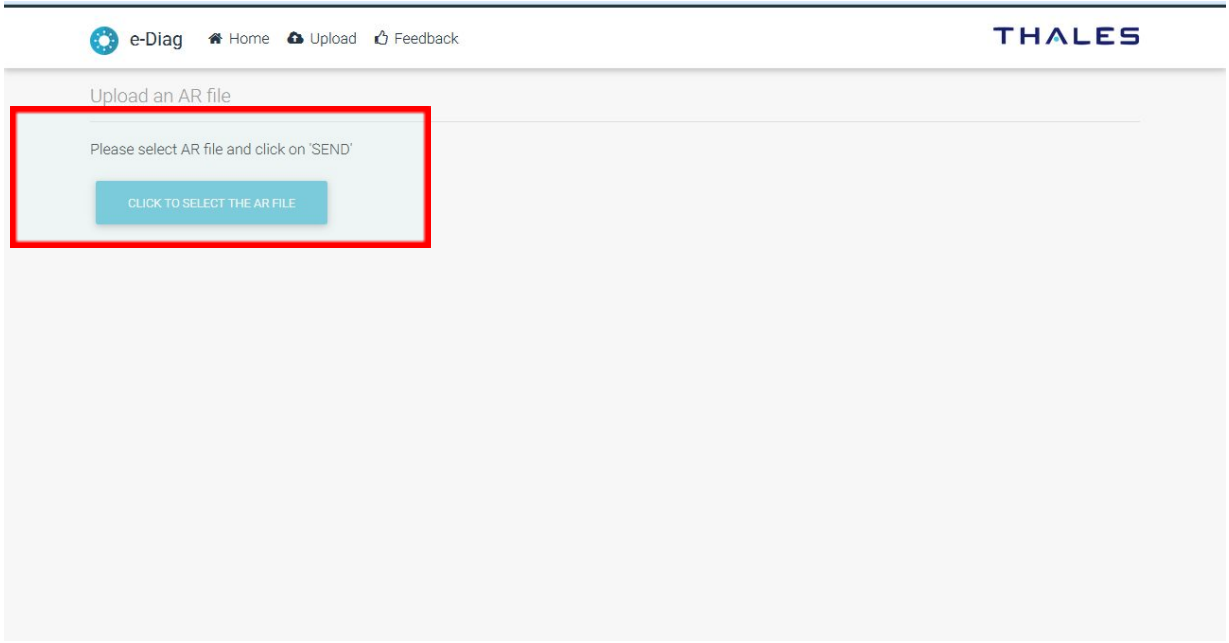
Consulting the Knowledge Base

The knowledge base is the ontology designed for the ELAC use case, plus all the types of failures learned by the system. We want to consult this knowledge base to find the type of failure that best suits the test file being consulted. Once this failure type is identified, we can retrieve the possible corrective actions.

In the consulting phase, first the technician uploads an .AR (All Results) file, which is the output of the test-bench.

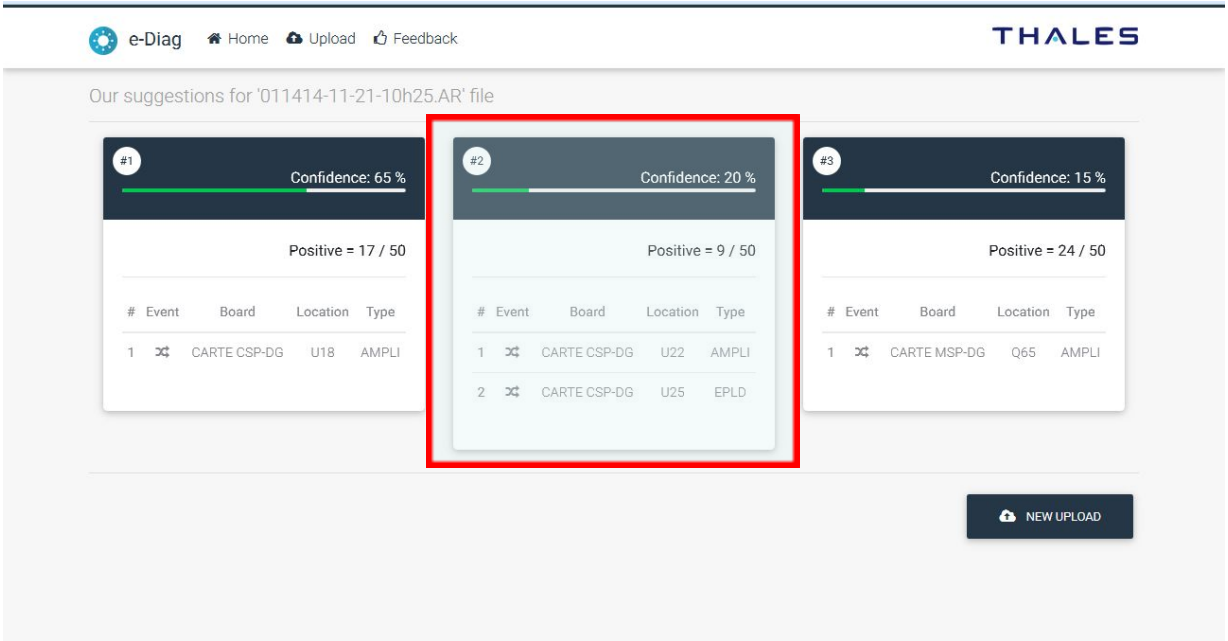


Then, the system provides the user with a pop-up to select the .AR file to analyze from the desired location (hard disk drive, usb key, *etc*.). Once the file chosen, its name will appear on the screen and a **SEND** button will appear to start the analysis.


e-Diag
Home
Upload
Feedback
THALES
Upload an AR file
Please select AR file and click on 'SEND'
CLICK TO SELECT THE AR FILE

Depending on the type of file and the size of the knowledge base, the system will take between a few seconds (20s) to several minutes (+10mn) to analyze the file, and retrieve the suggested actions. More than one suggestion might be proposed for each consulted file. Each suggested action is composed of:

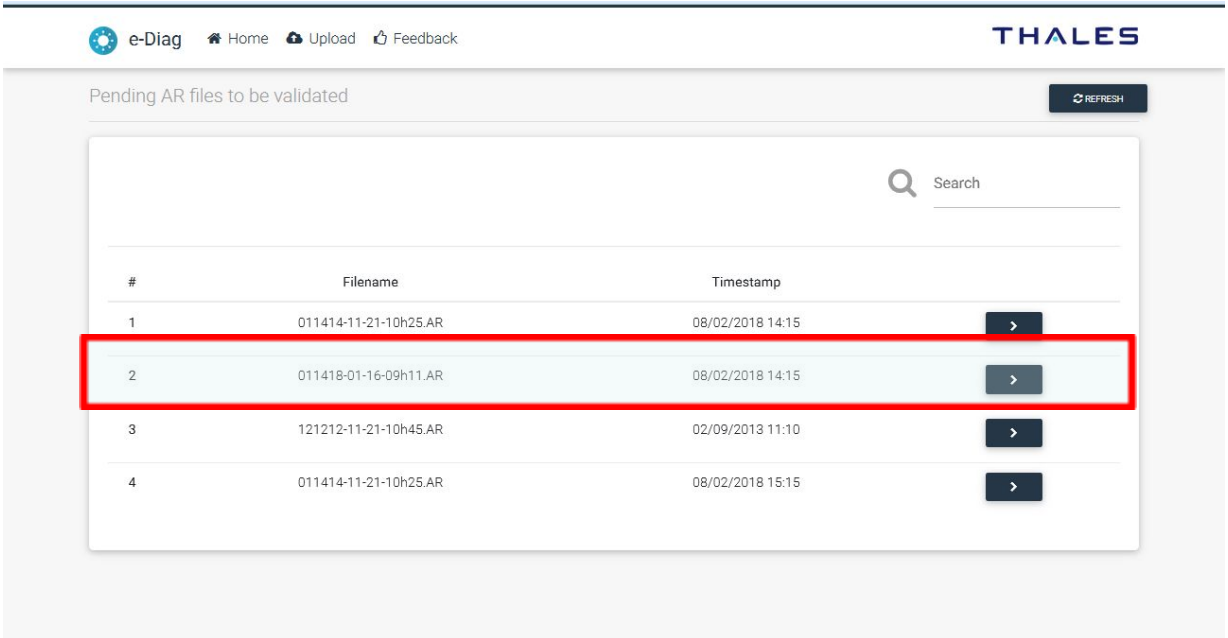
- Number of suggestion.First, second, etc.
- Confidence: the confidence assigned to this particular suggestion to repair the failure.
- Number of positive cases found: How many cases similar to the failure found, were solved by this corrective action (proportional to the confidence).
- Replacements: The components: their location and type, to be replaced.



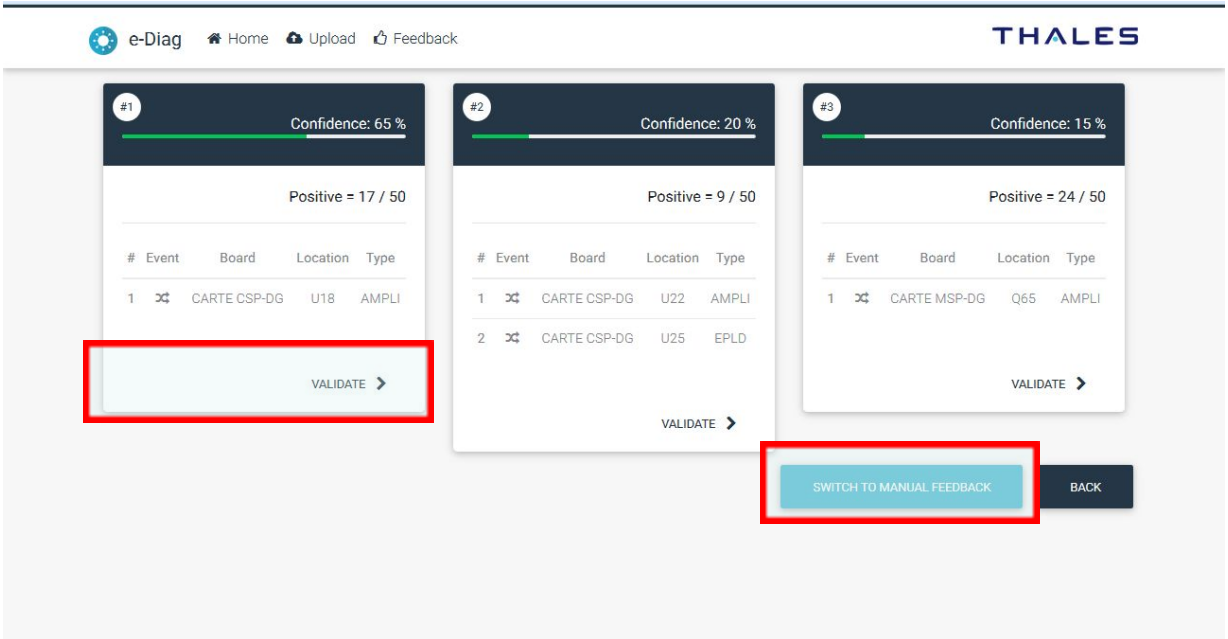
The consulting phase ends here. The technician may use the suggestions to help him diagnose and repair the equipment.

Feedback

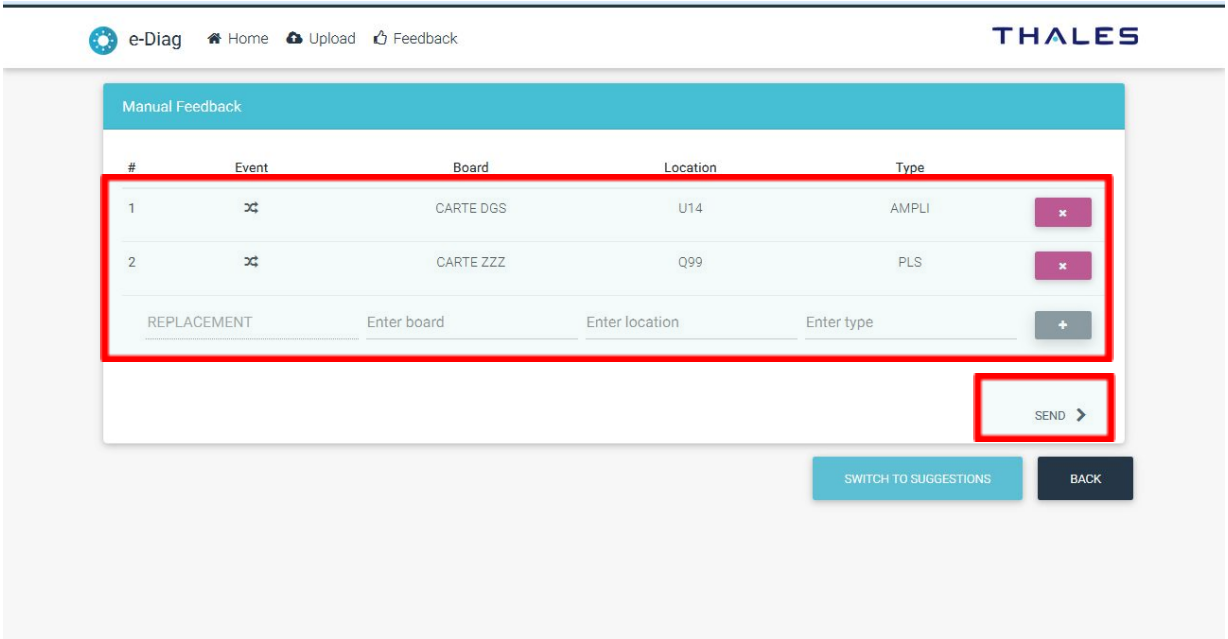
Once the technicians have made the repair, based or not on the proposition(s) given by the consultation phase, the right corrective action to repair the equipment is known (which might differ from the suggested one). This feedback is used to increase/decrease the confidence of the given suggestions. The system displays to the user a list of all the consulted files, for which the feedback has not yet been given. The user must select the correct file, and the system will show the previously found suggestions, so that the user can validate them.



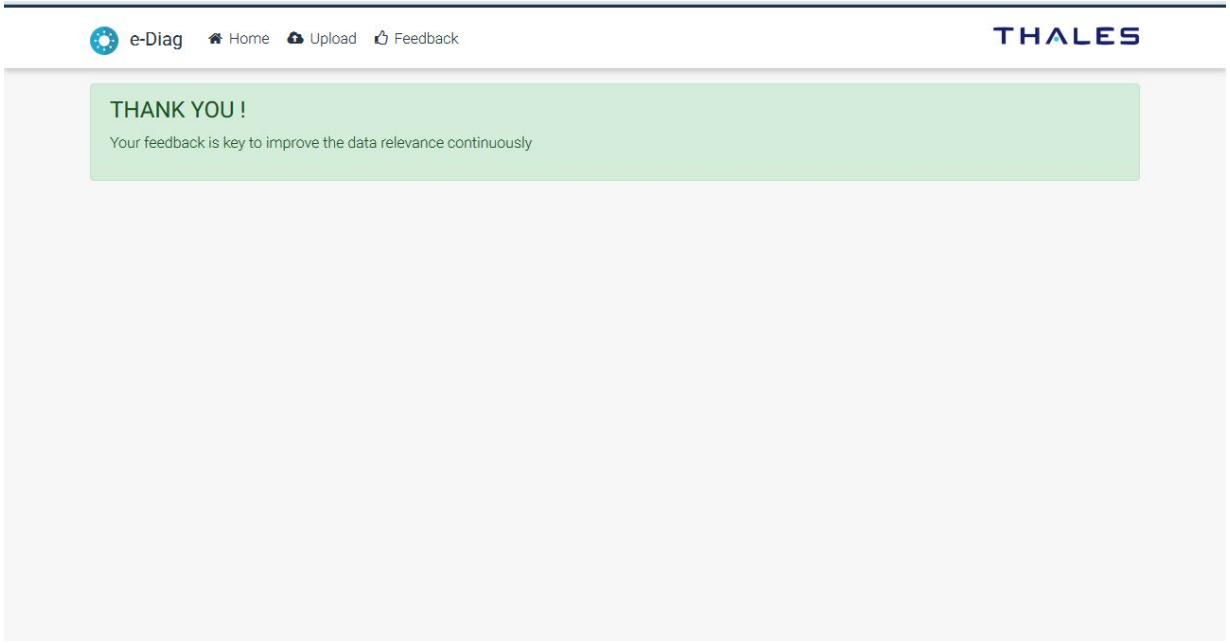
This screen is similar to the one obtained in the consulting phase, but this time the tool gives the user the chance to validate the suggestions (meaning the suggested replacement action indeed solved the failure) or to turn to manual feedback if no proposition was correct. Manual feedback implied that none of the suggestions were correct or that they were not accurate.


e-Diag
Home
Upload
Feedback
THALES
#1
Confidence: 65 %
Positive = 17 / 50
# Event Board Location Type
1 CARTE CSP-DG U18 AMPLI
VALIDATE
#2
Confidence: 20 %
Positive = 9 / 50
# Event Board Location Type
1 CARTE CSP-DG U22 AMPLI
2 CARTE CSP-DG U25 EPLD
VALIDATE
#3
Confidence: 15 %
Positive = 24 / 50
# Event Board Location Type
1 CARTE MSP-DG Q65 AMPLI
VALIDATE
SWITCH TO MANUAL FEEDBACK
BACK

If Manual Feedback is selected, the user is presented with a form to fill in the components replaced; for each component replaced he/she must fill in: the board, the location and the type of each replaced component. Since the initial suggestion was not correct, an additional learning phase takes place to verify if there exists a new type of failure, and to associate the new corrective actions. All this process runs in the background and is transparent to the user.
The background learning process analyses the test file, and tries to create a new failure description based on the information this new test contains. Once the new failure description is found, it is added to the knowledge based and is made available for the subsequent consultations.



Finally, either through the automatic or manual feedback, the system will acknowledge it has properly received the information from the technician with a “Thank you” message. This ends the feedback phase of the prototype.


e-Diag
Home
Upload
Feedback
THALES
THANK YOU !
Your feedback is key to improve the data relevance continuously

Appendix B

Experiments Appendix

This appendix presents the results of the cross validation experiments for the knowledge bases $KB_{100}^{p1 \cup p3}$ and $KB_{100}^{p2 \cup p3}$, that were not shown in chapter 6.

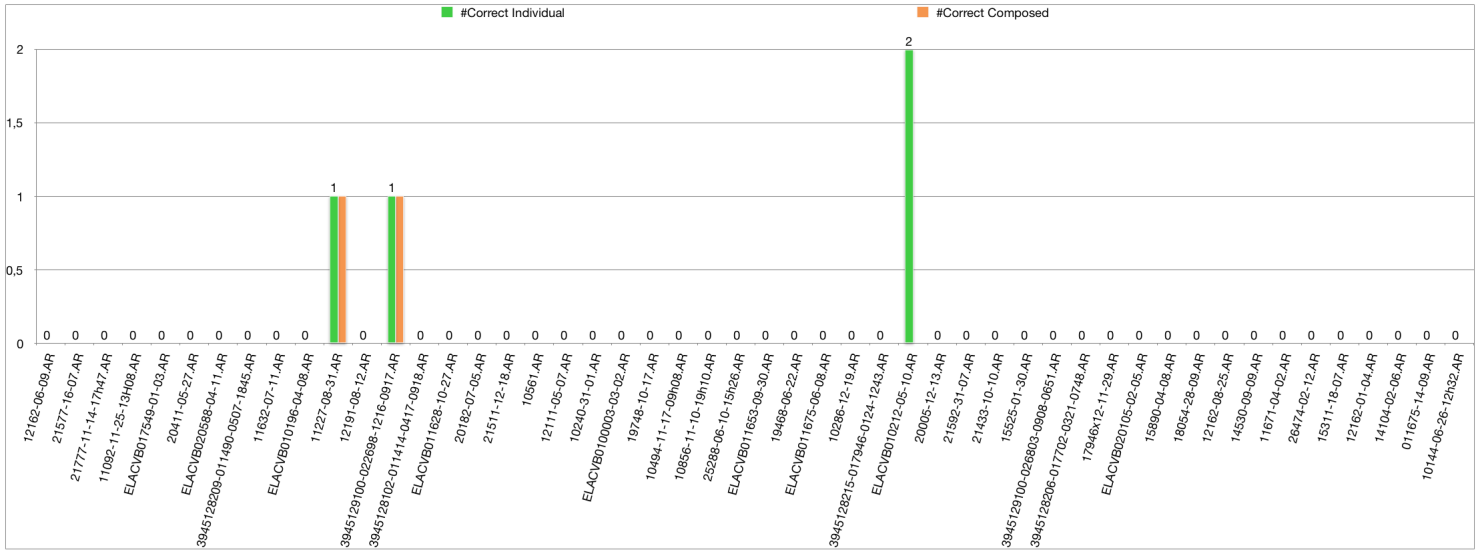


Figure B.1: The suggestions for the 50 files in partition $p2$, when consulting the knowledge base $KB_{100}^{p1 \cup p3}$. In the figure are shown the correct atomic actions (green) and the correct composed actions (orange).

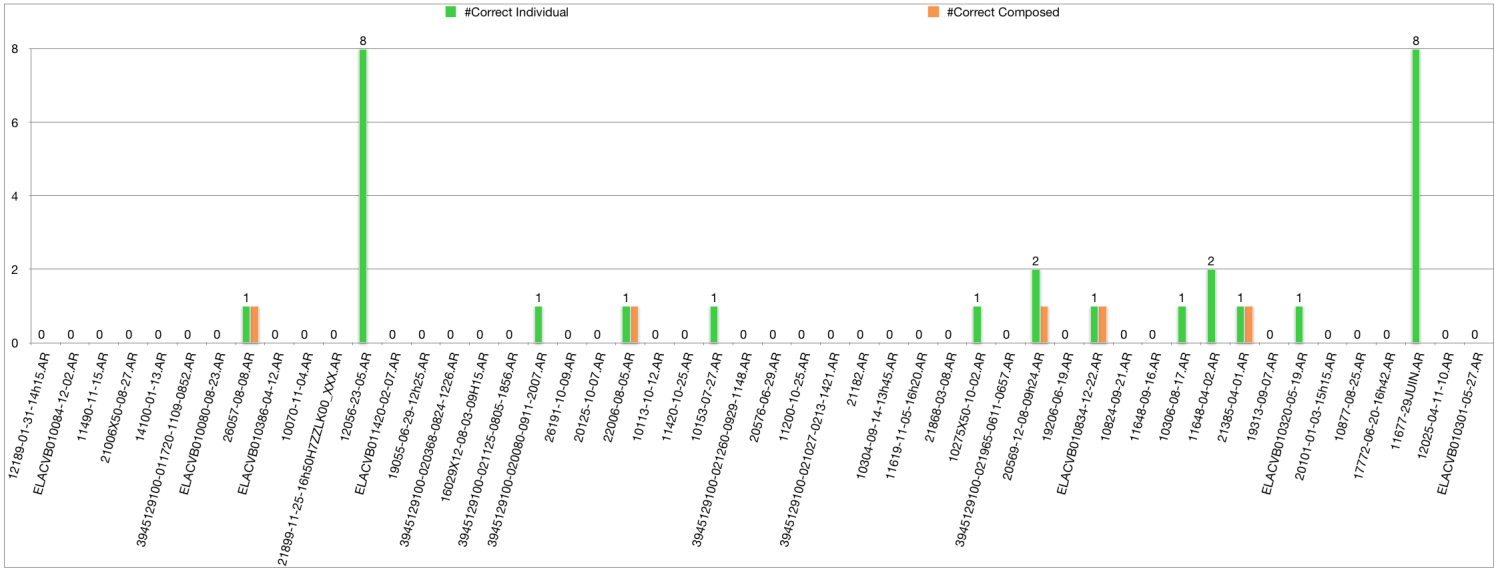


Figure B.2: The suggestions for the 50 files in partition $p1$, when consulting the knowledge base $KB_{100}^{p2 \cup p3}$. The figure shows the correct atomic actions (green) and the correct composed actions (orange).

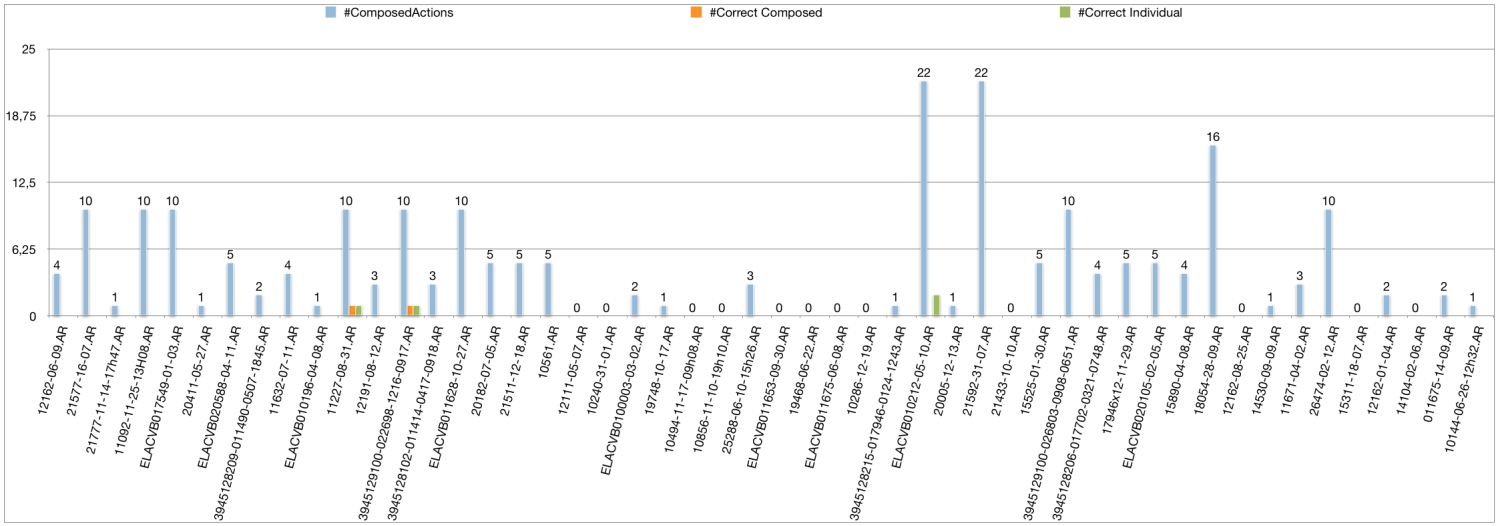


Figure B.3: The suggestions for the 50 files in partition $p2$, when consulting the knowledge base $KB_{100}^{p1 \cup p3}$. The figure shows all composed actions suggested (light blue) vs. the correct atomic actions (green) and the correct composed actions (orange). For visibility, a cut on 25 suggested actions limits the graphic.

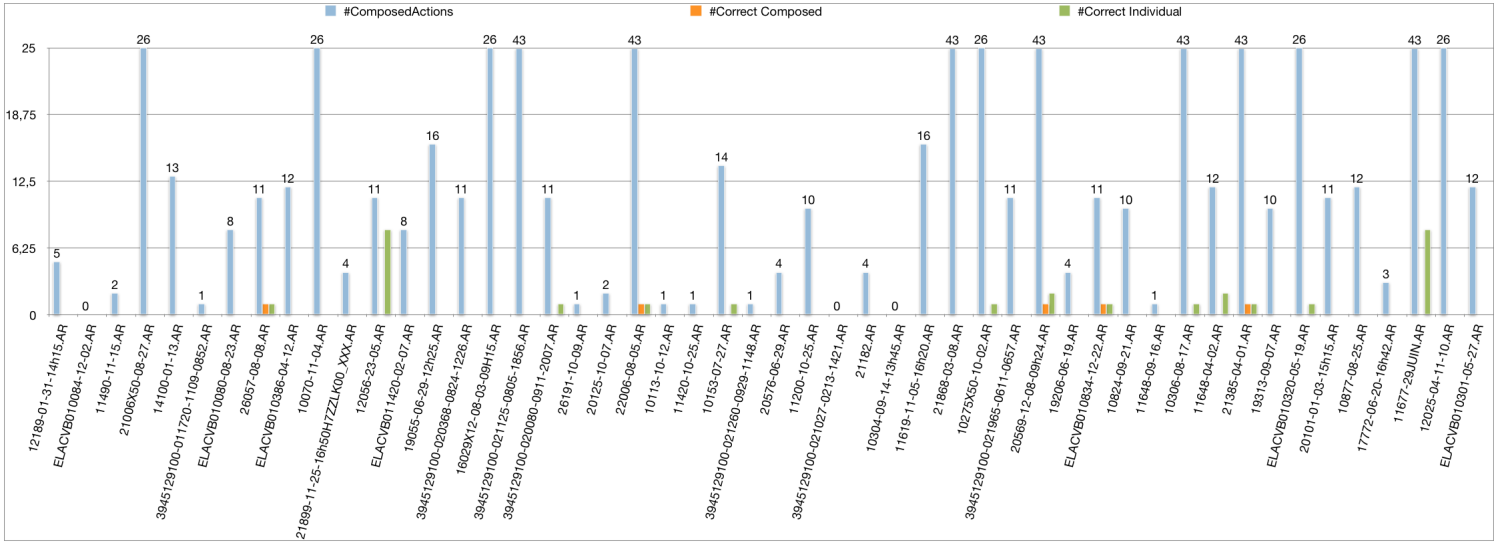


Figure B.4: The suggestions for the 50 files in partition $p1$, when consulting the knowledge base $KB_{100}^{p2 \cup p3}$. The figure shows all composed actions suggested (light blue) vs. the correct atomic actions (green) and the correct composed actions (orange). For visibility, a cut on 25 suggested actions limits the graphic.

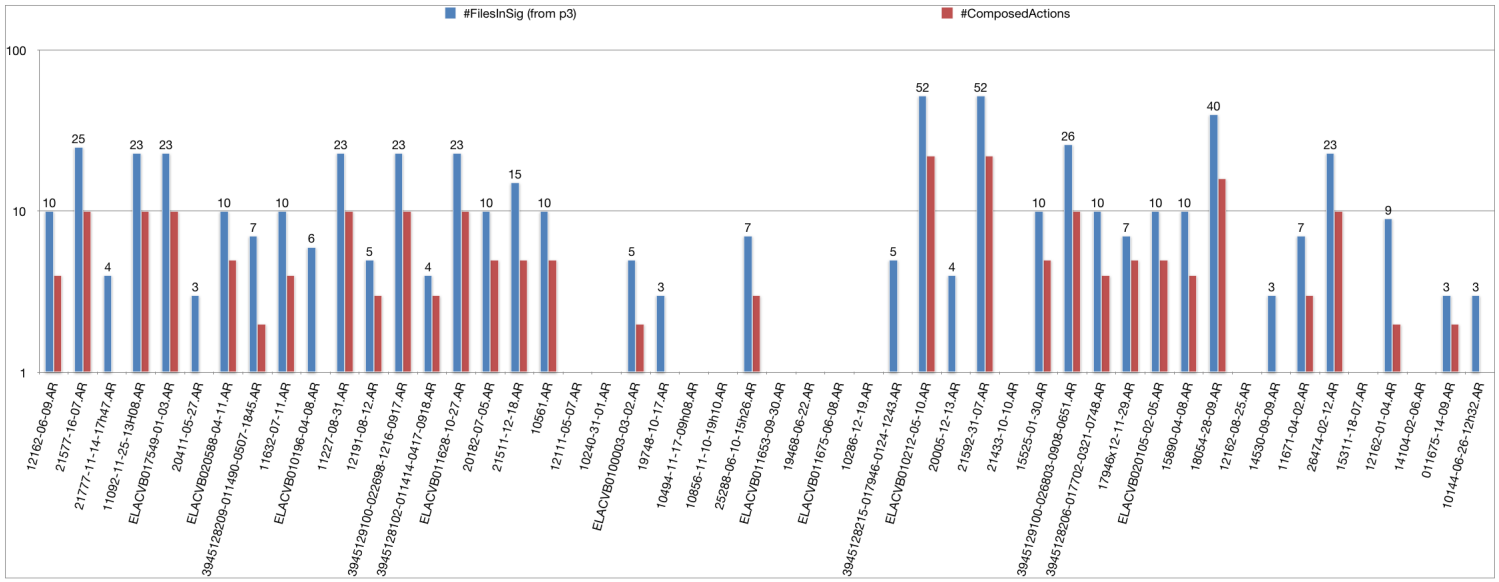


Figure B.5: When each of the 50 files in partition $p2$ were consulted, the knowledge base $KB_{100}^{p1 \cup p3}$ selected the best signature for them. The figure shows the total number of files that belong to the same signature, and the total number of suggestions associated.

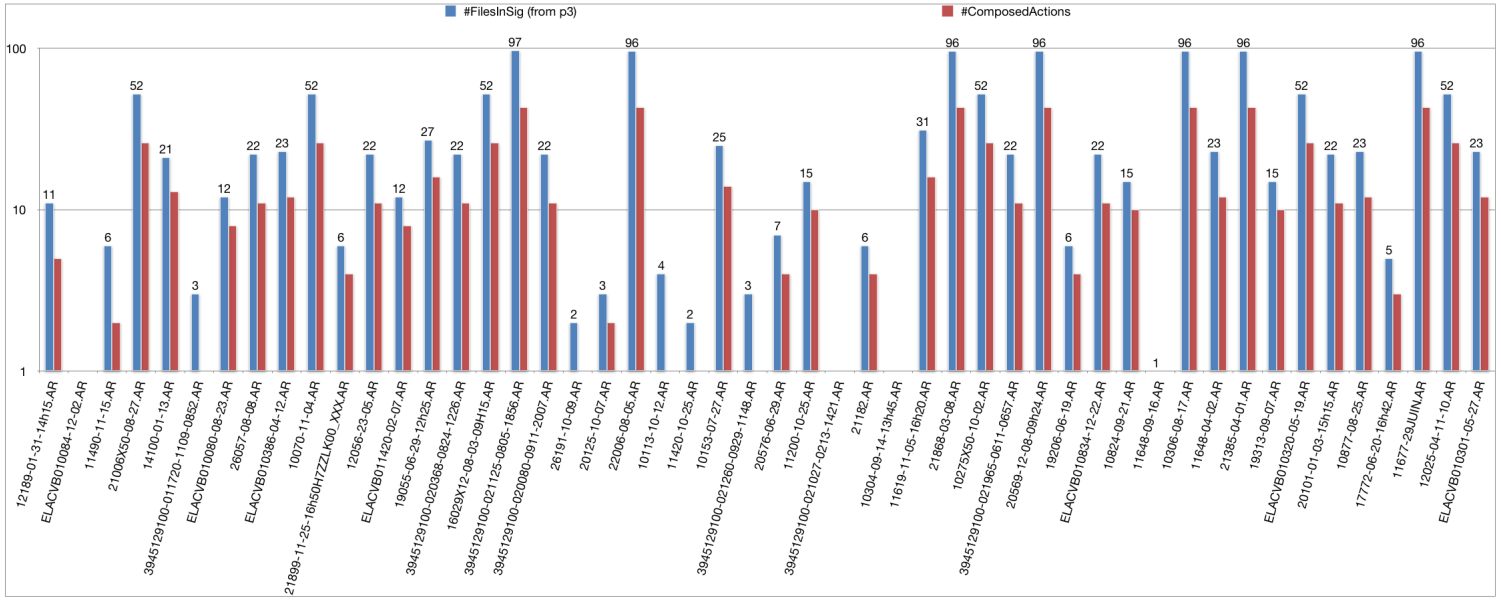


Figure B.6: When each of the 50 files in partition $p1$ were consulted, the knowledge base $KB_{100}^{p2 \cup p3}$ selected the best signature for them. The figure shows the total number of files that belong to the same signature, and the total number of suggestions associated.

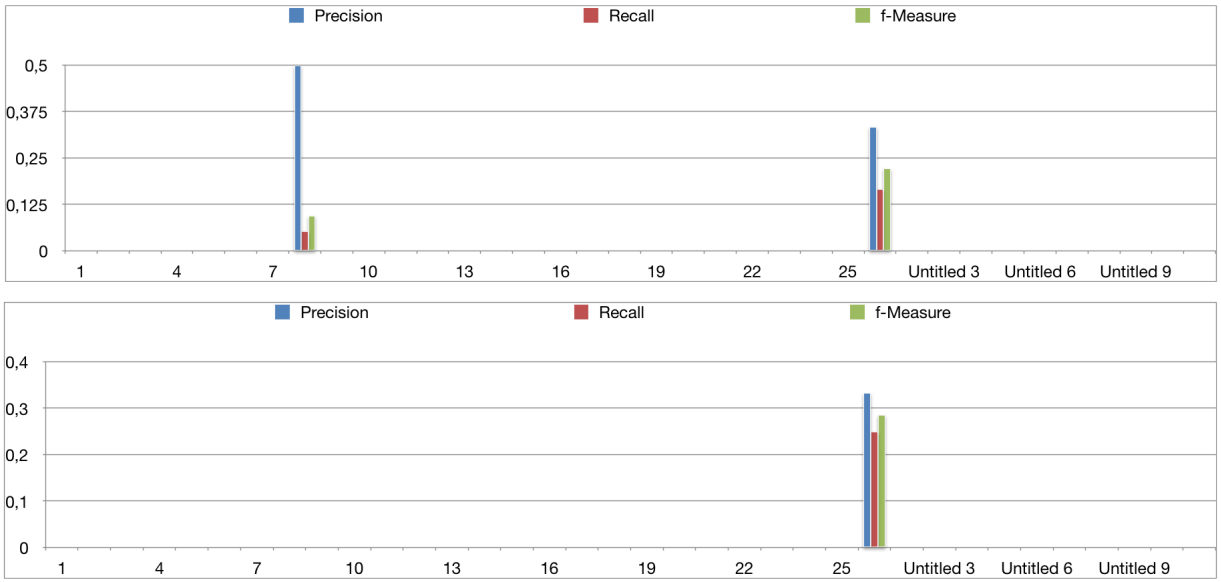


Figure B.7: The 50 files in partition $p2$ were classified under 25 signatures when consulting the knowledge base $KB_{100}^{p1 \cup p3}$. The figure shows, for each signature the precision, the recall and the f-measure, considering partial answers.

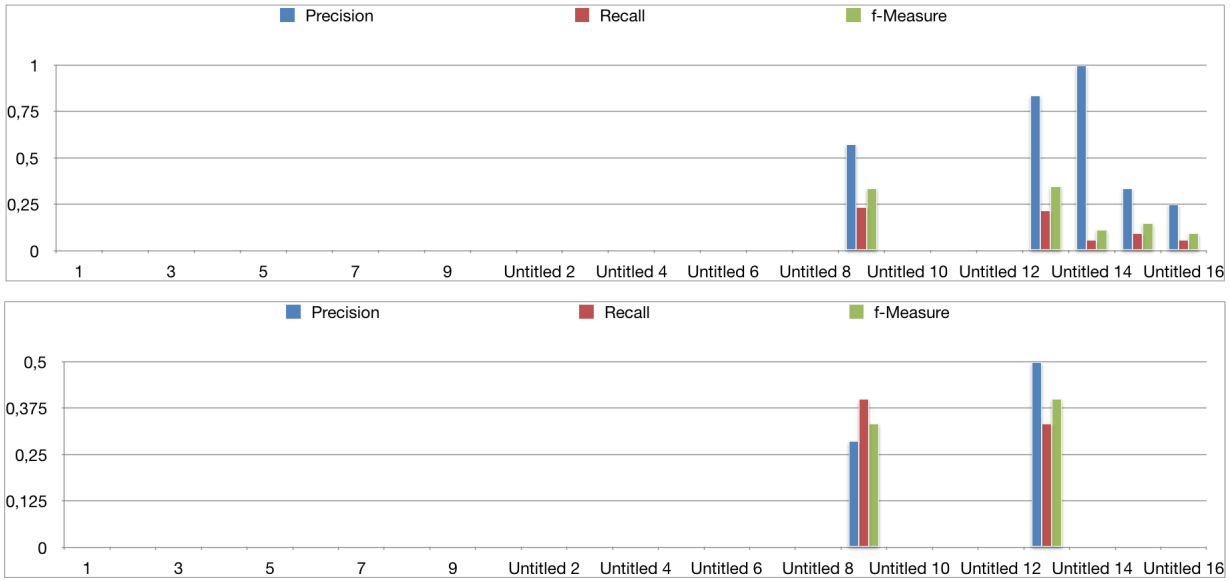


Figure B.8: The 50 files in partition $p1$ were classified under 25 signatures when consulting the knowledge base $KB_{100}^{p2 \cup p3}$. In the figure, for each signature we show the precision, the recall and the f-measure, considering partial answers.

Bibliography

[Aggarwal and Yu, 1998] Aggarwal, C. C. and Yu, P. S. (1998). Online generation of association rules. In *Proceedings of the Fourteenth International Conference on Data Engineering*, ICDE '98, pages 402–411, Washington, DC, USA. IEEE Computer Society.

[Arif et al., 2016] Arif, M. F., Mencía, C., Ignatiev, A., Manthey, N., Peñaloza, R., and Marques-Silva, J. (2016). BEACON: an efficient sat-based tool for debugging $EL\hat{}+$ ontologies. In *SAT*, volume 9710 of *Lecture Notes in Computer Science*, pages 521–530. Springer.

[Baader, 2003a] Baader, F. (2003a). *The description logic handbook: Theory, implementation and applications*. Cambridge university press.

[Baader, 2003b] Baader, F. (2003b). The instance problem and the most specific concept in the description logic el w.r.t. terminological cycles with descriptive semantics. volume LTCS-03-01.

[Baader et al., 2005] Baader, F., Brandt, S., and Lutz, C. (2005). Pushing the $\mathcal{EL}$ envelope. In *Proceedings of IJCAI'05*.

[Baader et al., 2010] Baader, F., Calvanese, D., McGuinness, D. L., Nardi, D., and Patel-Schneider, P. F., editors (2010). *The description logic handbook: theory, implementation, and applications*. Cambridge University Press, 2 edition.

[Baader et al., 2017] Baader, F., Horrocks, I., Lutz, C., and Sattler, U. (2017). *An Introduction to Description Logic*. Cambridge University Press.

[Baader and Molitor, 1999] Baader, F. and Molitor, R. (1999). Computing least common subsumers in description logics with existential restrictions.

[Baader et al., 2007] Baader, F., Peñaloza, R., and Suntisrivaraporn, B. (2007). Pinpointing in the description logic el.

[Badea and Nienhuys-Cheng, 2000] Badea, L. and Nienhuys-Cheng, S.-H. (2000). A refinement operator for description logics. In *International Conference on Inductive Logic Programming*, pages 40–59. Springer.

[Berners-Lee et al., 2001] Berners-Lee, T., Hendler, J., Lassila, O., et al. (2001). The semantic web. *Scientific american*, 284(5):28–37.

[Bühmann and Lehmann, 2012] Bühmann, L. and Lehmann, J. (2012). Universal owl axiom enrichment for large knowledge bases. In *International Conference on Knowledge Engineering and Knowledge Management*, pages 57–71. Springer.

[Bühmann and Lehmann, 2013] Bühmann, L. and Lehmann, J. (2013). Pattern based knowledge base enrichment. In *International Semantic Web Conference*, pages 33–48. Springer.

[C. Insaurralde, 2018] C. Insaurralde, C. (2018). Intelligent autonomy for aerospace engineering systems. pages 1–10.

[Carlson et al., 2010] Carlson, A., Betteridge, J., Kisiel, B., Settles, B., Hruschka, E. R., and Mitchell, T. M. (2010). Toward an architecture for never-ending language learning. In *Twenty-Fourth AAAI Conference on Artificial Intelligence*.

[Cherkassky and Mulier, 2007] Cherkassky, V. and Mulier, F. M. (2007). *Learning from data: concepts, theory, and methods*. John Wiley & Sons.

[Chilton et al., 2013] Chilton, L. B., Little, G., Edge, D., Weld, D. S., and Landay, J. A. (2013). Cascade: Crowdsourcing taxonomy creation. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1999–2008. ACM.

[Cohen and Hirsh, 1994] Cohen, W. W. and Hirsh, H. (1994). Learning the classic description logic: Theoretical and experimental results. *KR*, 94:121–133.

[Consortium et al., 2012] Consortium, W. W. W. et al. (2012). Owl 2 web ontology language document overview.

[De Kleer and Kurien, 2003] De Kleer, J. and Kurien, J. (2003). Fundamentals of model-based diagnosis. *IFAC Proceedings Volumes*, 36(5):25–36.

[Dietz Saldanha et al., 2018] Dietz Saldanha, E.-A., Hölldobler, S., Kencana Ramli, C. D. P., and Palacios Medinacelli, L. (2018). A core method for the weak completion semantics with skeptical abduction. *Journal of Artificial Intelligence Research*, 63:51–86.

[Dittmann et al., 2004] Dittmann, L., Rademacher, T., and Zelewski, S. (2004). Performing fmea using ontologies. In *18th International Workshop on Qualitative Reasoning. Evanston USA*, pages 209–216.

[Duda et al., 2012] Duda, R. O., Hart, P. E., and Stork, D. G. (2012). *Pattern classification*. John Wiley & Sons.

[Ebrahimipour et al., 2010] Ebrahimipour, V., Rezaie, K., and Shokravi, S. (2010). An ontology approach to support fmea studies. *Expert Systems with Applications*, 37(1):671–677.

[Ebrahimipour and Yacout, 2015] Ebrahimipour, V. and Yacout, S. (2015). Ontology-based schema to support maintenance knowledge representation with a case study of a pneumatic valve. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 45(4):702–712.

[Emmanouilidis et al., 2010] Emmanouilidis, C., Fumagalli, L., Jantunen, E., Pistofidis, P., Macchi, M., Garetti, M., et al. (2010). Condition monitoring based on incremental learning and domain ontology for condition-based maintenance. In *11th International Conference on Advances in Production Management Systems, APMS*.

[Fanizzi and d'Amato, 2007] Fanizzi, N. and d'Amato, C. (2007). A hierarchical clustering method for semantic knowledge bases. In *KES*.

[Fanizzi et al., 2008a] Fanizzi, N., d'Amato, C., and Esposito, F. (2008a). Conceptual clustering and its application to concept drift and novelty detection. In *European Semantic Web Conference*, pages 318–332. Springer.

[Fanizzi et al., 2008b] Fanizzi, N., d'Amato, C., and Esposito, F. (2008b). Dl-foil concept learning in description logics. In *International Conference on Inductive Logic Programming*, pages 107–121. Springer.

[Fanizzi et al., 2004] Fanizzi, N., Iannone, L., Palmisano, I., and Semeraro, G. (2004). Concept formation in expressive description logics. In *European Conference on Machine Learning*, pages 99–110. Springer.

[Glimm et al., 2017] Glimm, B., Kazakov, Y., and Tran, T.-K. (2017). Ontology materialization by abstraction refinement in horn shoif. In *Thirty-First AAAI Conference on Artificial Intelligence*.

[Goodfellow et al., 2016] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep learning*. MIT press.

[Ha et al., 2012] Ha, Q.-T., Hoang, T.-L.-G., Nguyen, L. A., Nguyen, H. S., Szałas, A., and Tran, T.-L. (2012). A bisimulation-based method of concept learning for knowledge bases in description logics. In *Proceedings of the Third Symposium on Information and Communication Technology*, pages 241–249. ACM.

[Hayes, 1981] Hayes, P. J. (1981). The logic of frames. In *Readings in Artificial Intelligence*, pages 451–458. Elsevier.

[Héder, 2017] Héder, M. (2017). From nasa to eu: the evolution of the trl scale in public sector innovation. *The Innovation Journal*, 22(2):1–23.

[Horridge, 2011] Horridge, M. (2011). *Justification based explanation in ontologies*. PhD thesis, The University of Manchester (United Kingdom).

[Inokuchi et al., 2000] Inokuchi, A., Washio, T., and Motoda, H. (2000). An apriori-based algorithm for mining frequent substructures from graph data. In *Proceedings of the 4th European Conference on Principles of Data Mining and Knowledge Discovery*, PKDD '00, pages 13–23, London, UK, UK. Springer-Verlag.

[Inokuchi et al., 2003] Inokuchi, A., Washio, T., and Motoda, H. (2003). Complete mining of frequent patterns from graphs: Mining graph data. *Machine Learning*, 50(3):321–354.

[James et al., 2013] James, G., Witten, D., Hastie, T., and Tibshirani, R. (2013). *An introduction to statistical learning*, volume 112. Springer.

[JEDEC, 2018] JEDEC (2018). *DICTIONARY OF TERMS FOR SOLID-STATE TECHNOLOGY, 7th Edition*. Global Standards for the Microelectronics Industry, 7 edition.

[Kalyanpur et al., 2007] Kalyanpur, A., Parsia, B., Horridge, M., and Sirin, E. (2007). Finding all justifications of owl dl entailments. In *The Semantic Web*, pages 267–280. Springer.

[Karampinas and Triantafillou, 2012] Karampinas, D. and Triantafillou, P. (2012). Crowdsourcing taxonomies. In *Extended Semantic Web Conference*, pages 545–559. Springer.

[Karray et al., 2012] Karray, M. H., Chebel-Morello, B., and Zerhouni, N. (2012). A formal ontology for industrial maintenance. *Applied Ontology*, 7(3):269–310.

[Karunaratne et al., 2010] Karunaratne, T., Bostrom, H., and Norinder, U. (2010). Pre-processing structured data for standard machine learning algorithms by supervised graph propositionalization - a case study with medicinal

chemistry datasets. In *2010 Ninth International Conference on Machine Learning and Applications*, pages 828–833.

[Kazakov et al., 2012] Kazakov, Y., Krötzsch, M., and Simancik, F. (2012). Practical reasoning with nominals in the el family of description logics. In *Thirteenth International Conference on the Principles of Knowledge Representation and Reasoning*.

[Kazakov and Skocovský, 2018] Kazakov, Y. and Skocovský, P. (2018). Enumerating justifications using resolution. In *Proceedings of IJCAR'18: the 9th International Joint Conference on Automated Reasoning*, pages 609–626.

[Keller, 2016] Keller, R. M. (2016). Ontologies for aviation data management. In *2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC)*, pages 1–9. IEEE.

[Ketkar et al., 2005] Ketkar, N. S., Holder, L. B., and Cook, D. J. (2005). Subdue: Compression-based frequent pattern discovery in graph data. In *Proceedings of the 1st International Workshop on Open Source Data Mining: Frequent Pattern Mining Implementations*, OSDM '05, pages 71–76, New York, NY, USA. ACM.

[Kietz and Morik, 1994] Kietz, J.-U. and Morik, K. (1994). A polynomial approach to the constructive induction of structural knowledge. *Machine Learning*, 14(2):193–217.

[Kontchakov et al., 2011] Kontchakov, R., Lutz, C., Toman, D., Wolter, F., and Zakharyaschev, M. (2011). The combined approach to ontology-based data access. In Walsh, T., editor, *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*, pages 2656–2661. IJCAI/AAAI.

[Lee and Gray, 1998] Lee, O. and Gray, P. (1998). Knowledge base clustering for kbs maintenance. *Journal of Software Maintenance: Research and Practice*, 10(6):395–414.

[Lehmann, 1992] Lehmann, F. (1992). *Semantic networks in artificial intelligence*. Elsevier Science Inc.

[Lehmann, 2009] Lehmann, J. (2009). Dl-learner: learning concepts in description logics. *Journal of Machine Learning Research*, 10(Nov):2639–2642.

[Lehmann et al., 2011] Lehmann, J., Auer, S., Bühmann, L., and Tramp, S. (2011). Class expression learning for ontology engineering. *Web Semantics: Science, Services and Agents on the World Wide Web*, 9(1):71–81.

[Lehmann and Haase, 2009] Lehmann, J. and Haase, C. (2009). Ideal downward refinement in the\ mathcal {EL} description logic. In *International Conference on Inductive Logic Programming*, pages 73–87. Springer.

[Lehmann and Hitzler, 2010] Lehmann, J. and Hitzler, P. (2010). Concept learning in description logics using refinement operators. *Machine Learning*, 78(1-2):203.

[Lehmann et al., 2017] Lehmann, J., Sejdiu, G., Bühmann, L., Westphal, P., Stadler, C., Ermilov, I., Bin, S., Chakraborty, N., Saleem, M., Ngomo, A.-C. N., et al. (2017). Distributed semantic analytics using the sansa stack. In *International Semantic Web Conference*, pages 147–155. Springer.

[Lehmann and Voelker, 2014] Lehmann, J. and Voelker, J. (2014). An introduction to ontology learning. *Perspectives on Ontology Learning. Amsterdam: IOS Press*.

[Lehmann and Völker, 2014] Lehmann, J. and Völker, J. (2014). *Perspectives on ontology learning*, volume 18. IOS Press.

[Lisi and Esposito, 2009] Lisi, F. A. and Esposito, F. (2009). Nonmonotonic onto-relational learning. In *International Conference on Inductive Logic Programming*, pages 88–95. Springer.

[Minsky, 1974] Minsky, M. (1974). A framework for representing knowledge.

[Motoda, 2007] Motoda, H. (2007). *Pattern Discovery from Graph-Structured Data - A Data Mining Perspective*, pages 12–22. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Nguyen and Szałas, 2013] Nguyen, L. A. and Szałas, A. (2013). Logic-based roughification. *Rough Sets and Intelligent Systems-Professor Zdzisław Pawlak in Memoriam*, pages 517–543.

[Nienhuys-Cheng and De Wolf, 1997] Nienhuys-Cheng, S.-H. and De Wolf, R. (1997). *Foundations of inductive logic programming*, volume 1228. Springer Science & Business Media.

[Nowak-Brzezińska, 2016] Nowak-Brzezińska, A. (2016). Mining rule-based knowledge bases inspired by rough set theory. *Fundamenta Informaticae*, 148(1-2):35–50.

[Olson and Delen, 2008] Olson, D. L. and Delen, D. (2008). *Advanced data mining techniques*. Springer Science & Business Media.

[Palacios Medinacelli et al., 2018] Palacios Medinacelli, L., Gaëlle, L., Yue, M., and Chantal, R. (2018). Knowledge discovery for avionics maintenance support. In *2018 IEEE/AIAA 37th Digital Avionics Systems Conference (DASC)*, pages 1–8. IEEE.

[Palacios Medinacelli et al., 2016] Palacios Medinacelli, L., Lortal, G., Laudy, C., Sannino, C., Simon, L., Fusco, G., Ma, Y., and Reynaud, C. (2016). Avionics maintenance ontology building for failure diagnosis support. In *Proceedings of the 8th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (IC3K 2016)*, pages 204–209.

[Palacios Medinacelli et al., 2017] Palacios Medinacelli, L., Ma, Y., Lortal, G., Laudy, C., and Reynaud, C. (2017). Data driven concept refinement to support avionics maintenance. In *Proceedings of the IJCAI Workshop on Semantic Machine Learning*.

[Pedrycz, 2005] Pedrycz, W. (2005). *Knowledge-based clustering: from data to information granules*. John Wiley & Sons.

[Peñaloza and Turhan, 2010] Peñaloza, R. and Turhan, A.-Y. (2010). Towards approximative most specific concepts by completion for el with subjective probabilities. CEUR.

[Quillian, 1967] Quillian, M. R. (1967). Word concepts: A theory and simulation of some basic semantic capabilities. *Behavioral science*, 12(5):410–430.

[Ratcliffe and Taylor, 2017] Ratcliffe, D. and Taylor, K. (2017). Refinement-based owl class induction with convex measures. In *Joint International Semantic Technology Conference*, pages 49–65. Springer.

[Regal and Pereira, 2014] Regal, T. and Pereira, C. (2014). Building an ontology for intelligent maintenance systems and spare parts supply chain integration. *IFAC Proceedings Volumes (IFAC-PapersOnline)*, 19:7843–7848.

[Simperl et al., 2012] Simperl, E., Bürger, T., Hangl, S., Wörgl, S., and Popov, I. (2012). Ontocom: A reliable cost estimation method for ontology development projects. *Web Semantics: Science, Services and Agents on the World Wide Web*, 16:1–16.

[Sowa, 1987] Sowa, J. F. (1987). Semantic networks.

[Tran et al., 2012] Tran, T.-L., Ha, Q.-T., Nguyen, L. A., Nguyen, H. S., Szalas, A., et al. (2012). Concept learning for description logic-based information systems. In *Knowledge and Systems Engineering (KSE), 2012 Fourth International Conference on*, pages 65–73. IEEE.

[Tran et al., 2015] Tran, T.-L., Nguyen, L. A., et al. (2015). Bisimulation-based concept learning for information systems in description logics. *Vietnam Journal of Computer Science*, 2(3):149–167.

[Völker and Niepert, 2011] Völker, J. and Niepert, M. (2011). Statistical schema induction. In *Extended Semantic Web Conference*, pages 124–138. Springer.

[Wu et al., 2014] Wu, Y., Ebrahimipour, V., and Yacout, S. (2014). Ontology-based modeling of aircraft to support maintenance management system. In *IIE Annual Conference. Proceedings*, page 1159. Institute of Industrial and Systems Engineers (IISE).

[Yan and Han, 2003] Yan, X. and Han, J. (2003). Closegraph: Mining closed frequent graph patterns. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '03, pages 286–295, New York, NY, USA. ACM.

[Yang et al., 2014] Yang, B., Yih, W.-t., He, X., Gao, J., and Deng, L. (2014). Embedding entities and relations for learning and inference in knowledge bases. *arXiv preprint arXiv:1412.6575*.

[Yoshida et al., 1994] Yoshida, K., Motoda, H., and Indurkhya, N. (1994). Graph-based induction as a unified learning framework. *Applied Intelligence*, 4(3):297–316.

université PARIS-SACLAY
ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

Titre : Découverte de connaissances pour la maintenance avionique, une approche d'apprentissage de concepts non supervisée

Mots clés : Ontologie, Base de Connaissance, Apprentissage Automatique, Maintenance Avionique, Raffinement de Concepts, Apprentissage de Concepts

Résumé : Dans cette thèse, nous étudions le problème de l'analyse de signatures de pannes dans le domaine de la maintenance avionique, afin d'identifier les défaillances au sein d'équipements en panne et suggérer des actions correctives permettant de les réparer. La thèse a été réalisée dans le cadre d'une convention CIFRE entre Thales Research & Technology et l'Université Paris-Sud. Les motivations sont donc à la fois théoriques et industrielles.
Une signature de panne fournit toutes les informations nécessaires pour identifier, comprendre et réparer la panne. Son identification doit donc être explicable. Nous proposons une approche à base d'ontologies pour modéliser le domaine d'étude, permettant une interprétation automatisée des tests techniques réalisés afin d'identifier les pannes et obtenir les actions correctives associées. Il s'agit d'une approche d'apprentissage de concepts permettant de découvrir des concepts représentant les signatures de pannes. Comme les signatures ne sont pas connues a priori, un algorithme d'apprentissage automatique non supervisé approxime les définitions des concepts. Les signatures apprises sont fournies sous forme de définitions de la logique de description (DL) et ces définitions servent d'explications. Contrairement aux techniques courantes d'apprentissage de concepts conçues pour faire de l'apprentissage supervisé ou basées sur l'analyse de patterns fréquents au sein de gros volumes de données, l'approche proposée adopte une perspective différente. Elle repose sur une construction bottom-up de l'ontologie. Le processus d'apprentissage est réalisé via un opérateur de raffinement appliqué sur l'espace des expressions de concepts et le processus est guidé par les données, c'est-à-dire les individus de l'ontologie. Ainsi, les notions de justifications, de concepts plus spécifiques et de raffinement de concepts ont été révisés et adaptés pour correspondre à nos besoins. L'approche a ensuite été appliquée au problème de la maintenance avionique. Un prototype a été implémenté et mis en œuvre au sein de Thales Avionics à titre de preuve de concept.

Title : Knowledge Discovery for Avionics Maintenance, An Unsupervised Concept Learning Approach

Keywords : Ontology, Knowledge Base, Machine Learning, Avionics Maintenance, Concept Refinement, Concepts Learning

Abstract : In this thesis we explore the problem of signature analysis in avionics maintenance, to identify failures in faulty equipment and suggest corrective actions to resolve the failure. The thesis takes place in the context of a CIFRE convention between Thales R&T and the Université Paris-Sud, thus it has both a theoretical and an industrial motivation. The signature of a failure provides all the information necessary to understand, identify and ultimately repair a failure. Thus when identifying the signature of a failure it is important to make it explainable. We propose an ontology based approach to model the domain, that provides a level of automatic interpretation of the highly technical tests performed in the equipment. Once the tests can be interpreted, corrective actions are associated to them. The approach is rooted on concept learning, used to approximate description logic concepts that represent the failure signatures. Since these signatures are not known in advance, we require an unsupervised learning algorithm to compute the approximations. In our approach the learned signatures are provided as description logics (DL) definitions which in turn are associated to a minimal set of axioms in the A-Box. These serve as explanations for the discovered signatures. Thus providing a glass-box approach to trace the reasons on how and why a signature was obtained. Current concept learning techniques are either designed for supervised learning problems, or rely on frequent patterns and large amounts of data. We use a different perspective, and rely on a bottom-up construction of the ontology. Similarly to other approaches, the learning process is achieved through a refinement operator that traverses the space of concept expressions, but an important difference is that in our algorithms this search is guided by the information of the individuals in the ontology. To this end the notions of justifications in ontologies, most specific concepts and concept refinements, are revised and adapted to our needs. The approach is then adapted to the specific avionics maintenance case in Thales Avionics, where a prototype has been implemented to test and evaluate the approach as a proof of concept.

Université Paris-Saclay
Espace Technologique / Immeuble Discovery
Route de l'Orme aux Merisiers RD 128 / 91190 Saint-Aubin, France