Universidad San Jorge

Escuela de Arquitectura y Tecnología

Grado en Ingeniería Informática

Proyecto Final

Exploring Program Synthesis in Model Driven Engineering

Autor del proyecto: Rodrigo Casamayor Moragriega

Director del proyecto: Carlos Cetina Englada, PhD

Zaragoza, 15 de junio de 2021

Este trabajo constituye parte de mi candidatura para la obtención del título de Graduado en Ingeniería Informática por la Universidad San Jorge y no ha sido entregado previamente (o simultáneamente) para la obtención de cualquier otro título.

Este documento es el resultado de mi propio trabajo, excepto donde de otra manera esté indicado y referido.

Doy mi consentimiento para que se archive este trabajo en la biblioteca universitaria de Universidad San Jorge, donde se puede facilitar su consulta.

Firma

Fecha

Mar., 15 de junio de 2021

Dedicatoria y Agradecimiento

A Gaspar Ferrer por ser el profesor con el que más disfruté de toda la etapa de Secundaria y del que guardo un mejor recuerdo. Nunca olvidaré aquellos recreos introduciéndome a la programación web o aquella fantástica tarde en el simulador de vuelo. Es para mí un orgullo que fuera él quien me enseñara la que sería mi primera línea de código.

A mis profesores y compañeros del grupo SVIT, África, Gabriel, Jaime, Jesús, Jorge, Lorena, Paqui, Raúl, Violeta, por su generosidad. Han sido unos años inimaginables junto a ellos. Me siento un privilegiado.

A Carlos Cetina por ser un director de tesis magnífico, por su ayuda y acompañamiento constantes y por confiar en mí.

A la Universidad San Jorge y a Inycom, por contribuir a mi formación personal y profesional.

A mis amigos, gracias por las risas, los viajes, las cenas…, que me han ayudado a desconectar del día a día. Me gustaría hacer una mención especial a Juan Carlos, por la gran persona y amigo que es, por sus incansables ánimos, por echarme un cable cuando lo he necesitado, y, en definitiva, por su manera de ser. ¡Por muchos éxitos juntos!

Y por supuesto, a mi familia y, en especial, a mis padres, por esta genética, su infinito cariño y su apoyo incondicional, incluido el económico, claro, a los que estaré eternamente agradecido y a los que dedico este trabajo.

Por último, pero no menos importante, a todas las personas que, aunque no haya nombrado, se han cruzado en mi camino y me han ayudado a llegar hasta aquí.

Si cabe, a mí, por mi dedicación y perseverancia. Sin ellas, estoy seguro de que no lo habría conseguido.

Table of Contents

1. Introduction 3
2. Background 5
2.1. Model Driven Engineering 6
2.2. Program Synthesis 6
2.2.1. User Intent 8
2.2.2. Search Space 8
2.2.3. Search Technique 9
2.2.4. Oracle-Guided Inductive Synthesis (OGIS) 10
2.3. Machine Learning 11
2.3.1. Classification 11
3. State of the Art 13
3.1. Taxonomy of the Related Work 13
3.2. Identification of the Research Gap 20
4. Objectives 25
5. Methodology 26
5.1. Design science 26
6. Analysis 30
6.1. Encoding 30
6.2. Training 32
6.3. Evaluation 32
7. Design 34
7.1. Tools 36
7.1.1. Programming language 36
7.1.2. IDEs 36
7.1.3. Version control 37
7.1.4. Libraries 38
7.1.5. Task management 38
8. Implementation 39
8.1. Baseline relationships 39

8.2. Mutated relationships 40
8.3. Ranking 40
9. Results 42
9.1. Cross-validation 43
9.2. Modeling experiment with students 45
10. Economic study 52
10.1. Cost breakdown 52
10.1.1. Material costs 52
10.1.2. Human costs 53
10.1.3. Infrastructure costs 54
10.1.4. Total costs 54
11. Conclusion 55
11.1. Future work 55
12. Bibliography 56
13. Annexes 59
13.1. Annex 1: Project proposal 59
13.2. Annex 2: Meeting notes 60
13.3. Annex 3: Scopus queries 67
13.4. Annex 4: Taxonomy of the Related Work 69

Table Index

Table 1 - Top venues for Software Systems related publications [Source: Google Scholar] 15
Table 2 - Summary of filtered and non-filtered document results per query (sorted by ascending number of results) 18
Table 3 - Related work selection to be evaluated. 19
Table 4 - Relationships separated by type (SDML modeling experiment) 48
Table 5 - Baseline relationships separated by type (SDML modeling experiment) 48
Table 6 - Mutated relationships separated by type (SDML modeling experiment) 49
Table 7 - Project material costs 53
Table 8 - Project human costs 53
Table 9 - Project infrastructure costs 54
Table 10 - Total project costs 54

Illustration Index

Figure 1 - MDE pretended workflow 3
Figure 2 - Model-driven engineering (MDE) and Program synthesis (PS) stack 7
Figure 3 - Dimensions in Program Synthesis 7
Figure 4 - Simple DAG representation [Source: Wikipedia] 12
Figure 5 - Scope of this project 13
Figure 6 - Search's decision path 15
Figure 7 - State of the Art roadmap 20
Figure 8 - Screenshot of a task in Microsoft Planner 28
Figure 9 - SDML metamodel from Kromaia videogame (graphical representation) 31
Figure 10 - Representation of the aggregation Link in the metamodel and in the feature vector 35
Figure 11 - Python logo 36
Figure 12 - Visual Studio Code logo 37
Figure 13 - Jupyter logo 37
Figure 14 - GitHub logo 37
Figure 15 - Microsoft Planner logo 38
Figure 16 - Dataset construction (algorithm) 39
Figure 17 - Mutant creation for training (algorithm) 40

Figure 18 - Model classification (algorithm) 41
Figure 19 - Flowchart representing the evaluation design. 43
Figure 20 - Mosaic graph with the accuracies of the three classifiers for each relationship 44
Figure 21 - First exercise (SDML modeling experiment) 45
Figure 22 – Laptops' setup (SDML modeling experiment) 46
Figure 23 - SDML modeling experiment with university students 47
Figure 24 - Featured models made by the students during the experiment 50

Resumen

En la actualidad, la mayoría del software se sigue desarrollando a través de métodos convencionales. Esto es, a una mayor demanda de software, mayor contratación de personas físicas que llevan a cabo el desarrollo. Esto es viable cuando el alcance del proyecto es limitado y no se requiere de un mantenimiento prolongado en el tiempo o mejoras de este. Pero, cuando se necesita continuar desarrollando una familia de productos, esto hace que se encarezca el producto final como consecuencia de cada vez más un mayor número de desarrolladores que construyan el software y lo mantengan en el tiempo, con la contraprestación de que posiblemente se introduzcan nuevos errores en el software de estos productos debido a la implementación repetitiva de las características que lo definen, en vez de reaprovechar y modificar las ya existentes. Al automatizar estos procesos, se disminuye en gran medida el error humano y se abaratan los costes de producción.

Este proyecto pretende ayudar a los desarrolladores de software, en particular, a aquellos realizan Ingeniería Dirigida por Modelos (MDE). Más concretamente, a través de técnicas de Síntesis de Programas (PS), ayudar en las tareas que realiza un ingeniero de MDE. A continuación, presentamos un estudio que abarca el estado del arte en la intersección entre MDE y PS. Por último, proponemos una aplicación de Machine Learning (ML) como técnica de síntesis en combinación con modelos. Así, se ha desarrollado un asistente basado en ML, capaz de extraer los patrones que subyacen a las relaciones entre los elementos de un modelo.

Abstract

Today, most software is still developed through conventional methods. That is, the greater the demand for software, the more individuals are hired to carry out the development. This is feasible when the scope of the project is limited and there is no need for prolonged maintenance or enhancements. But, when it is necessary to continue developing a family of products, this makes the final product more expensive because of an increasing number of developers who build the software and maintain it over time, with the counterpart of possibly introducing new bugs in the software of these products due to the repetitive implementation of the features that define it, instead of reusing and modifying the existing ones. By automating these processes, human error is greatly reduced, and production costs are lowered.

This project aims to help software developers, particularly, to those who do Model-Driven Engineering (MDE). More specifically, through Program Synthesis (PS) techniques, assist in the tasks that an MDE engineer carries out. We then present a survey covering the state-of-the-art at the intersection of MDE and PS. Finally, we propose an application of Machine Learning (ML) as a synthesis technique in combination with models. Thus, an ML-based assistant, capable of extracting the patterns behind the relationships between model elements, has been developed.

Keywords: Model-driven Engineering (MDE), Program Synthesis (PS), Machine Learning (ML)

1. Introduction

This project is aimed at helping engineers who make software, more specifically, people who do MDE (i.e., those engineers who use models), by improving their workflow.

First, it was necessary to see how things stood to help them, that is, what our starting point was. We discovered that Program Synthesis (PS) is the one thing that could assist in the tasks that an MDE engineer carries out. But that fact alone was not enough. So, we conducted a survey combining models and synthesis. Once we finished it, we detected that the synthesis part has hardly been done, as there are just a few studies in this area.

One of the essential parts of modeling is to connect the elements of the models. That task requires the engineer to look at the elements, their properties, and decide how they should be connected.

Our hypothesis was that connections between model elements are made deliberately. Using Machine Learning (ML) techniques, it is possible to extract the patterns behind those connections. Then, the goal was that ML can connect the different elements in the same way as a human would. Accordingly, what we thought we could do, is within what is yet to be done.

Among its utilities, it can be used to teach beginners and to check before committing (as is the case with tests; see Figure 1 - MDE pretended workflow).

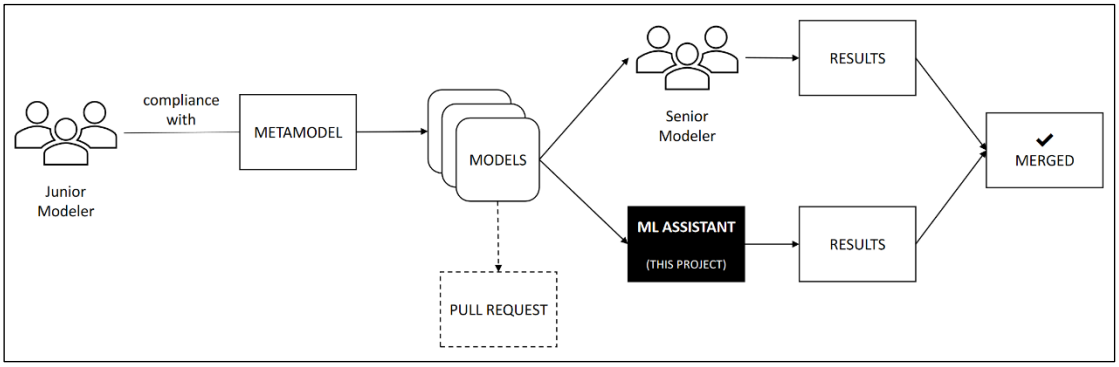

Figure 1 - MDE pretended workflow

Then we developed an ML-based assistant, capable of testing whether the connections between the elements of a model are compliant with those that would be made by a domain expert human.

Therefore, a new path in the model development life cycle is opened. In a world that requires frequent deliveries of new content, it creates the possibility of CI/CD (continuous integration and either continuous delivery or continuous deployment.).

We evaluated our approach in Kromaia video game case study (detailed in Background section). The case study for our work were the game characters at the end of each stage of the video game, i.e., the final bosses [1]. Additionally, to have real data from novice modelers, we conducted a modeling experiment with university students with no knowledge in modeling nor experience with the model editor used during the test, and alien to the application domain (i.e., Kromaia video game).

The source code, along with the input datasets and the generated models ML classifiers and results are open source and available through the links in the footer[1].

[1] Public repositories:

- https://github.com/rocammo/scopus-search-api
- https://github.com/rocammo/kromaia-ai

2. Background

We wanted to help people who do MDE, and where the work of people who do MDE is most automated is in Program Synthesis (PS). Furthermore, Machine Learning (ML) was a key player in the performance of the synthesis technique. For these reasons, the following concepts are relevant for a proper understanding of the project:

Model-Driven Engineering (MDE) [2] aims to facilitate the development of complex systems by using models as the cornerstone of the software development process. Models are built in accordance with a metamodel that embodies the particularities and rules of a specific domain, formalizing what is valid and what is not when building a model for that metamodel. Models are used to formalize a system and capture each of its particularities. Then, those models can be used to reason about the system, perform validations, or transform it into different metalanguages, source code, or even run-time objects.

Program Synthesis (PS) [3] is the task of automatically finding a program in the underlying programming language that satisfies the user intent expressed in the form of some specification. This problem has been regarded as the holy grail of Computer Science since the 1950s, when AI was first introduced. Despite inherent problems such as imprecise user intent and a usually large search space of programs, the field of program synthesis has developed multiple techniques that enable program synthesis in a variety of real-world application domains. It is now widely used in software engineering, biological discovery, computer-aided education, end-user programming, and data cleaning. Several synthesis applications in the field of programming by examples have been implemented in mass-market industrial products over the previous decade.

Machine Learning (ML) is a subfield of Artificial Intelligence (AI) that aims to create strategies that allow computers to learn. It is all about creating algorithms capable of generalizing behaviors and recognizing patterns from information provided in the form of examples. Hence, it is a knowledge induction process, i.e., a method for obtaining a general statement from case-specific statements through generalization.

Likewise, it is worth mentioning the case study that we have taken for the evaluation of the ML-based assistant developed:

Kromaia is a commercial video game released worldwide in both physical and digital versions for Steam and PlayStation 4. It has been translated into 8 languages: Spanish, English, French, Italian, German, Japanese, Russian, and Portuguese.

2.1. Model Driven Engineering

In the case of Kromaia, models are built against the Shooter Definition Model Language (SDML), a Domain-Specific Language (DSL) model for the video game domain created by Kraken Empire, which is the company that developed Kromaia. SDML allows for the definition of every element that will be present in the game, including worlds, vehicles, creatures, missions, enemies, etc. Specifically, SDML defines aspects included in video game entities [4]:

- The anatomical structure, including which parts are used in it, their physical properties, and how they are connected to each other.
- The amount and distribution of weapons, and defenses in the structure of the character.
- The movement behaviors associated to the whole body or its parts.

This modelling language has concepts such as hulls, links, weapons, and AI components.

2.2. Program Synthesis

It has long been a goal of computer science to automatically generate programs from declarative specifications.

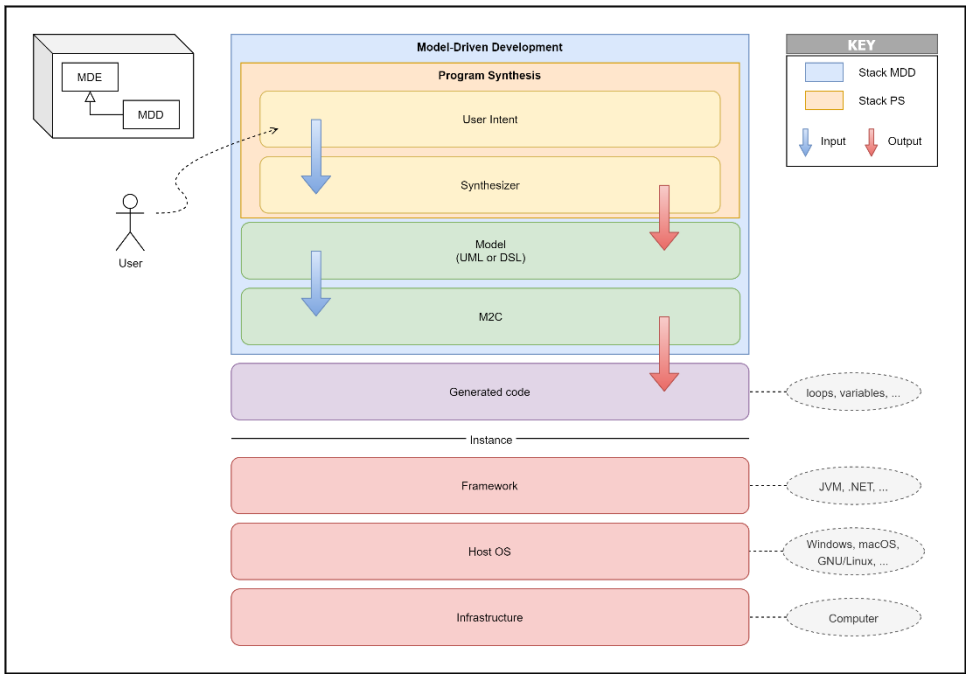

Figure 2 - Model-driven engineering (MDE) and Program synthesis (PS) stack

Three key dimensions define a synthesizer [3]: the types of constraints it accepts as expression of user intent, the program space it searches, and the search technique it utilizes.

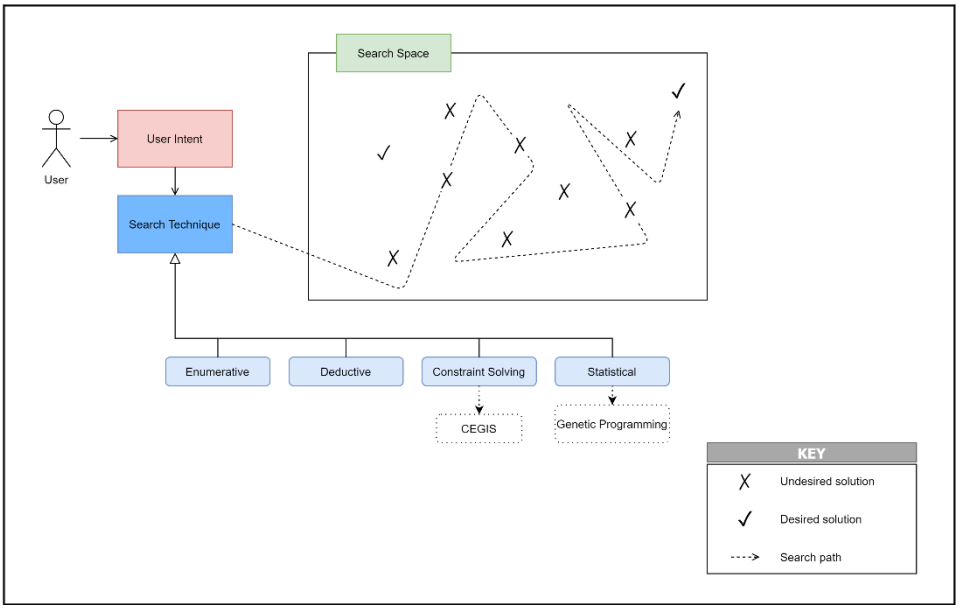

Figure 3 - Dimensions in Program Synthesis

The synthesized program can be shown to the user explicitly for debugging, re-use, or integration into a broader workflow. However, the synthesized program may be implicit in other circumstances and is simply employed to automate the user's intended one-time task.

2.2.1. User Intent

The user intent can be conveyed in a variety of ways, such as logical specifications, examples, traces, plain language, incomplete programs, and even linked programs. Depending on the underlying task and the user's technical background, a particular option may be more suited in each circumstance.

A *logical specification* defines the relationship between a program's inputs and outputs. It can be used as a concise and exact functional definition for the intended program. Complete logical specifications, on the other hand, are frequently difficult to construct. End users who are not programmers may find *examples* more engaging and natural.

A *trace* is more extensive than an input-output example because it shows how a certain input should be translated into the appropriate output rather than just explaining what the output should be. The intermediate states resulting from the user's sequential actions on a user interface provide a valid trace, which is an effective model for programming by demonstration systems. Traces are preferable to input-output examples from the synthesizer's standpoint since they contain more information. Demonstrations may be more taxing than providing input-output examples from the user's perspective.

In certain circumstances, the best way to specify the intent is through the *program* itself. This is straightforward in some cases, such as super optimization, de-obfuscation, and synthesis of program inverses, where the specification is the program to be optimized, de-obfuscated, or inverted. Users may find it easier to define the specification as an inefficient program rather than a logical relation, even for applications like the discovery of novel algorithms.

2.2.2. Search Space

The search space should strike a good balance between expressiveness and efficiency. On one hand, the space should be large/expressive enough to include a large set of programs for the underlying domain. On the other hand, the program space should be constrained enough to allow for efficient search and reasoning, and it should cover a domain of programs that can be efficiently reasoned about.

The program space can be limited to a subset of an existing programming language (general purpose or domain-specific) or to a domain-specific language that has been specially constructed

(DSL). At least two features can be utilized to qualify the space of programs: (i) the program's operators, and (ii) the program's control structure. The program's control structure may be limited to a looping template provided by the user, a partially completed program with holes, straight-line programs, or a guarded statement set with control flow at the top.

2.2.3. Search Technique

Enumerative search, deduction, constraint solving, statistical approaches, or a mixture of these can be used in the search technique.

Enumerative

Enumerative search enumerates programs in the underlying search space in some sequence and verifies whether each program fulfills the synthesis constraints for each program. While this may appear to be a basic method, it is frequently quite powerful. A rudimentary version of enumerative search rarely scales. Many practical systems that use enumerative search innovate by providing different techniques for trimming or sorting the search space.

Deductive

The typical divide-and-conquer methodology is used in the deductive top-down search, with the main notion being to recursively reduce the challenge of synthesizing a program expression to smaller sub-problems.

While enumerative search is bottom-up (i.e., it enumerates smaller sub-expressions before enumerating larger expressions), the deductive search is top-down (i.e., it fixes the top-part of an expression and then searches for its sub-expressions).

Constraint Solving

Constraint generation and constraint resolution are the two primary processes in constraint solving techniques.

The process of creating a logical constraint whose solution yields the desired program is referred to as constraint generation. Making a logical restriction like this usually entails assuming about the unknown program's control flow and then encoding that control flow in some way. It

is combined with a *counterexample-guided inductive synthesis strategy (CEGIS)*, detailed in 2.2.4, to obtain a higher level of efficiency.

Solving the constraints generated by the constraint generation step is known as constraint solving. Second-order unknowns and universal quantifiers are common under these restrictions. A common technique is to reduce second-order unknowns to first-order unknowns, eliminate universal quantifiers, and then solve the resulting first-order quantifier-free constraints with an off-the-shelf SAT/SMT solver, as described in 2.2.4.

Statistical

Machine learning of probabilistic grammars or genetic programming, for instance, are two examples of statistical methodologies.

Machine learning approaches can be used to supplement other search approaches based on enumerative search or deduction by offering likelihood of various choices at any choice point. The likelihood probabilities can be based on cues discovered in the user-provided input-output examples or other inputs. These functions are derived from training data and learned offline.

Genetic programming is a program synthesis method inspired by biological evolution. It entails maintaining a population of individual programs and utilizing computational analogs of biological mutation and crossover to generate program variants. Crossover allows useful parts of code to be shared between evolving programs, whereas mutation produces random changes. A user-defined fitness function is used to assess the suitability of each variant, and successful variants are chosen for further evolution. The fitness function is critical to the success of a genetic programming-based system. In imperative programs, genetic programming has been used to find mutual exclusion algorithms and to solve defects.

2.2.4. Oracle-Guided Inductive Synthesis (OGIS)

When they are simple to create, templates are effective at decreasing the complexity of program synthesis. However, in most circumstances, domain-specific templates are unavailable. In many cases, the community has developed a simple alternative to program synthesis. It is founded on the observation that, whereas *synthesizing* a program that fulfills a certain condition is a second-order problem that may be infeasible, *verifying* whether a given program fulfills that condition is a first-order problem that is often more straightforward.

In OGIS, the solver looks for a program candidate using a simplified specification (as stated above), then queries an oracle to determine whether the candidate is valid. Different oracles are used in different OGIS flavors. The most popular variation of OGIS, *counterexample-guided inductive synthesis (CEGIS)*, requires a correctness oracle: for a given candidate program, the oracle may return "YES" if it meets the requisite specification, or "NO" with a counterexample if it does not.

In the inductive generalization and validation phases, CEGIS-based systems heavily rely on *SAT/SMT solvers* [5] to generate test inputs and find candidate solutions, respectively.

2.3. Machine Learning

One of the most common uses of Artificial Intelligence is Machine Learning (ML). A machine learns to do tasks based on the data it gets. Its performance improves as it gains experience. Supervised, unsupervised, and reinforced learning approaches are all included in ML. However, this project just focuses on *supervised learning*.

The system is trained with a labeled dataset in Supervised Learning. For a dataset to be considered labeled, the data (input) must be labeled with the correct result (output).

In real-world computational challenges, supervised machine learning is extremely useful. The system learns from labeled training data to predict outcomes for unforeseen data. Models can be retrained throughout time to ensure the integrity of the insights.

2.3.1. Classification

Within supervised learning, there are also different types. In this project, we have focused on *classification*.

Classification involves grouping the data into classes. Binary classification occurs when a supervised learning algorithm labels input data into two separate classes. Multiple classifications refer to categorizing data into more than two classes.

There are also different types of classifiers or classification algorithms. Three have been used: *Decision Tree*, *Naïve Bayes*, and *Random Forest*.

Naïve Bayes classifier

The Naïve Bayes (NB) classifier is useful for large datasets classification. A directed acyclic graph (DAG) [6], a type of directed graph that has no cycles and goes in one direction only, is used to assign class labels.

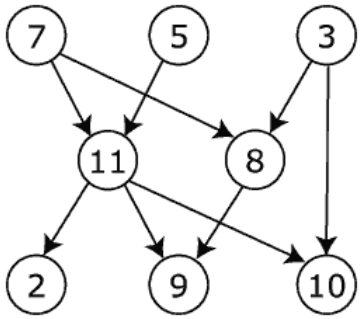

Figure 4 - Simple DAG representation
[Source: Wikipedia]

Decision Tree classifier

A decision tree is a flowchart-like model that includes conditional control statements, decisions, and their probable outcomes. The output relates to the labelling of unforeseen data. The leaf nodes in the tree representation correspond to class labels, whereas the inside nodes represent attributes.

Random Forest classifier

The Random Forest classifier is an ensemble method. It operates by constructing a multitude of decision trees and outputs a classification of the individual trees.

3. State of the Art

Before proposing an approach or proceeding with next steps, it was essential to correctly identify the research gap. In this chapter we have carried out a systematic survey of the literature.

In this regard, we first proceeded with a taxonomy of existing related work. For a more in-depth level of detail, please see Annex 4: Taxonomy of the Related Work.

3.1. Taxonomy of the Related Work

For this task, it is necessary to make a search, being important how that search is made – *Google Search* is not the ideal way. Somehow, it is necessary to index the papers that in one way or another can be related to the terms 'Model Driven Development' and 'Program Synthesis'.

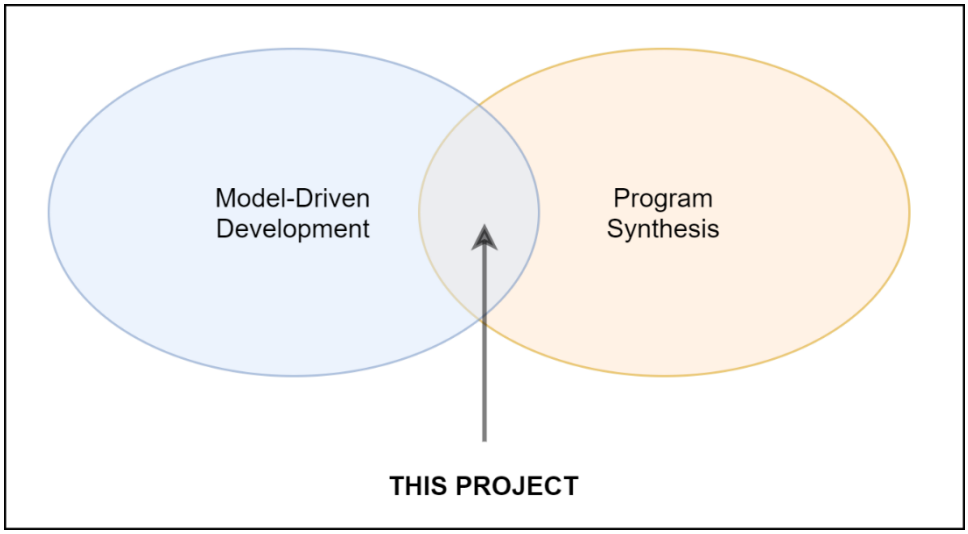

Figure 5 - Scope of this project

The recommendation here is to use *Scopus*, the largest multidisciplinary database of bibliographic references and citations, of international scope and with coverage of references cited since 1996. This utility has a web page with free access for universities.

Among the advantages of Scopus, besides the enormous number of documents it indexes, it also has an advanced search system. They have designed a search language defined by boolean and

proximity operators, wildcards, braces, or quotation marks, to filter the results in an optimal manner.

In my case, the shape of my searches looked like this:

```
TITLE-ABS-KEY ("Model Driven Development" AND "Program Synthesis") SUBJAREA (COMP)
```

In parts, the query above means:

- **`TITLE-ABS-KEY ("Model Driven Development" AND "Program Synthesis")`**
  The TITLE-ABS-KEY search takes the TITLE+ABSTRACT+KEYWORDS fields as whole, making those 3 fields into just one and then running a text search. In this case, every document should contain the terms: "Model Driven Development" AND (^) "Program Synthesis".
- **`SUBJAREA (COMP)`**
  Entering SUBJAREA (COMP) will return documents classified under the subject area of Computer Science.

Consequently, a quality indicator is chosen to further reduce the search space. This indicator, taken form *Google Scholar*, would try to filter out documents that are not among the top venues for *Software Systems* related publications.

	Publication	h5 index	h5 median
1.	ACM/IEEE International Conference on Software Engineering	74	111
2.	Journal of Systems and Software	61	90
3.	Information and Software Technology	59	90
4.	ACM SIGSOFT International Symposium on Foundations of Software Engineering	53	78
5.	Empirical Software Engineering	53	75
6.	IEEE Transactions on Software Engineering	52	77
7.	ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)	48	76
8.	ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)	46	78
9.	IEEE/ACM International Conference on Automated Software Engineering (ASE)	45	75
10.	IEEE Software	44	90
11.	Symposium on Operating Systems Principles	42	77
12.	Software & Systems Modeling	41	55
13.	Mining Software Repositories	40	52
14.	International Conference on Software Analysis, Evolution, and Reengineering (SANER)	40	48
15.	International Symposium on Software Testing and Analysis	36	61
16.	International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)	33	54

17.	IEEE International Conference on Software Maintenance and Evolution	33	46
18.	Proceedings of the ACM on Programming Languages	31	46
19.	Software: Practice and Experience	30	36
20.	ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)	29	44

Table 1 - Top venues for Software Systems related publications
[Source: Google Scholar]

To review the searches that are made to Scopus database, see Annex 3: Scopus queries. There, you will see that some queries have been discarded due to the large number of results, as it would require too much time to classify all the documents. A query that resulted in more than a thousand documents is rejected.

Of the queries that have not been discarded, the highest number of documents is 340. Although this is not a very high number, 340 documents are still a lot of documents, considering that they are intended to be classified manually.

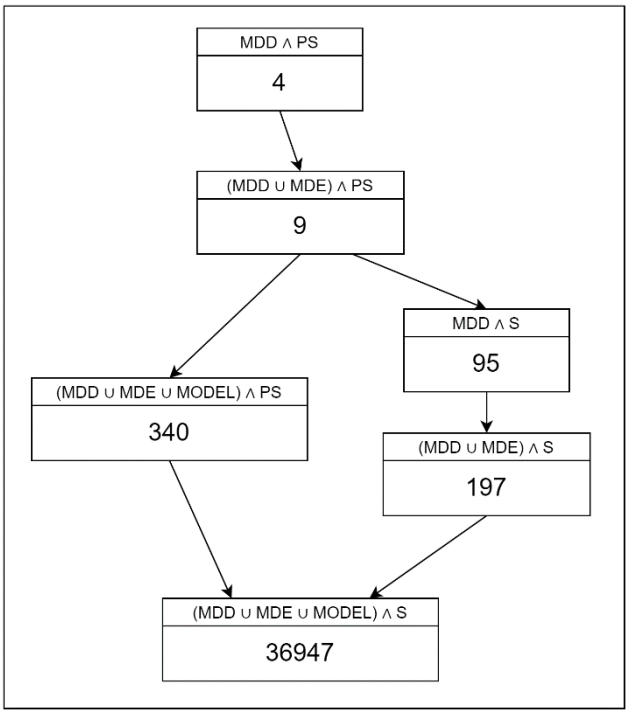

Figure 6 - Search's decision path

So, I start with an ideal Q1 query, looking for documents that contain exactly the terms "Program Synthesis" and "Model Driven Development" or, failing that, their acronym "MDD". Unfortunately, the search results in only 4 documents, which, applying the quality filter based on where it was published, results in a selection of 0 documents. This happens again with the next Q2 query following the ascending restriction order, resulting in 0 documents as well.

It is not until the Q4 query that a total of 95 documents are obtained, which after applying the filter, results in 4 documents only.

Despite having already results, the number of documents obtained is not enough. With only 4 documents, it cannot be considered a taxonomy per se of the work in this field that has already been published.

It is necessary to continue expanding the search spectrum to eventually reach an optimal number that is sufficiently representative to be considered a suitable candidate for a taxonomy of the related work.

Having reached this point, I rethink how I am filtering the searches. Until now, since the number of results was relatively small, I could afford to do a handmade analysis. But, from now on, the results of my searches contain 197, 340, 36947 documents, unviable values to be handled by a human, considering the limited time scope of this project.

The probability of making any mistake is very high — e.g., letting valid documents escape. So, I decide that I am going to write a utility that will help me with this task.

Elsevier, the company behind Scopus, has a developer portal with documentation, several guides and use cases on how to get programmatic access to the Scopus database – what is interesting to me –, among other products and solutions Elsevier has (e.g., ScienceDirect or Embase). They expose a REST API and a Python SDK for developers, too.

The SDK options are too wide for my use case, so I chose to just make a request to the specific REST API endpoint I needed [7].

You can check out the code through the following url: https://github.com/rocammo/scopus-search-api, so that you can reproduce the searches and their respective filtering.

Keep in mind that the code will always be kept up to date, since below it calls the Elsevier REST API for Scopus, the document results could be updated in the future.

It is time to test the new algorithm. The Q3 and Q5 queries, 340 and 197 document results respectively, seem to be the perfect candidates for this. After the filtering phase, I am finally getting promising results. In the case of the Q3 query, a total of 16 results are obtained. In the case of the Q5 query, a few less are obtained, namely 4 results. Also, all previous queries (i.e., Q1, Q2 and Q4) are repeated and re-performed using the new automation.

```
Query
TITLE-ABS-KEY ("Program Synthesis") AND TITLE-ABS-KEY ("Model Driven
Engineering" OR "MDE" OR "Model Driven Development" OR "MDD" OR "Model") AND
SUBJAREA (COMP)

Step 1
Indexing  |████████████████████████████████████| 340/340

Step 2
Filtering |████████████████████████████████████| 340/340
```

Figure 3 - Output produced after running a query over the algorithm.

Surprisingly, there is no difference in the number of results between Q4 and Q5 queries – 95 and 197 document results respectively –, after filtering has been applied. For both queries, only 4 document results are suitable. Not only is the number of results identical, but also the document results are the same.

It was very tempting to test the algorithm with the Q6 query, with a total of 36947 results, and see how many documents end up being candidates for evaluation, at least after the first screening with the quality indicator that has been defined.

But, after running it, the algorithm stopped without being able to continue at exactly 5000 results. This is a limitation of the API that it has been dealing with since the beginning, and that is that in the usual operation it only allows to bring results in chunks of 25 documents.

To alleviate this restriction, the solution has been to iterate the queries by applying a 25-position offset, which has proved to be effective enough to reach a viable number of documents with which to perform a sufficiently representative related work taxonomy.

What happens is that by doing a high number of searches on a certain query in a short time, the infrastructure surrounding the REST API that Elsevier exposes considers that a DoS attack is taking place, so it discredits my IP and for a certain time I am penalized.

In short, this table below summarizes the outputs generated by the algorithm for searching, indexing, and filtering documents according to criteria:

Query Id.	*Document results*	*Filtered document results*
Q1	4	0
Q2	9	0
Q4	95	4
Q5	197	4
Q3	340	16

Table 2 - Summary of filtered and non-filtered document results per query (sorted by ascending number of results)

It should be noted that the results obtained are the same, both in the 95 and 197 document results (i.e., 4 in total).

In addition to the document results listed in the table above, 3 more documents obtained through a manual query were added to the related work selection list that, despite not respecting the automated selection criteria, they are closely related to the topic, and I think may have some interest to be analyzed.

If possible, I would also like to emphasize the usefulness of the tool that has been developed, which although simple, has been of great help. Without something similar that would automate

the screening of the results, it would have been impossible to go beyond 95, or at most, 197 documents. Thanks to this, it has been possible to overcome this barrier and go beyond.

Finally, the resulting table that brings together all the documents is as follows:

No.	Name	Journal	Year
1.	Constraint-driven development	IST	2008
2.	Tool support for the rapid composition, analysis, and implementation of reactive services	JSS	2009
3.	Constraint-based specification of model transformations	JSS	2013
4.	Feature Modularity in Software Product Lines	SPLC	2006
5.	Program Refactoring, Program Synthesis, and Model-Driven Development	LNCS	2007
6.	Feature Oriented Model Driven Development: A Case Study for Portlets	ICSE	2007
7.	Provenance-guided synthesis of datalog programs	POPL	2020
8.	From typestate verification to interpretable deep models (invited talk abstract)	ISSTA	2019
9.	Synthesis and machine learning for heterogeneous extraction	PLDI	2019
10.	Accelerating search-based program synthesis using learned probabilistic models	PLDI	2018
11.	FlashExtract: A framework for data extraction by examples	PLDI	2014
12.	Automated feedback generation for introductory programming assignments	PLDI	2013
13.	TRANSIT: Specifying protocols with concolic snippets	PLDI	2013

Table 3 - Related work selection to be evaluated.

As can be seen, the resulting list of documents clearly does not add up to 4 *(95-197)* + 16 *(340)* + 3 *(extra)* = 23 *(total)*. This is because after having analyzed them (refer to the appendix INSERT), some have been discarded because the search algorithm developed does not check the key terms semantically. Consequently, although very well some of them talked about models, they did not refer to MDE/MDD techniques, but to CAD models, just to give an example.

Ultimately, after subtracting the discarded documents (i.e., 9 documents), the subtotal is **14 related work papers analyzed**.

3.2. Identification of the Research Gap

At this point, we have a broad background in Model Driven Engineering and Program Synthesis. Likewise, we also have a sufficiently representative sample of the state of the art in this field. Now it is time to study all the related work gathered in the previous step.

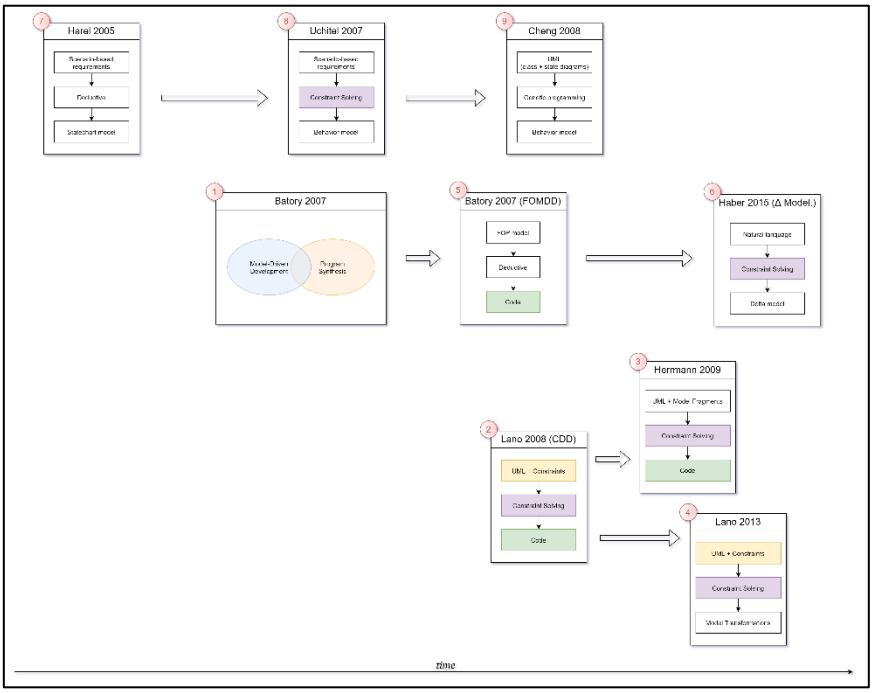

Figure 7 - State of the Art roadmap

Back in 2007 (number 1 encircled in red in Figure 7 - State of the Art roadmap), Batory identified overlapping ideas between Model-Driven Development (MDD) and Program Synthesis (PS) and proposed a theory to unify them called **architectural metaprogramming**. This theory was based on the idea that programming and design is a computation, where programs are values and functions (a.k.a. transformations) that map programs to programs. This theory emerges from **Feature Oriented Programming** (**FOP**). FOP is a software paradigm where programs are synthesized by composing features, where features are either metaprogramming constants or functions. This is how **AHEAD**, an implementation of FOP, works.

There is a category with those works whose objective of the synthesis is to generate models. Within this category, there are those that attempt to generate behavioral models and those that attempt to generate variability models.

Among those that try to generate behavioral models, there are those whose user intent is given by means of natural language.

In 2005 (number 7 encircled in red in Figure 7 - State of the Art roadmap), Harel et al. proposed to tackle model construction as a PS problem. The objective of the synthesis was to generate **behavioral models**. They used **requirements** expressed in natural language as user intent. The search technique of the synthesis was *deductive*, where the key idea is to recursively reduce the problem of synthesizing a program to simpler sub-problems. They were able to develop a prototype capable of generating behavioral models that can then be supported and executed by UML tools.

Two years later, in 2007 (number 8 encircled in red in Figure 7 - State of the Art roadmap), Uchitel et al., addressed the same problem as Harel et al. They had the same synthesis objective and used the same user intent. In fact, they took as a baseline a book from 2003 [8] by Harel. However, the difference between the papers was the synthesis search technique. In this case, *constraint solving* was used. They conducted case studies to validate their approach. The key to success of the approach is in providing adequate support for model elaboration, starting from partial models synthesized from a few scenarios and properties.

Only two papers ranging from requirements to models have been identified. Perhaps, one explanation is that researchers are using **Natural Language Processing** (**NLP**) techniques as in [9], rather than continuing to explore synthesis capabilities, as it seems that there have been more advances with NLP to go from requirements to models recently, than with synthesis.

Still within those that try to generate behavioral models, there are those whose user intent is given by means of UML class and state diagrams.

In 2008 (number 9 encircled in red in Figure 7 - State of the Art roadmap), Cheng et al. tackled the synthesis of behavioral models from UML class and state diagrams taken as user intent. The search technique in this case was based on statistical techniques, specifically *genetic programming*. They illustrated their approach using GridStix, an adaptative flood warning system. In doing so, they can address uncertainty of high-assurance applications that rely on dynamically adaptative systems by predicting the future execution environment and using functional and non-functional trade-offs to respond to environmental changes.

Returning to those who attempt to generate models of variability, their user intent is expressed in natural language.

In 2015 (number 6 encircled in red in Figure 7 - State of the Art roadmap), Haber built on Batory's ideas to introduce a method to synthesize **delta languages**. Delta languages represent variability by explicitly capturing system changes. As an expression of user intent, it uses *natural language* to arrive at a textual base language definition of delta modeling. As a search technique, the synthesizer uses *constraint solving*. Haber uses a comparative case study to evaluate the method, which compares existing originally handwritten delta languages to automatically generated ones and to the extended delta languages using well-defined metrics, demonstrating that both are equally semantically expressive.

There is another category with those papers whose synthesis objective is to generate implementations (i.e., code).

Among those that try to generate implementations, there are those whose user intent is expressed as a composition of fragments.

In 2007 (number 5 encircled in red in Figure 7 - State of the Art roadmap), Trujillo et al. materialized these previous ideas in a new software paradigm called **Feature Oriented Model Driven Development** (**FOMDD**). FOMDD is a blend of FOP and MDD where programs are synthesized by composing features to create *FOP models* that act as user intent expression, and then transforming these models into executables using a *deductive* search technique. The evaluation was conducted around web portal components (a.k.a. portlets). The authors claim that they validate the correctness of the abstractions, tools, and specifications, as well as optimize portlet synthesis.

In 2009 (number 3 encircled in red in Figure 7 - State of the Art roadmap), Herrmann extended the work previously done by Lano with the idea of using **model fragments** as an expression of user intent. His motivation comes from the idea that it is easier to assemble model fragments than to start from models from scratch. Herrmann succeeded in demonstrating that this idea was feasible in the domain of home automation and learning platforms that make use of location-aware services.

Still within those that try to generate implementations, there are those whose user intent is given by means of UML.

Almost simultaneously with Batory, in 2008 (number 2 encircled in red in Figure 7 - State of the Art roadmap), Lano was the first to combine PS and MDD using UML as user intent expression. Lano proposed to leverage constraints in models to achieve synthesis, thus introducing the concept of **constraint-driven development** (**CDD**). In this way, the synthesizer, through *Constraint Solving* as a search technique, can derive into system implementations. Substantial examples of development using the CDD approach have been carried out, including fault-tolerant and real-time versions of a robot production cell and commercial web applications.

There is another category with those papers whose synthesis objective is to generate model transformations. Again, UML is used as expression of the user intent.

In 2013 (number 4 encircled in red in Figure 7 - State of the Art roadmap), Lano again, continuing with CDD, his focus shifted from synthesizing implementations to synthesizing **model transformations**. Model transformations are so central to the modeling community that they are even considered as the heart and soul of MDD. Lano evaluated the transformation using a test suite and demonstrated its efficiency.

There are two recent surveys in PS published in 2017 [3] [5]. These surveys identify about 150 works during years 1962 to 2017. However, neither survey identify work at the intersection of PS and MDD. Surveys identify these synthesis techniques combined with code:

- enumerative search,
- deduction,
- constraint solving (e.g., CEGIS),
- statistical techniques,
- or some combination of these;

whereas with models, only constraint solving has been used for the most part.

There seems to be scope for more synthesis techniques to be combined with models as an expression of user intent.

Thus, this project will focus on the meeting point between Machine Learning (ML) as a search technique and SDML models as an expression of user intent. Likewise, the search space will be limited to SDML models belonging to Kromaia video game.

4. Objectives

The main objectives set for this project, which are included in the project proposal [Annex 1: Project proposal], are:

1) Identification of the research gap and proposal of an approach.

2) Design and implementation of the approach.

3) Choice of baseline and case study.

4) Evaluation comparing with baseline.

5) Discussion of the results obtained.

5. Methodology

We were not clear what the expected outcome was going to be at the end of the project, so it would be decided based on the information we collected as the project progressed, for instance, from the analysis of the state-of-the-art phase when the research gap was identified. The case study was an unknown at the time this project was proposed. Even at the beginning of the project it was still an unknown.

Also, the project has been simultaneous with other academic and professional circumstances: classes, work, and exams of the subjects of the career combined with my work in Inycom and the collaboration with the SVIT research group. So, a full-time dedication was not possible. The availability to work on this project would change from week to week depending on the external workload at the time.

With these characteristics in mind, we chose **Design science** as the methodology to apply to this project.

Also, note that this project does not have a target client, so it will be my director who will act as a client. Due to its high availability, meetings can be held very frequently and feedback on progress can be obtained.

5.1. Design science

Since it is not a very common methodology in the field of Computer Science or, at least, it is not one of the widely extended in the Agile current in vogue today, such as Scrum or eXtreme Programming (XP), I think it is convenient to explain it first, since its adoption has also been something new for me in this project.

Since the dawn of computer science, computer scientists have been conducting design science research without calling it that. Design science has become a significant aspect in management studies over the previous few decades.

The primary purpose of design science research is to produce knowledge that individuals may use to design solutions for their field problems. Design science is concerned with the process of deciding what is possible and useful.

The design process is a series of expert-led activities that results in a unique product. The artifact allows the researcher to have a better understanding of the problem, since the problem's re-evaluation increases the design process' quality. Before the final design artifact is generated, this build-and-evaluate loop is usually repeated several times.

For this project, this methodology has provided us great flexibility, since it gave us the possibility to start the project without knowing for sure what the expected result will be, allowing – through the analysis of the state-of-the-art – to identify a research gap and, from there, to propose a case study in tandem with the implementation of an approach.

Informally, we planned as we went along and as more information became available. However, this does not mean that there was no planning. Since it was not possible to plan the entire project from start to finish, small plans were made, in which, as progress was made, corrections were made and then iterated again to make another small plan, and so on. Just to mention one, in the first analysis that was made of the approach, we did not consider using several ML algorithms for classification, but one day we realized that it could be useful to try several, so we replanned to extend to more (finally three) to compare their performances.

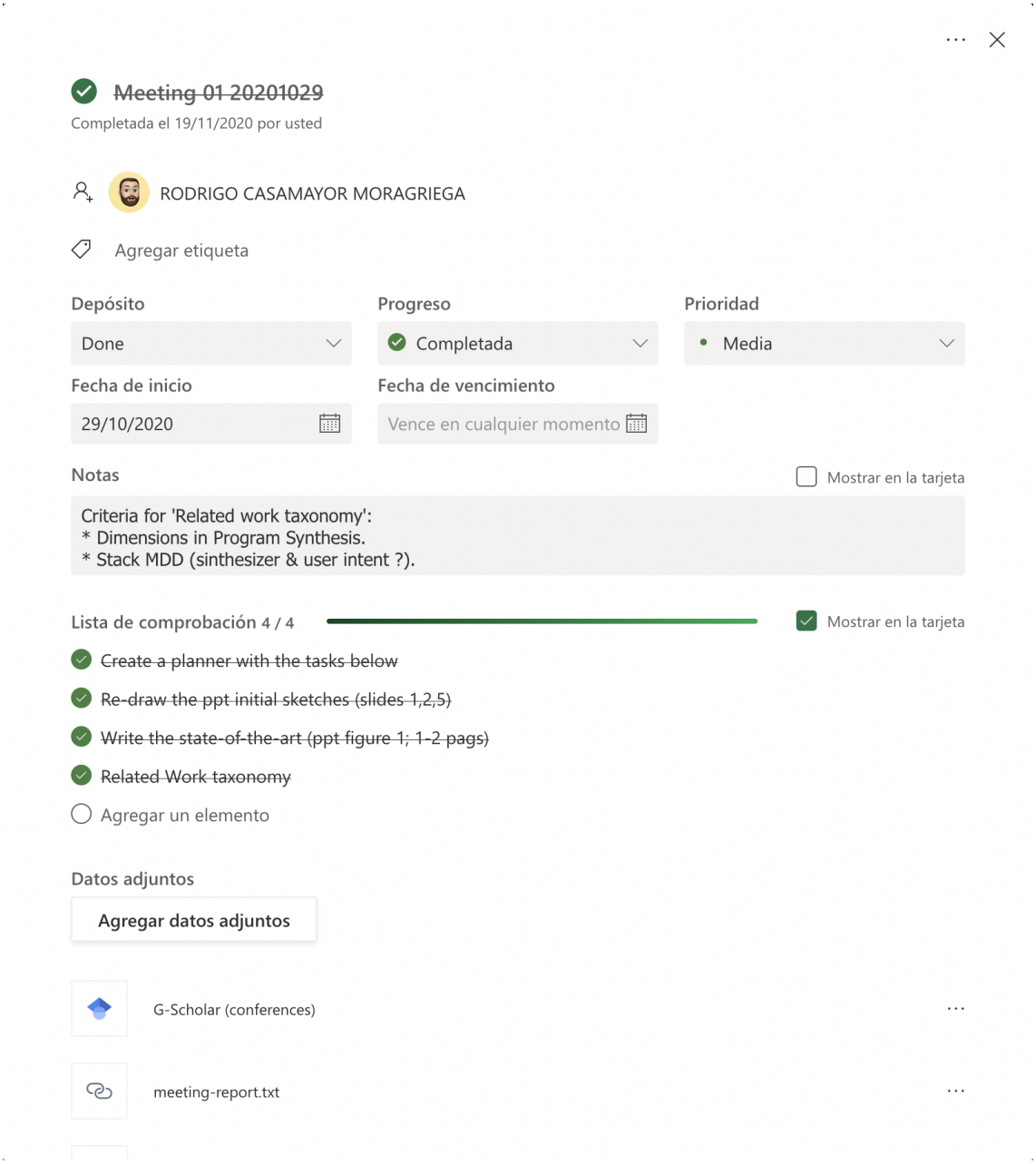

Figure 8 - Screenshot of a task in Microsoft Planner

Minutes were taken after each meeting and the date of the next meeting is set. The minutes can be found in the Annex 2: Meeting notes.

Unfortunately, few projects manage to finish without any deviations, and this one was no exception. To point something out, when we finished the synthesis survey with models, we thought it was going to be much more narrowed down what was to be done, but was very

ethereal, which was a big surprise. This led us to consider how to choose something and make it much more concrete.

Despite not being as we had estimated, we wanted to stick to the scope of the project. There were things we were going to do that we could no longer do, as indicated in the Future work section. But, if with the deviation that we already had we could not do them, with what we knew then, we knew that we were going to deviate even more. So, we took an alternative: instead of using only the models of the final bosses as originally planned, we extended it to all the simple enemies, in addition to the final bosses.

6. Analysis

The purpose is to help developers with the connections of the elements of a model. As stated before, the hypothesis was that connections between model elements are made deliberately, and by means of ML techniques, it is possible to extract the patterns behind those connections and learn from them.

Although this is our baseline hypothesis, there are several challenges to address first: encoding, training and evaluation.

6.1. Encoding

Whatever ML technique is used, at the end of it all we will need to do an encoding. An encoding consists of finding the most representative information so that the ML technique can proceed.

What happens is that, in the case of models, there are no papers that tell you how or in what way to do that encoding, or, at least, we have not found how to do that encoding when we want to learn the connections between elements.

So, we could consider that a human could do it by hand, but we would be condemned to the domain where that human is an expert (in this case, Kromaia video game), so we would be limiting the approach to be generalized and extrapolated to other contexts or application domains. In the same way, if we design an encoding that is too sophisticated, it may change for each domain. Therefore, neither of the two previous options is of interest to us.

Consequently, we have focused on the most optimistic scenario that we can think of, the one that would mean the least resistance. This is, taking the elements that are in the metamodel, i.e., given a relationship, the properties of the elements involved in the relationship.

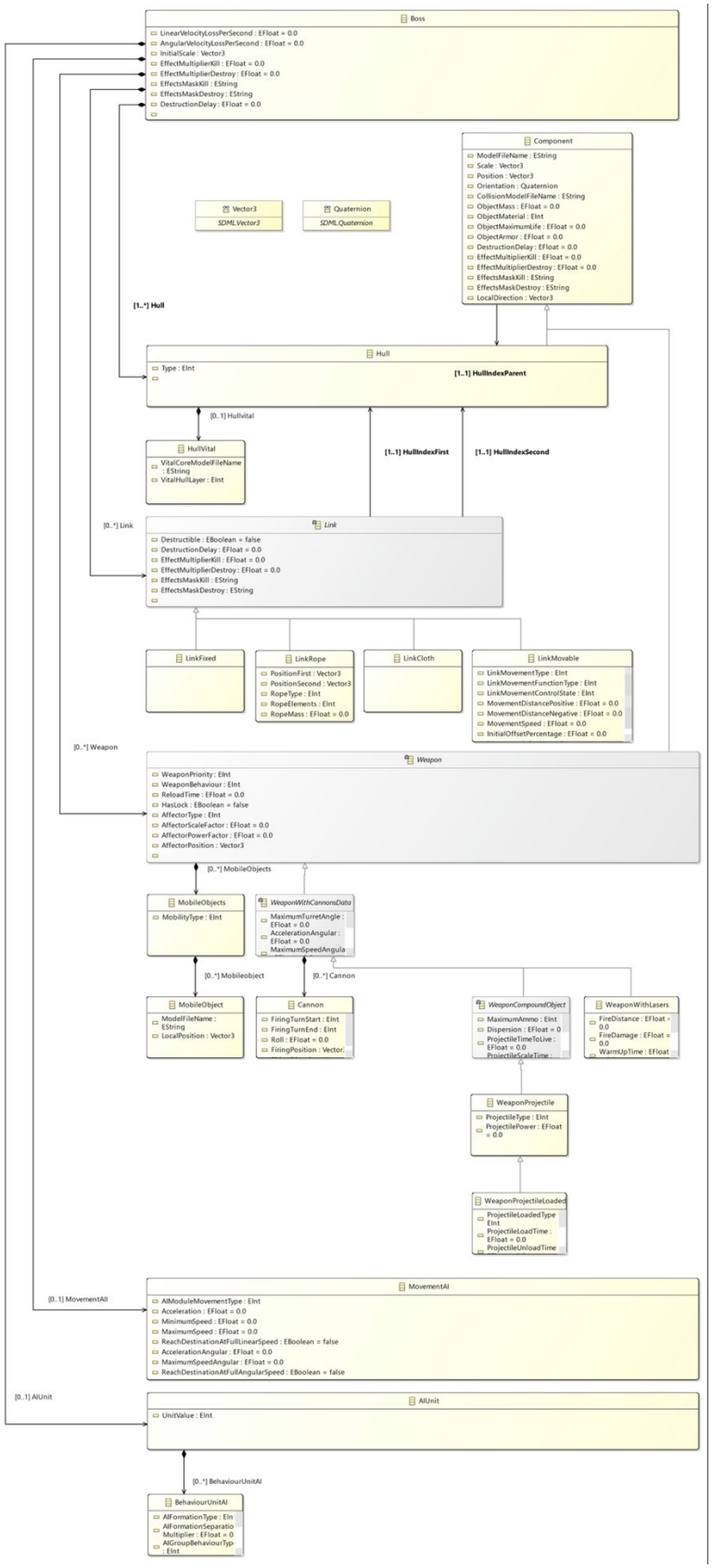

Figure 9 - SDML metamodel from Kromaia videogame (graphical representation)

At this point, it is difficult to know whether the chosen approach will work or not, but we believe it is the ideal one and, therefore, we will start here.

6.2. Training

To do ML, we need training, because we are going to extract patterns and learn from them. And to train, we can get to pull out examples of connections if it is not a case study.

Here is where we run into a big "but", and is that if we intend to classify the relationships depending on whether they are more like a good relationship as the existing ones, extracted from the models made by the developers, or on the contrary they are distant and therefore considered a bad relationship, we also need examples of bad relationships and not only of good relationships as those of the models included in the game. Something so simple, and that apparently could go unnoticed, is a problem, because usually developers do not keep what they would not do.

There is no easy answer to where we were going to get them. We had to find a way that from the good ones, we can figure out how to generate connections that have never been made before.

Another question to be solved was which ML techniques to use, as there are many. It was unfeasible to explore them all because of the wide range of techniques, considering the limited time we had available.

The search was going to be restricted to supervised learning types, specifically classification or ranking techniques. We will try to compare several to see if there are differences or no differences, while maintaining the possibility of being able to exchange one for another.

6.3. Evaluation

It is also crucial to know what and how the evaluation mechanisms will be.

For instance, we thought of a test case that would have an entry in which all were bad relationships and there was only one good relationship, all together, the ML classifier would highlight the good relationship over the bad relationships, putting the good one first.

All test cases were not defined from the first brainstorming that took place in this analysis phase. Additionally, to have real data from novice modelers, we thought about conducting a modeling experiment with university students with no knowledge in modeling nor experience with the model editor used during the test, and alien to the application domain (i.e., Kromaia video game).

7. Design

A machine learning based approach will be designed for classifying the relations. That is, given a model relation, our approach detects which model the relation belongs to. The approach can distinguish between a model made by a human and an artificially made model. To do this, the approach has two phases: training and testing.

In the training phase, a classifier is trained to learn which relations corresponds to each model. To do this, the input consists of a set of models used as datasets. In the testing phase, the classifier is used to classify the set of model relations.

Before looking for the properties of the elements in the metamodel to do the encoding, it is essential to know the two types of relationships that we handle to get the attributes of each of them: association and aggregation or inheritance.

In total, we identified 9 relationships that are present in the metamodel:

- Hull
- Weapon
- Link
- AIUnit
- MovementAI

5 aggregation relationships,

- Link-HullIndexFirst-Hull
- Link-HullIndexSecond-Hull
- Weapon-MobileObject
- Weapon-Cannon

and 4 association relationships.

The training phase consists of two steps: encoding and training. An example is shown below.

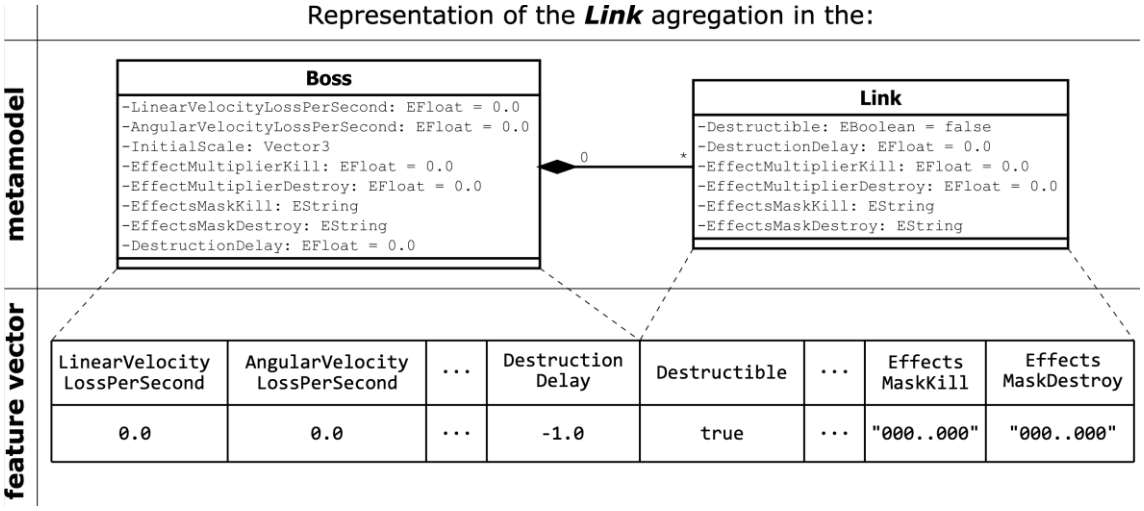

Figure 10 - Representation of the aggregation Link in the metamodel and in the feature vector

In the encoding step, the relations of the models are turned into a feature vector. For each relation, we consider each property of the relation as a feature in the feature vector. The value of each feature is the value that the property has in the model. Fig. Figure 10 - Representation of the aggregation Link in the metamodel and in the feature vector shows an example of the relation named Link and the feature vector that would be generated by this step. Properties of the related classes in the metamodel are features in the feature vector. For example, the property *DestructionDelay* from the *Boss* class or the property *Destructible* from the *Link* class. Moreover, their values correspond to the value of the property in the model. Therefore, the value of the feature *DestructionDelay* is *-1.0*, and the value of the feature *Destructible* is *true*.

All these data are collected in what we call datasets. In total, 9 datasets are generated, i.e., one for each type of relationship. Each entry in a dataset is a relationship of a particular type. Each model adds N relationships, i.e., from 0 to N (0...N).

In the training step, the feature vectors are used to train the classifier, which learns a ruleset through the comparison of the feature vectors.

Regarding ML techniques, finally, three classification algorithms were chosen: Decision Tree, Naïve Bayes, and Random Forest. We chose these three and not others because they are the top 3 most named in the forums we visited around this topic.

Revisiting the question about where to draw bad examples (mentioned in the Analysis section), finally, we invented some mutation operations, in which to inject bad relations with mutations.

7.1. Tools

At this point, we will proceed to detail the different software tools that have been used. To improve your understanding, we will proceed to detail section by section as appropriate.

7.1.1. Programming language

For the development/coding part, **Python** [10] has been chosen as programming language. It should be noted that this language was developed by Guido van Rossum in 1991.

Figure 11 - Python logo

Python is a platform-independent, object-oriented scripting language, prepared to perform any type of program, from Windows applications to network servers or even web pages. It is an interpreted language, which means that you do not need to compile the source code to run it, which offers advantages such as speed of development and disadvantages such as slower speed.

On the other hand, I consider important to highlight the great capacity and simplicity that this language has for artificial intelligence tools, since without them this project would not have been possible to be carried out.

7.1.2. IDEs

An IDE (Integrated Development Environment) is a computer application that provides comprehensive services to facilitate software development for the developer or programmer.

In this case, two different IDEs have been used:

- **Visual Studio Code** [11]
  It is a source code editor developed by Microsoft. It is free and open source.

Figure 12 - Visual Studio Code logo

- **Jupyter** [12]
  This is a web-based interactive computing environment for creating documents that include text and code that can be executed.

Figure 13 - Jupyter logo

7.1.3. Version control

Git [13] is a version control system that records changes made to a file or set of files over time, so that you can retrieve specific versions later.

For this project, **GitHub** [14] has been used as a cloud-based code repository.

Figure 14 - GitHub logo

7.1.4. Libraries

A library is defined as a set of functional implementations, coded in a programming language. In this case, we used the set of libraries included with **SciPy** [15], which are a set of instructions and mathematical tools oriented to Python.

Below, I detail the most relevant ones used in this project:

- **Scikit** [15]
  This is an open source, machine learning oriented library. It has several group analysis algorithms, classification, and regression algorithms.

- **NumPy** [16]
  It is a library that allows the creation of vectors and large multidimensional matrices.
  - **Pandas** [17]
    It is an extension of NumPy which allows to perform big data analysis.

- **Matplotlib** [18]
  It is a library for the generation of graphs from data contained in lists or arrays.
  - **Seaborn** [19]
    It is a library for data visualization based on Matplotlib.

7.1.5. Task management

The purpose of this type of software is to allow a correct development of the projects, in most of its facets. In our case, we used Microsoft's proprietary program, **Planner** [20]. This has allowed us to create plans, generate and assign tasks, share files, and collaborate online.

Figure 15 - Microsoft Planner logo

8. Implementation

8.1. Baseline relationships

The first algorithm is key to the process, since without data we cannot proceed. The algorithm receives a metamodel and one or more models that implement it. As output, the algorithm returns datasets and feature vectors. In both cases, the algorithm extracts one for each relation of the metamodel.

We develop our construction of datasets as outlined in Algorithm 1. It also prepares and instantiates two empty arrays, one that will be used to store the feature vectors and the other to store the datasets, which, later, after having completed the execution of the algorithm, will be returned filled to be used in subsequent algorithms.

Algorithm 1 Dataset construction

```
MM ← MetaModel
M ← Models
V ← []                                   ▷ Empty features vectors
D ← []                                   ▷ Empty dataset

for RR_i ∈ RR ← GetRelationshipsFrom(MM) do
    V_i ← BuildFeaturesVectorFrom(RR_i)
    d ← []
    for M_i ∈ M do
        R ← GetRelationshipsFrom(M)
        for R_i ∈ R do
            d ← FromModelRelToDataset(R_i)
        end for
    end for
    D[RR_i] ← d                          ▷ Add the new individual
end for
```

Figure 16 - Dataset construction (algorithm)

8.2. Mutated relationships

Subjects a given set of properties belonging to a specific dataset to a mutation process. These properties can be mutated by a specific percentage or within a range. This process is repeated N times. The result after having applied the mutation can be derived in:

- value incremented by a percent,
- value decremented by a percent, or
- default value.

Algorithm 2 Mutant creation for training

$D \leftarrow Dataset$
$V \leftarrow FeaturesVectors$
$percentage \leftarrow [min, max]$ or n
$times \leftarrow N$

for $D_i \in D$ **do**
    $D_i \leftarrow mutation(D_i, V_i, percentage, times)$
end for

Figure 17 - Mutant creation for training (algorithm)

8.3. Ranking

For model training, it is used a technique called cross-validation, in its 10-fold variant (see section 9.1). In combination with this technique, three different classifiers are tested: Decision Tree, Naïve Bayes, and Random Forest. Each one is trained and validated in the order of 1000 times, and the one with a higher accuracy is kept.

Algorithm 3 Model classification (10-fold Cross-validation)

```
1:  D ← Dataset
2:  V ← FeaturesVectors
3:  C ← Classifiers
4:
5:  for D_i ∈ D do
6:      CrossValidation(D_i, V_i, C_i, 1000)
7:  end for
8:
9:  function CrossValidation(dataset, features, classifier, duration)
10:     I ← dataset[features].values                    ▷ All inputs
11:     C ← dataset["class"].values                     ▷ All classes
12:     Best ← 0
13:
14:     for i ← 1, duration do
15:         (E_I, E_C, T_I, T_C) ← TrainTestSplit(I, C)
16:
17:         classifier.TRAIN(E_I, E_C)
18:
19:         Current ← classifier.SCORE(T_I, T_C)
20:         if Current > Best then
21:             Best ← Current
22:         end if
23:     end for
24:
25:     return Best
26: end function
```

Figure 18 - Model classification (algorithm)

9. Results

The evaluation design was carried out in two major steps:

1) Mutation of the baseline in-game human-made relationships.
2) Testing to identify the gap between baseline and mutated relationships.

The metamodel present in our case study contains aggregation relationships, as well as association or inheritance relationships (which become aggregation or association, as the case may be) between the elements of the metamodel.

Our objective is to evaluate the effectiveness of ML techniques to detect and learn the patterns behind the relationships so that, later, to have an ML classifier that, given a pattern, determines if it is correct, or, in other words, if it is as a human would do or, on the contrary, not.

The technique we used was ranking. Starting from a relation (for example, a relation linking a Hull to a Weapon), an encoding is used which, as with Evolutionary Algorithms (EA), takes a set of properties. These properties that have been extracted are what is called in ML a feature vector. Machine Learning needs to know what the essential characteristics are of what it is going to learn. It is, therefore, a critical step that determines the success or failure of the learning process.

So, we start from the premise that if given a relationship that joins two elements end-to-end, the features of the feature vector are going to be the properties of each element at the ends.

For example, if we are joining a Hull, which is an element of the model, with an association relationship to a Weapon, the feature vector will be formed by all the features of the Hull and all the features of the Weapon, so that each property will be an entry in the feature vector. Let us imagine that we have a Hull that has a property that is size and another one that is vitality. This Hull has a Weapon associated with it that has a property that is power. In this case, the vector of characteristics would be the size of the Hull, power of the Hull and vitality of the Weapon. With this procedure, we can obtain from all the models, every time that relationship that links a Hull with a Weapon appears, all the properties as they are in the vector of characteristics and learn about it.

Every two elements of the model that are linked by a relationship, all their properties are extracted and deposited in the feature vector. This is the knowledge base that ML must learn from.

But there is a small catch. ML needs "good" examples and "bad" examples to learn to differentiate the good ones from the bad ones. Kromaia's in-game models assume examples of relationships that humans have made, therefore, only "good" examples, since after all they are the ones that are inside the video game and are the ones that work. Therefore, we need to obtain "bad" examples.

Our technique to obtain examples of relationships that are not well done has been through mutations: we take those that are well done and mutate them, that is, we break them. In this way, we manage to have well done examples and poorly done examples, the result after mutations.

Having the data, we take some of the good ones and some of the bad ones, we train the ML classifier.

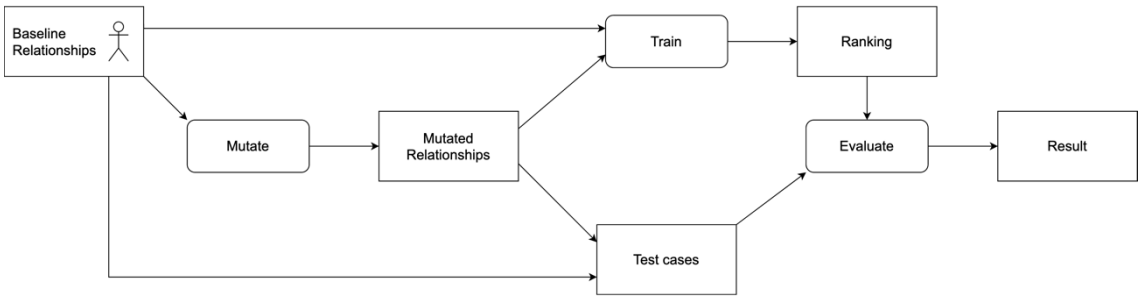

Figure 19 - Flowchart representing the evaluation design.

9.1. Cross-validation

To apply classifier's algorithms in models, we first must split our datasets into random train and test subsets.

The training set is used to train a classifier, which learns a ruleset through the comparison of the feature vectors of the training set [21]. However, before using this classifier to rank the models in the testing process, it is worth analyzing the performance of the classifier through cross-validation.

Cross-validation is a statistical method of evaluating and comparing ML algorithms by dividing data into two segments: one used to train a classifier, and the other used to validate the classifier [22]. Moreover, to reduce variability, multiple rounds of cross-validation are performed using different partitions, and the results are averaged over the rounds [23].

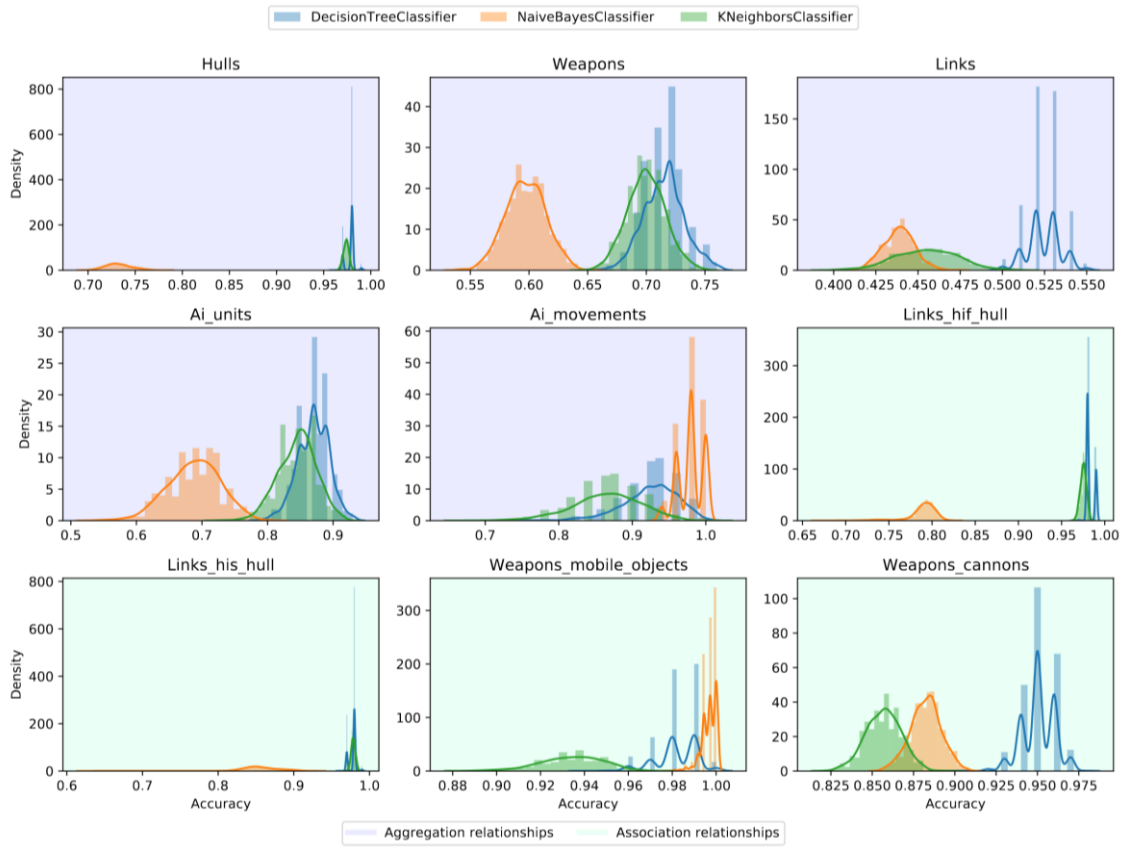

Figure 20 - Mosaic graph with the accuracies of the three classifiers for each relationship

It is obviously a problem that the ML algorithms perform quite differently depending on the subset of the data it is trained on. This phenomenon is known as overfitting.

Overfitting means learning to classify the training set so well that it does not generalize and perform well on data it has not seen before. This problem is the main reason that most data scientists perform *k*-fold cross-validation on their models: split the original data set into *k* subsets, use one of the subsets as the testing set, and the rest of the subsets are used as the training set. This process is then repeated *k* times such that each subset is used as the testing set exactly once.

In this evaluation, a *k*-fold validation with *k* value equal to 10 is used.

Note that the training time is set at 100min (i.e., 1h 40min). If one of the three algorithms were chosen to have only one ML classifier for each relation, it is highly probable that the performance would be superior for that classifier. Also, know that a dataset has an average size of 2.8 MB.

9.2. Modeling experiment with students

The experiment consisted in performing two SDML modeling exercises from a game model editor. The first exercise was about modeling as faithfully as possible an invented enemy of the video game starting from its graphical representation. On the other hand, the second exercise was a free modeling exercise, which extended the modeling generated in the previous exercise.

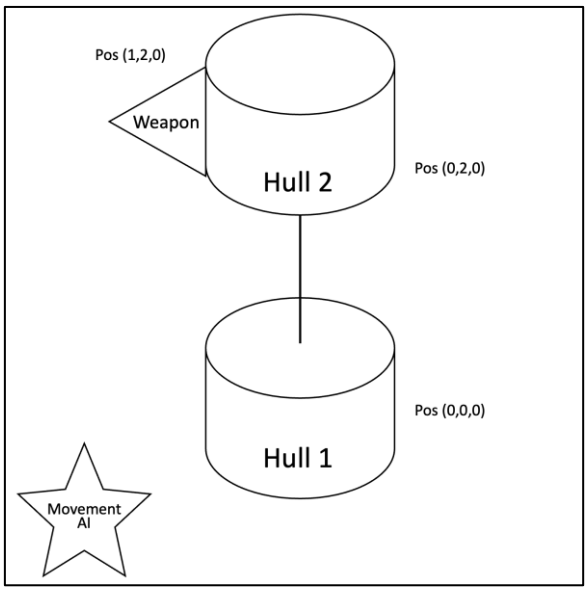

Figure 21 - First exercise (SDML modeling experiment)

The Kromaia video game has two development environments (IDE) that facilitate the generation of new game content for developers. One is the model editor that offers the developer the possibility of generating new models and editing those already created graphically through a tree hierarchy. The other shows a preview of what the model will look like once loaded into the game.

The model editor has a peculiarity, and despite being a model editor with a GUI, the modeling is "blind". That is to say, the modeler is not able to visualize in real time the model he/she is building.

Although the models can be visualized prior to their inclusion in the game, thanks to a model viewer, the model viewer is a separate piece of the model editor, thus slowing down model development by having to export the model from the editor and then import it into the model viewer, thus creating a bottleneck in the developer's creative process.

Figure 22 – Laptops' setup (SDML modeling experiment)

When preparing the experiment, since the number of laptops at our disposal was far from the number of students required to consider a sufficiently representative sample (at least 30 students), we had to divide the experiment into at least two sessions. So, we held a first session on Wednesday, May 19 with students of the degree in computer engineering only, and a second one on Monday, May 24 with students enrolled in the double degree in computer engineering and video game design and development.

In addition, the laptops required previous work on our part, since it was necessary to install the program and its dependencies (Java, Eclipse Modeling Tools, etc.) for the model viewer. In any

case, taking this into account, it would have been unfeasible to have done a joint session even if we had had a larger number of laptops for the time it would have taken us to prepare them all.

To avoid the work afterwards, a shared repository in the OneDrive cloud was used where the students uploaded their models as they finished the exercises. In this way, we were able to make the experiment much more dynamic and the students were able to see their models displayed on the classroom projector.

Figure 23 - SDML modeling experiment with university students

Finally, a total of 31 students participated in the experiment. Since each student performed two modeling exercises, we had a set of 62 models. Thus, a total of 756 relationships were obtained. Of which, separated by type these were:

NAME OF THE RELATIONSHIP	COUNT
AI_MOVEMENTS	58
AI_UNITS	0
HULLS	198
LINKS	134
WEAPONS	97
LINKS_HIF_HULL	134
LINKS_HIS_HULL	134
WEAPONS_CANNONS	1
WEAPONS_MOBILE_OBJECTS	0

Table 4 - Relationships separated by type (SDML modeling experiment)

After running the models developed by the students through the classifier, the relationships categorized as baseline have been (separated by type):

NAME OF THE RELATIONSHIP	COUNT	RELATIONSHIP NO.	SIMILARITY
AI_MOVEMENTS	0	–	–
AI_UNITS	0	–	–
HULLS	4	117	Larva
		129	DaimonAlpha
		151	Orion
		165	Larva
LINKS	0	–	–
WEAPONS	0	–	–
LINKS_HIF_HULL	2	72	Larva
		105	Larva
LINKS_HIS_HULL	1	58	Orion
WEAPONS_CANNONS	0	–	–
WEAPONS_MOBILE_OBJECTS	0	–	–

Table 5 - Baseline relationships separated by type (SDML modeling experiment)

Similarly, relationships categorized as mutated have been (separated by type):

NAME OF THE RELATIONSHIP	COUNT
AI_MOVEMENTS	58
AI_UNITS	0
HULLS	194
LINKS	134
WEAPONS	97
LINKS_HIF_HULL	132
LINKS_HIS_HULL	133
WEAPONS_CANNONS	1
WEAPONS_MOBILE_OBJECTS	0

Table 6 - Mutated relationships separated by type (SDML modeling experiment)

In total, the ML classifier found 7 baseline and 749 mutated relationships. In other words, barely 1% of the relationships resemble a "good" relationship from the models made by the developers. Most relationships have been classified as "bad" (the mutated ones).

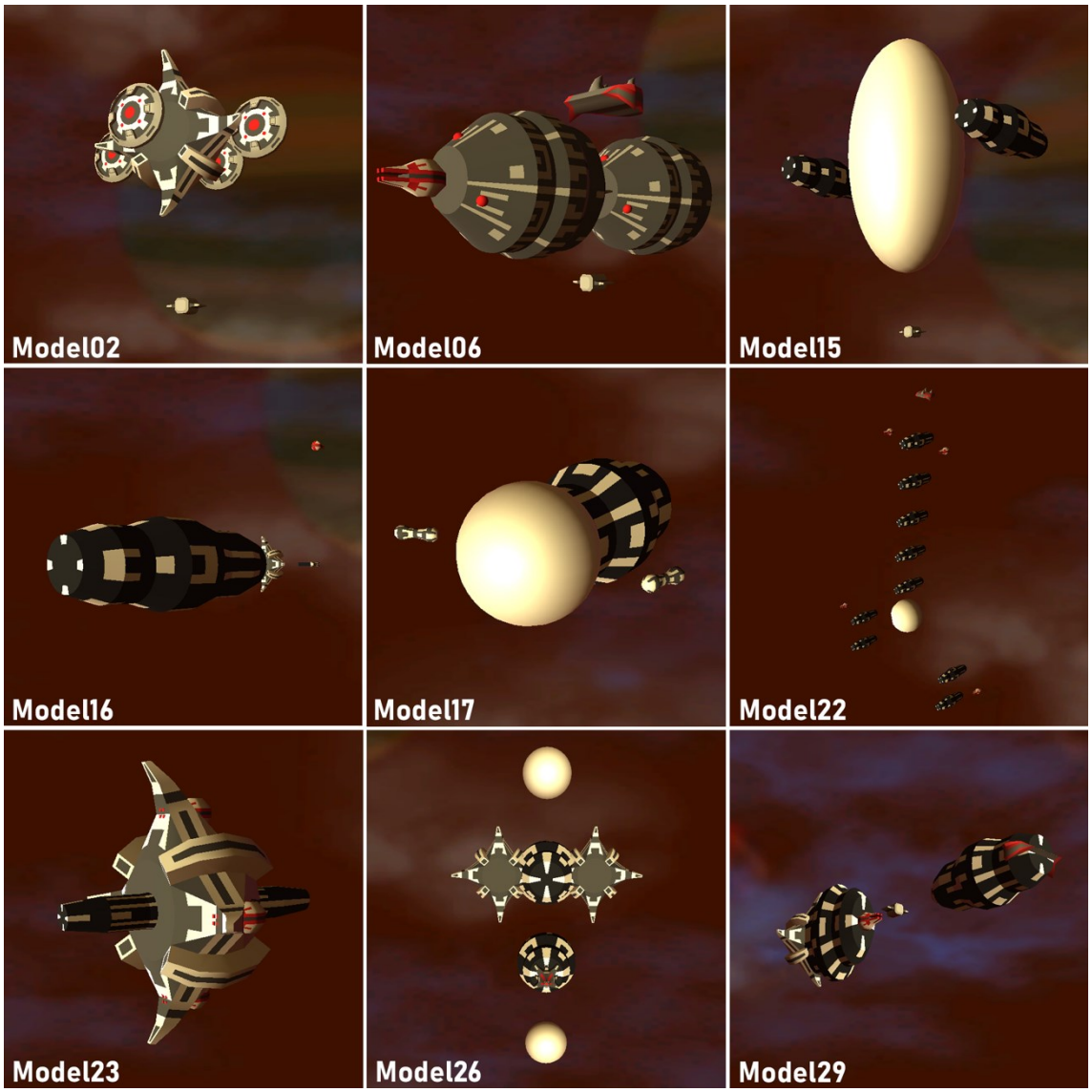

Figure 24 - Featured models made by the students during the experiment

This mosaic above is made up of those models that stand out for their better visual appearance. Named models ending with an even number were made by the students of the second session, and those ending with an odd number were made by students in the first session.

Generally, students enrolled in the double degree in computer engineering and video game design and development were perceived to perform better than those enrolled in the computer engineering degree alone.

Only 7 relationships rated as good seems little but given that the students are not pro modelers and are alien to the peculiarities of the video game and of course to the editor, it seems reasonable.

10. Economic study

To calculate the cost of the project, we will analyze the cost of the tools and resources necessary for its development, the human cost, as well as the necessary infrastructure.

10.1. Cost breakdown

10.1.1. Material costs

According to section 7.1 where the tools used in the development have been described, all the tools described are free to use, even for commercial projects, so no costs are derived from their use.

The necessary training for the development of the project has been acquired throughout the subjects of the degree, online documentation, as well as personal projects. In any case, this training has been prior to the project, so no direct training expenses have been generated.

To carry out the development of the project it has been necessary to use a personal computer, in which the programming, the preparation of the graphics and the elaboration of this report have been carried out. The chosen computer is a 13-inch MacBook Pro with M1 chip, with a price of 1,909.00 € and a useful life calculated in 4 years (maximum straight-line depreciation rate of 25%), which will be the amortization period in the calculations [24]. This results in an annual cost of €477.25.

In addition, given the pandemic situation in which this project has been circumscribed due to COVID-19, it has been essential to use an application for communication purposes with the tutor, and thus be able to hold project follow-up meetings via videoconference. Meetings were carried out using Microsoft Teams. Both for the tutor and for me, since we belong to the university, the use of this tool is free of charge for us. Consequently, no direct cost has been incurred to the project derived from the use of this tool.

The table below shows the sum of the costs associated with tools and resources; free elements have been omitted.

	Monthly cost	Duration	Total cost
MacBook Pro 13”	39.77 €	8 months	318.17 €
		TOTAL	**318.17€**

Table 7 - Project material costs

10.1.2. Human costs

The work of a Researcher has been necessary for the development of the project. This profile is included in the category Researcher with a degree, engineer, architect, or graduate degree. The INE salary structure survey [25] was used to estimate the cost of the worker. The search has been limited to the autonomous community of Aragon, Services sector and both sexes, in the period of 2018. Therefore, the average gross annual salary is €22,495.7.

But that is the worker's gross salary. If a company or institution wanted to hire him/her, in addition to the salary, it would have to face a series of expenses related to social security that range between 30% and 35% [26].

Taking these expenses into account, the total cost will be about 30,000 € per year, which means a cost of 2,500 € / month (at 12 payments).

Considering that a month means 22 working days and, given that a full working day is contemplated, i.e., 8 hours a day, this makes a total of 176 hours a month. Therefore, by establishing a linear relationship between the values, we can deduce an approximate cost of 14 € per hour.

	Hourly cost	Time spent	Total cost
Researcher	14 € / hour	300 hours	4,200 €
		TOTAL	**4,200 €**

Table 8 - Project human costs

Since the project is limited to 12 ECTS, and each ECTS credit corresponds to 25 hours of work, the scope in hours is 300 hours.

10.1.3. Infrastructure costs

During the 8 months (Oct-2020 to May-2021) in which this project has been developed, it has been necessary a stay from which to work – in this case from my home – and, consequently, this translates into infrastructure costs, reflected below:

	Average monthly charge		Duration		Total cost	
Rent	350	€ / month	8	months	2,800	€
Water supply	15	€ / month			120	€
Electricity supply	70	€ / month			560	€
Gas supply	175	€ / month			1,400	€
Internet	50	€ / month			400	€
				TOTAL	**5,280**	**€**

Table 9 - Project infrastructure costs

10.1.4. Total costs

The total cost charged for the development of the project is detailed below:

	Total cost	
Material costs	318.17	€
Human costs	4,200	€
Infrastructure costs	5,280	€
TOTAL	**9,798.17**	**€**

Table 10 - Total project costs

The total cost amounts to 9,798.17 €.

11. Conclusion

The project has been successfully completed, as the objectives set out in the initial proposal have been met. Despite having focused the search space on the final bosses of the video game Kromaia, it is possible to extend it. All the elements that make up the video game (i.e., the models) are defined by a metamodel. In this way, it is possible to raise the level of abstraction, which provides the necessary flexibility to generalize the scope of application to other case studies, whether they are video games or not, if there is a metamodel that defines a ruleset that comply with the models that implement them.

11.1. Future work

In addition to what has been explored, there are more plans that have been left unexplored due to the limited scope of this project. For instance, extend the training of ML classifiers to every model in the game, i.e., not only to elements such as simple enemies and final level bosses, but also to other levels or objects. This would introduce much more variability into the ML models, which could result in either a greater richness due to a larger data source or the opposite, a poorer performance due to a drop in accuracy.

Another unexplored option would be, in any case, to adjust the way the mutants are calculated. To adjust it, you would still have to look at the errors made by students and have the mutants inject those errors.

Finally, in addition to the interest there would be in integrating the model viewer within the model editor to be able to see in real time a preview of the model being built, we are also convinced that it would increase productivity to have an integration of the ML classifier as a plugin with the editor, so that it would be able to validate the relationships of the model in real time, giving suggestions of how similar are the relationships between the elements that make up the model, compared to the relationships of the other models already included in the game.

12. Bibliography

[1] D. Blasco, C. Cetina and O. Pastor, "A fine-grained requirement traceability evolutionary algorithm: Kromaia, a commercial video game case study," *Inf. Softw. Technol.,* vol. 119, 2020.

[2] S. Kent, "Model Driven Engineering," in *Integrated Formal Methods*, 2002.

[3] S. Gulwani, O. Polozov and R. Singh, "Program Synthesis," *Found. Trends Program. Lang.,* vol. 4, pp. 1-119, 2017.

[4] D. Blasco, J. Font, M. Zamorano and C. Cetina, "An evolutionary approach for generating software models: The case of Kromaia in Game Software Engineering," *J. Syst. Softw.,* vol. 171, p. 110804, 2021.

[5] C. David and D. Kroening, "Program synthesis: challenges and opportunities," *Phil. Trans. R. Soc. A,* vol. 375, 2017.

[6] "Directed acyclic graph," [Online]. Available: https://es.wikipedia.org/wiki/Grafo_ac%C3%ADclico_dirigido. [Accessed June 2021].

[7] Elsevier Developers, "Scopus Search API," Documentation: API Specification, [Online]. Available: https://dev.elsevier.com/documentation/SCOPUSSearchAPI.wadl.

[8] D. Harel and R. Marelly, Come, Let's Play: Scenario-Based Programming Using LSCs and the Play-Engine, Springer, 2003.

[9] C. Arora, M. Sabetzadeh, S. Nejati and L. C. Briand, "An active learning approach for improving the accuracy of automated domain model extraction," *ACM Trans. Softw. Eng. Methodol.,* vol. 28, no. 1, p. 4, 2019.

[10] "Python," [Online]. Available: https://www.python.org/. [Accessed May 2021].

[11] "Visual Studio Code," [Online]. Available: https://code.visualstudio.com/. [Accessed May 2021].

[12] "Jupyter Notebook," [Online]. Available: https://jupyter.org/. [Accessed May 2021].

[13] "Git," [Online]. Available: https://git-scm.com/. [Accessed May 2021].

[14] "GitHub," [Online]. Available: https://github.com/. [Accessed May 2021].

[15] "SciPy," [Online]. Available: https://www.scipy.org/. [Accessed May 2021].

[16] "NumPy," [Online]. Available: https://numpy.org/. [Accessed May 2021].

[17] "Pandas," [Online]. Available: https://pandas.pydata.org/. [Accessed May 2021].

[18] "Matplotlib," [Online]. Available: https://matplotlib.org/. [Accessed May 2021].

[19] "Seaborn," [Online]. Available: https://seaborn.pydata.org/. [Accessed May 2021].

[20] "Microsoft Planner," [Online]. Available: https://www.microsoft.com/es-es/microsoft-365/business/task-management-software. [Accessed May 2021].

[21] A. Shabtai, R. Moskovitch, Y. Elovici and C. Glezer, "Detection of Malicious Code by Applying Machine Learning Classifiers on Static Features: A State-of-the-Art Survey," *Inf. Secur. Tech. Rep.,* vol. 14, no. 1, pp. 16-29, 2009.

[22] P. Refaeilzadeh, L. Tang and H. Liu, "Cross-Validation," in *Encyclopedia of Database Systems*, L. Liu and M. Tamer, Eds., Springer, 2009, pp. 532-538.

[23] Q. Song, Z. Jia, M. J. Shepperd, S. Ying and J. Liu, "A General Software Defect-Proneness Prediction Framework," *IEEE Trans. Software Eng.,* vol. 37, no. 3, pp. 356-370, 2011.

[24] "Tabla de coeficientes de amortización lineal," Agencia Tributaria, [Online]. Available: https://www.agenciatributaria.es/AEAT.internet/Inicio/_Segmentos_/Empresas_y_profesionales/Empresas/Impuesto_sobre_Sociedades/Periodos_impositivos_a_partir_de_1_1_2015/Base_imponible/Amortizacion/Tabla_de_coeficientes_de_amortizacion_lineal_.shtml. [Accessed May 2021].

[25] "Ganancia media anual por trabajador," Instituto Nacional de Estadística (INE), [Online]. Available: https://www.ine.es/jaxiT3/Datos.htm?t=28192. [Accessed May 2021].

[26] "Bases y tipos de cotización 2021," Seguridad Social, [Online]. Available: https://www.seg-social.es/wps/portal/wss/internet/Trabajadores/CotizacionRecaudacionTrabajadores/36537. [Accessed May 2021].

[27] K. Lano, "Constraint-driven development," *Information and Software Technology,* pp. 406-423, 2008.

[28] F. Kraemer, V. Slåten and P. Herrmann, "Tool support for the rapid composition, analysis and implementation of reactive services," *Journal of Systems and Software,* pp. 2068-2080, 2009.

[29] K. Lano and S. Kolahdouz-Rahimi, "Constraint-based specification of model transformations," *Journal of Systems and Software,* pp. 412-436, 2013.

[30] D. Batory, "Feature Modularity in Software Product Lines," in *10th International Software Product Line Conference*, Baltimore, 2006.

[31] D. Batory, "Program Refactoring, Program Synthesis, and Model-Driven Development," in *Lecture Notes in Computer Science*, Braga, 2007.

[32] S. Trujillo, D. Batory and O. Diaz, "Feature oriented model driven development: A case study for portlets," in *29th International Conference on Software Engineering*, Minneapolis, 2007.

[33] M. Raghothaman, J. Mendelson, D. Zhao, M. Naik and B. Scholz, "Provenance-guided synthesis of datalog programs," *Proceedings of the ACM on Programming Languages,* pp. Volume 4, Article number 62, 2020.

[34] E. Yahav, S. Fink, N. Dor, G. Ramalingam and E. Geavy, "From typestate verification to interpretable deep models (invited talk abstract)," in *28th ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2019*, Beijing, China, 2019.

[35] A. Iyer, M. Jonnalagedda, S. Parthasarathy, A. Radhakrishna and S. Rajamani, "Synthesis and machine learning for heterogeneous extraction," in *40th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2019*, Phoenix, United States, 2019.

[36] W. Lee, K. Heo, R. Alur and M. Naik, "Accelerating search-based program synthesis using learned probabilistic models," in *39th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2018*, Philadelphia, United States, 2018.

[37] V. Le and S. Gulwani, "FlashExtract: A framework for data extraction by examples," in *35th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2014*, Edinburgh, United Kingdom, 2014.

[38] R. Singh, S. Gulwani and A. Solar-Lezama, "Automated feedback generation for introductory programming assignments," in *34th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2013*, Seattle, WA, United States, 2013.

[39] A. Udupa, S. Mador-Haim, A. Raghavan, M. Martin, J. Deshmukh and R. Alur, "TRANSIT: Specifying protocols with concolic snippets," in *34th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2013*, Seattle, WA, United States, 2013.

[40] A. Haber, K. Hölldobler, C. Kolassa, M. Look, K. Müller, B. Rumpe, I. Schaefer and C. Schulze, "Systematic synthesis of delta modeling languages," *International Journal on Software Tools for Technology Transfer,* pp. 601-626, 2015.

13. Annexes

13.1. Annex 1: Project proposal

Nombre alumno:	RODRIGO CASAMAYOR MORAGRIEGA
Titulación:	GRADUADO EN INGENIERÍA INFORMÁTICA. PLAN 2008 (BOE 15/12/2008)
Curso académico:	CUARTO CURSO

1. TÍTULO DEL PROYECTO

Exploring Program Synthesis in Model Driven Engineering.

2. DESCRIPCIÓN Y JUSTIFICACIÓN DEL TEMA A TRATAR

En la actualidad, la mayoría del software se sigue desarrollando a través de métodos convencionales. Esto es, a una mayor demanda de software, mayor contratación de personas físicas que llevan a cabo el desarrollo. Esto es viable cuando el alcance del proyecto es limitado y no se requiere de un mantenimiento prolongado en el tiempo o mejoras de este. Pero, cuando se necesita continuar desarrollando una familia de productos, esto hace que se encarezca el producto final como consecuencia de cada vez más un mayor número de desarrolladores que construyan el software y lo mantengan en el tiempo, con la contraprestación de que posiblemente se introduzcan nuevos errores en el software de estos productos debido a la implementación repetitiva de las características que lo definen, en vez de reaprovechar y modificar las ya existentes. Al automatizar estos procesos, se disminuye en gran medida el error humano y se abaratan los costes de producción.

3. OBJETIVOS DEL PROYECTO

- Identificación del gap de investigación y propuesta de una aproximación.
- Diseño e implementación de la aproximación.
- Elección de baseline y caso de estudio.
- Evaluación comparando con baseline.
- Discusión de los resultados obtenidos.

4. METODOLOGÍA

La metodología se establecerá en las primeras fases del proyecto.

5. PLANIFICACIÓN DE TAREAS

Las tareas quedan predefinidas de manera global en los objetivos. Serán fijadas de forma concreta durante el desarrollo del proyecto.

6. OBSERVACIONES ADICIONALES

Esta propuesta ha sido revisada por el docente Carlos Cetina.

13.2. Annex 2: Meeting notes

MEETING: 01

Date:	29/10/2020	
Starts:	17:00	**Ends:** 17:30
Location:	Microsoft Teams meeting	
Prepares minutes:	Rodrigo Casamayor	
Invitees:	Carlos Cetina, Rodrigo Casamayor	

Checklist

No.	**Subject**
1	Create a planner with the tasks below.
2	Re-draw the ppt initial sketches (slides 1, 2, 5).
3	Write the state-of-the-art (ppt figure 1; 1-2 pages).
4	Related work classification: • Dimension in Program Synthesis. • Stack MDD (synthesizer and user intent).
5	**Next meeting on** 05/11/2020.

MEETING: 02

Date:	05/11/2020	
Starts:	17:00	**Ends:** 17:30
Location:	Microsoft Teams meeting	
Prepares minutes:	Rodrigo Casamayor	
Invitees:	Carlos Cetina, Rodrigo Casamayor	

Checklist

No.	**Subject**
1	Apply corrections to the abstract's figures.
2	Continue working on the Related Work taxonomy.
3	Explain the reasoning behind the queries (i.e., "program synthesis" vs. just "synthesis").
4	Build an ANNEX with a table with the different queries and the resulting lists of papers.
5	**Next meeting on** 19/11/2020

MEETING: 03

Date:	19/11/2020	
Starts:	17:00	**Ends:** 17:30
Location:	Microsoft Teams meeting	
Prepares minutes:	Rodrigo Casamayor	
Invitees:	Carlos Cetina, Rodrigo Casamayor	

Checklist

No.	**Subject**
1	Change "Dimensions in Program Synthesis" figure: replace left hand side X by Y.
2	Reasoning of why considering the different two queries for PS: "Program Synthesis" is more restrictive than "Synthesis".
3	Scopus queries table: • Add `Id` column (Q1, Q2, …, Qn). • Put colors for bold words PS/MDD (orange/blue). • Sort by ascending restriction.
4	Draw figure from "rationaleSQ.pptx": • Reason that we would keep the third consultation, which results in 95 documents, if we had to do it by hand. • Evaluate making a web scrapping utility to further analyze the other queries with huge document results.
5	**Next meeting on** 26/11/2020

MEETING: 04

Date:	26/11/2020	
Starts:	17:00	**Ends:** 17:30
Location:	Microsoft Teams meeting	
Prepares minutes:	Rodrigo Casamayor	
Invitees:	Carlos Cetina, Rodrigo Casamayor	

Checklist

No.	**Subject**
1	Argue restriction of 5000 results per search.
2	Assignment of the code to reproduce the searches and their respective filtering. • Code always updated, because below it calls the Elsevier REST API for Scopus, so the result documents can be updated in the future.
3	Related Work taxonomy (focus on the introductory and concluding sections of the papers): • What do they use models for? • What do they use synthesis for? • What do they accomplish? • Identify the area of expertise (i.e., testing, maintenance, refactoring, inverse engineering, …).
4	**Next meeting on** 21/01/2021

MEETING:	**05**

Date:	21/01/2021	
Starts:	17:00	**Ends:** 17:30
Location:	Microsoft Teams meeting	
Prepares minutes:	Rodrigo Casamayor	
Invitees:	Carlos Cetina, Rodrigo Casamayor	

Checklist

No.	Subject
1	Related Work taxonomy (apply corrections and continue working…).
2	**Next meeting on** 29/01/2021

MEETING: 06

Date:	29/01/2021	
Starts:	17:00	**Ends:** 17:30
Location:	Microsoft Teams meeting	
Prepares minutes:	Rodrigo Casamayor	
Invitees:	Carlos Cetina, Rodrigo Casamayor	

Checklist

No.	**Subject**
1	Request full-text paper of: Feature modularity in software product lines, Batory, D., SPLC 2006.
2	Search Google Scholar for papers citing the work of D. Batory and K. Lano.
3	End with the taxonomy of the Related Work.
4	**Next meeting on** 05/03/2021

MEETING: 07

Date:	05/03/2021	
Starts:	17:00	**Ends:** 17:30
Location:	Microsoft Teams meeting	
Prepares minutes:	Rodrigo Casamayor	
Invitees:	Carlos Cetina, Rodrigo Casamayor	

Checklist

No.	**Subject**
1	Ask Jorge Chueca for the image of the SDML metamodel.
2	Perform metamorphic testing (model mutations).
3	Try out SciPy.
4	**Next meeting on**

13.3. Annex 3: Scopus queries

ID	Query	Document Count	Discarded?
Q1	TITLE-ABS-KEY ("Program Synthesis") AND TITLE-ABS-KEY ("Model Driven Development" OR "MDD") SUBJAREA (COMP)	4	No
Q2	TITLE-ABS-KEY ("Program Synthesis") AND TITLE-ABS-KEY ("Model Driven Engineering" OR "MDE" OR "Model Driven Development" OR "MDD") SUBJAREA (COMP)	9	No
Q3	TITLE-ABS-KEY ("Program Synthesis") AND TITLE-ABS-KEY ("Model Driven Engineering" OR "MDE" OR "Model Driven Development" OR "MDD" OR "Model") SUBJAREA (COMP)	340	No
Q4	TITLE-ABS-KEY ("Synthesis") AND TITLE-ABS-KEY ("Model Driven Development" OR "MDD") SUBJAREA(COMP)	95	No

Q5	TITLE-ABS-KEY ("Synthesis") AND TITLE-ABS-KEY ("Model Driven Engineering" OR "MDE" OR "Model Driven Development" OR "MDD") SUBJAREA (COMP)	197	No
Q6	TITLE-ABS-KEY ("Synthesis") AND TITLE-ABS-KEY ("Model Driven Engineering" OR "MDE" OR "Model Driven Development" OR "MDD" OR "Model") SUBJAREA (COMP)	36947	Yes

These table below represents the queries that are performed at Scopus database. The search was conducted on October 29th, 2020.

The idea is to build queries that would allow me to go from less restrictive to more restrictive searches. It is therefore the table above is also sorted by ascending restriction.

Note that the combination of "Synthesis" with "Program Synthesis" as a query, has no added value for us. The query will result like "Synthesis" OR "Program Synthesis", but the document count will be unaffected. Considering that,

`COUNT ("Program Synthesis" OR "Synthesis")` is the same as `MAX [COUNT ("Program Synthesis"), COUNT ("Synthesis")]`, which in all queries turns out to be `COUNT ("Synthesis")`; it follows that: **`COUNT ("Program Synthesis" OR "Synthesis") = COUNT ("Synthesis").`**

13.4. Annex 4: Taxonomy of the Related Work

Constraint-driven development [27]

Area of expertise

Identify the area of expertise.

Computer-aided education.

Usage of models and synthesis

What are models and synthesis used for?

Regarding the dimensions in program synthesis:

What is used as user intent expression?	UML-RSDS
What search technique does the synthesizer use?	Constraint solving

In this seminal paper, the classical approach of program synthesis with model-driven development in UML are combined.

In this combined approach, developers specify a platform-independent model (PIM) using notations such as use cases, class diagrams and state machines, with constraints such as class invariants and operation postconditions being used as the primary (and ideally complete) description of the functionality of the system.

From these descriptions, platform-specific models (PSMs) and implementations can be derived by a systematic process, with many steps being automatable. This approach is called *constraint-driven development (CDD)*.

Result

What is achieved?

What use is made of the results?	**ACADEMIC**	**INDUSTRIAL**
	X	X
How have the results been evaluated?	A toolset has been provided for educational use in universities in Canada and the UK and evaluated for industrial use by several companies.	

The concept of **constraint-driven development (CDD)** has been introduced.

Tool support for the rapid composition, analysis, and implementation of reactive services [28]

Area of expertise

Identify the area of expertise.

End-user programming.

Usage of models and synthesis

What are models and synthesis used for?

Regarding the dimensions in program synthesis:

What is used as user intent expression?	UML 2.0
What search technique does the synthesizer use?	Constraint Solving

In this paper, an integrated set of tools *Arctis* for the rapid development of services is presented.

By following this method, services are composed of collaborative building blocks expressed as a combination of UML 2.0 collaborations, activities, and so-called external state machines (ESMs) to document their externally visible behavior.

An engineering method SPACE is developed, which compromises a speed up of the development, since the design of a service is facilitated by applying reusable building blocks that are general or domain specific collaborations which can be integrated into several system descriptions.

To guarantee that important system properties are kept, a fully automated process of model checking is carried out. Thus, the collaborative models can be fully automatically transformed into executable code.

Result

What is achieved?

What use is made of the results?	**ACADEMIC**	**INDUSTRIAL**
	X	
How have the results been evaluated?	Several platforms have been deployed using the toolset which have been developed in the domain of home	

	automation or learning platforms that make use of location-aware services.

The Eclipse workbench has been used to build the library browser and editor called *Arctis* on it.

Arctis is used within the applied research project ISIS (*Infrastructure for Integrated Services*) funded by the Research Council of Norway. In this project, methods, tools, and platforms have been developed for the rapid specification and deployment of services in the domain of home automation.

The tool is also used within the FABULA project, which deals with the creation of learning platforms that make use of location-aware services.

Constraint-based specification of model transformations [29]

Area of expertise

Identify the area of expertise.

Software engineering.

Usage of models and synthesis

What are models and synthesis used for?

Regarding the dimensions in program synthesis:

What is used as user intent expression?	UML-RSDS
What search technique does the synthesizer use?	Constraint Solving

In this paper, an approach for the automated derivation of correct-by-construction transformation (with transformations considered as single operations or use cases, defined by pre- and post-conditions) implementations from high-level specifications is described.

For this approach, a range of model transformation case studies of different kinds (re-expression, refinement, quality improvement and abstraction transformations) illustrate it and describe ways in which transformations can be composed and evolved using it.

This process has been implemented as part of the UML-RSDS ('Reactive System Design Support') toolset for MDD, a subset of UML to specify the state and behavior of a system at the platform-independent model (PIM) level, using standard UML notations as far as possible, to improve the reusability of model transformations and minimize the amount of training required to use the approach.

Result

What is achieved?

What use is made of the results?	**ACADEMIC**	**INDUSTRIAL**
	X	
How have the results been evaluated?	Usability has been measured by survey results. The efficiency of the generated code has also been evaluated against a test suite.	

The focus in this work has been on providing an extensive automation able to support rapid and agile development at a higher level of abstraction through transformations for use in a **wide range of application scenarios**.

Feature Modularity in Software Product Lines [30]

Area of expertise

Identify the area of expertise.

End-user programming.

Usage of models and synthesis

What are models and synthesis used for?

Regarding the dimensions in program synthesis:

What is used as user intent expression?	FOP model
What search technique does the synthesizer use?	N/A[2]

In this seminal paper, the foundation of *Feature Oriented Programming (FOP)* — a design methodology and tools for **program synthesis** in software product lines (SPLs) — is laid.

This programming paradigm consists of programs that are specified in terms of features.

The fundamental units of modularization in FOP are program extensions (aspects, mixins, or traits) that encapsulate the implementation of an individual feature. A FOP model of a product-line is an algebra: base programs are constants and program extensions are functions (that add a specified feature to an input program). Program designs are expressions — compositions of functions and constants — that are amenable to optimization and analysis.

Result

What is achieved?

What use is made of the results?	**ACADEMIC**	**INDUSTRIAL**
	X	X

[2] This paper is a summary of a tutorial, so no further information is available.

Explanation: In the conferences, in addition to the technical papers there are other satellite events, one type of these are the tutorials (researchers go to teach what they know how to do to other researchers through demos or classes).

How have the results been evaluated?	FOP has been used to develop product-lines in widely varying domains, including compilers for extensible Java dialects, fire support simulators for the U.S. Army, network protocols, web portlets, and program verification tools.

This paper reviews core results on FOP:

- compositional models of software development and *program synthesis*,
- models and tools for synthesizing code and non-code artifacts,
- formal representations of feature models and automatic algorithms for verifying feature compositions,
- relationships between metaprogramming, product lines, and *model driven engineering (MDE)*, and
- tool demonstrations of the above.

Program Refactoring, Program Synthesis, and Model-Driven Development [31]

Area of expertise

Identify the area of expertise.

Maintenance.

Usage of models and synthesis

What are models and synthesis used for?

This paper is about the essential complexity of software structure. There are increasingly overlapping ideas in the areas of program refactoring, program synthesis, and model-driven development, all of which deal with program structure and maintenance.

Continuing his recent work carried out a year ago [30], updates on recent advances and provides a forecast of how they will evolve in terms of complexity management by raising the level of abstraction in programming.

Result

What is achieved?

It is sketched the ideas of *architectural metaprogramming*: the idea that programming and design is a computation, where programs are values and functions (a.k.a. transformations) that map programs to programs; and then they are reflected on recent advances at that time in program refactoring, program synthesis, and model-driven development from its perspective.

Feature Oriented Model Driven Development: A Case Study for Portlets [32]

Area of expertise

Identify the area of expertise.

End-user programming.

Usage of models and synthesis

What are models and synthesis used for?

Regarding the dimensions in program synthesis:

What is used as user intent expression?	FOP model
What search technique does the synthesizer use?	Deductive

In this paper, *Feature Oriented Model Driven Development (FOMDD)* — a blend of FOP and MDD that shows how products in a software product line can be synthesized in an MDD way by composing features to create models, and then transforming these models into executables — is reviewed.

Model Driven Development (MDD) is an emerging paradigm for software construction that uses models to specify programs, and model transformations to synthesize executables. Feature Oriented Programming (FOP) is a paradigm for software product lines where programs are synthesized by composing features.

Result

What is achieved?

What use is made of the results?	**ACADEMIC**	**INDUSTRIAL**
	X	
How have the results been evaluated?	Through the mathematical properties of portlet synthesis, the correctness of the abstractions, tools, and specifications were validated, as well as optimized portlet synthesis.	

A case study of FOMDD on a product line of portlets has been presented, which are components of web portals.

Provenance-guided synthesis of datalog programs [33]

Area of expertise

Identify the area of expertise.

End-user programming.

Usage of models and synthesis

What are models and synthesis used for?

Regarding the dimensions in program synthesis:

What is used as user intent expression?	Datalog programs
What search technique does the synthesizer use?	Constraint Solving (CEGIS)

In this paper, a new approach to synthesize Datalog programs from input-output specifications has been introduced.

The approach leverages query provenance to scale the counterexample-guided inductive synthesis (CEGIS) procedure for program synthesis. In each iteration of the procedure, a SAT solver proposes a candidate Datalog program, and a Datalog solver evaluates the proposed program to determine whether it meets the desired specification. Failure to satisfy the specification results in additional constraints to the SAT solver.

Efficient algorithms are proposed to learn these constraints based on "why" or "why not" provenance information obtained from the Datalog solver.

Result

What is achieved?

What use is made of the results?	**ACADEMIC**	**INDUSTRIAL**
	X	
How have the results been evaluated?	The efficiency of the generated code has been evaluated against a suite of 40 synthesis tasks from three different domains.	

A tool called *ProSynth* implements the stated approach and present experimental results that demonstrate significant improvements over the state-of-the-art, including in synthesizing invented predicates, reducing running times, and in decreasing variance in synthesis performance.

From typestate verification to interpretable deep models (invited talk abstract) [34]

Area of expertise

Identify the area of expertise.

Maintenance.

Usage of models and synthesis

What are models and synthesis used for?

Regarding the dimensions in program synthesis:

What is used as user intent expression?	Partial programs
What search technique does the synthesizer use?	Statistical (Machine Learning)

In this paper, it is reviewed the original paper "Effective Typestate Verification in the Presence of Aliasing" (ISSTA 2006 Proceedings), which described a scalable framework for verification of typestate properties in real-world Java programs, showing the evolution of the ideas contained therein over the years.

Result

What is achieved?

What use is made of the results?	**ACADEMIC**	**INDUSTRIAL**
	X	
How have the results been evaluated?	The paper introduced several techniques that have been used widely in the static analysis of real-world programs.	

The paper shows how some of these ideas have evolved into work on machine learning for code completion and discuss recent general results in machine learning for programming.

Synthesis and machine learning for heterogeneous extraction [35]

Area of expertise

Identify the area of expertise.

Software engineering.

Usage of models and synthesis

What are models and synthesis used for?

Regarding the dimensions in program synthesis:

What is used as user intent expression?	Examples
What search technique does the synthesizer use?	Statistical (Machine Learning)

In this paper, a way to combine techniques from the program synthesis and machine learning communities to extract structured information from heterogeneous data is presented.

A new approach is introduced, called HDEF, *Heterogeneous Data Extraction Framework*, which works by first training an ML model using some training data. Afterwards, this base ML model is used to produce candidate output labels for the entire heterogeneous dataset, including on input formats for which there is no training data.

Due to the generalization, the base ML model produces output labels even for the formats with no training data. However, the labels so generated are typically noisy. Then, noisy labels are used as input specifications to a modified program synthesis algorithm.

Since the input data is heterogeneous, a single program cannot handle all the inputs. Thus, a synthesis algorithm, called NoisyDysjSyn, *Noisy Disjunctive Program Synthesis*, produces a set of programs that cover the entire input dataset and maximizes the number of inputs for which correct outputs are produced, where correctness of an output is defined using a type of specification called *field constraint*.

Result

What is achieved?

	ACADEMIC	**INDUSTRIAL**

What use is made of the results?	X	X
How have the results been evaluated?	The results have been shared with a team that builds an enterprise-scale M2H email extractor, and with whom there is a collaboration to bring these ideas to the product.	

While the HDEF algorithm can handle random noise in ML models using sampling, it does not work in cases with systematic noise, and ends up boosting it.

One possible solution to this issue is to write field constraints that eliminate the systematic noise. For example, if we provide a field constraint that always match a given regular expression, the systematic noise can be eliminated easily. However, it might not be possible to write such field constraints, at least not for all the cases.

The HDEF algorithm also has difficulty handing inputs with multiple semantic contexts.

Accelerating search-based program synthesis using learned probabilistic models [36]

Area of expertise

Identify the area of expertise.

Software engineering.

Usage of models and synthesis

What are models and synthesis used for?

Regarding the dimensions in program synthesis:

What is used as user intent expression?	Related programs
What search technique does the synthesizer use?	Statistical (Probabilistic)

In this paper, a general approach to accelerate search-based program synthesis by leveraging a probabilistic program model to guide the search towards likely programs, *syntax-guided synthesis* (SyGuS), has been presented.

The approach comprises a weighted search algorithm that is applicable to a wide range of probabilistic models. A method based on transfer learning that allows a state-of-the-art probabilistic model to avoid over-fitting, PHOG, is also proposed.

Result

What is achieved?

What use is made of the results?	**ACADEMIC**	**INDUSTRIAL**
	X	
How have the results been evaluated?	The effectiveness of the approach has been demonstrated on many synthesis problems from a variety of application domains.	

The approach has been implemented in a tool called *Euphony* and evaluated it on SyGuS benchmark problems form a variety of domains. The experimental results show that the approach outperforms existing general-purpose and domain-specific synthesis tools.

FlashExtract: A framework for data extraction by examples [37]

Area of expertise

Identify the area of expertise.

End-user programming.

Usage of models and synthesis

What are models and synthesis used for?

Regarding the dimensions in program synthesis:

What is used as user intent expression?	Examples
What search technique does the synthesizer use?	Deductive

In this paper, a general framework to extract relevant data from semi-structured documents using examples called *FlashExtract* has been presented.

It includes: (a) an interaction model that allows end-users to give examples to extract various fields and to relate them in a hierarchical organization using structure and sequence constructs; and (b), an inductive synthesis algorithm to synthesize the intended program from few examples in any underlying domain-specific language for data extraction that has been built using a specified algebra of few core operators (map, filter, merge, and pair).

Result

What is achieved?

What use is made of the results?	**ACADEMIC**	**INDUSTRIAL**
	X	
How have the results been evaluated?	The technique has been evaluated by means of a benchmark containing 75 documents.	

In terms of synthesis time, *FlashExtract* required an average of 0.82 seconds per field across all documents.

Automated feedback generation for introductory programming assignments [38]

Area of expertise

Identify the area of expertise.

Computer-aided education.

Usage of models and synthesis

What are models and synthesis used for?

Regarding the dimensions in program synthesis:

What is used as user intent expression?	Partial program, Reference implementation
What search technique does the synthesizer use?	Constraint Solving

In this paper, a new technique of automatically providing feedback for introductory programming assignments that can complement manual and test-cases based techniques.

It relies on constraint-based synthesis technology to efficiently search over this large space of programs. Specifically, the *Sketch* synthesizer is used, which uses a SAT-based algorithm to complete program sketches (programs whit holes) so that they meet a given specification.

The technique uses an error model describing the potential corrections and constraint-based synthesis to compute minimal corrections to student's incorrect solutions.

Result

What is achieved?

What use is made of the results?	**ACADEMIC**	**INDUSTRIAL**
	X	
How have the results been evaluated?	The technique has been evaluated on a large set of benchmarks and it can correct 64% of incorrect solutions in the benchmark set.	

This technique can establish a foundation for providing automated feedback to students being taught from online introductory programming courses through learning platforms such as MITx, Coursera, and Udacity.

TRANSIT: Specifying protocols with concolic snippets [39]

Area of expertise

Identify the area of expertise.

End-user programming.

Usage of models and synthesis

What are models and synthesis used for?

Regarding the dimensions in program synthesis:

What is used as user intent expression?	Examples, Partial programs
What search technique does the synthesizer use?	Constraint Solving

In this paper, an approach for specifying distributed protocols by adopting verification tools that interact with the programmer has been proposed.

The way to program distributed protocols is using *concolic snippets*. These are sample execution fragments that contain both concrete and symbolic values, that allow a programmer to specify the protocol behavior as a mix of concrete examples and symbolic partial transitions.

TRANSIT, a language and prototype implementation of the proposed specification methodology for distributed protocols, has been described.

Result

What is achieved?

What use is made of the results?	**ACADEMIC**	**INDUSTRIAL**
	X	X
How have the results been evaluated?	To evaluate the proposed approach for specifying protocols with *concolic snippets*, the experiences of three different programmers have been documented.	

The preliminary case studies using this tool allowed inexperienced programmers to correctly synthesize representative cache coherence protocols of modest complexity with several hours of human effort.

The translation of an incomplete flow-based description of an industrial-strength protocol into a working implementation by effectively exploiting the flexibility afforded by concolic specifications has also been carried out.

Systematic synthesis of delta modeling languages [40]

Area of expertise

Identify the area of expertise.

End-user programming.

Usage of models and synthesis

What are models and synthesis used for?

Regarding the dimensions in program synthesis:

What is used as user intent expression?	Natural language
What search technique does the synthesizer use?	Constraint Solving

In this seminal paper, a method to synthesize delta languages from a textual base language definition has been presented.

Delta modeling is a modular, yet flexible approach to represent variability by explicitly capturing system changes. To use delta modeling for a certain modeling language that has no corresponding delta modeling language yet, a separate delta language must be synthesized. Hence, to use delta modeling techniques within every step of a development process, one needs to create many delta languages manually.

The usage of this new method decreases the effort of creating a delta language and allows to synthesize an initial delta language that then can be adapted and extended.

Result

What is achieved?

What use is made of the results?	**ACADEMIC**	**INDUSTRIAL**
	X	X
How have the results been evaluated?	A comparative case study has been used to evaluate the method, which compares existing originally handwritten delta languages to automatically generated ones and to the extended delta languages using well-defined metrics.	

	The evaluation has that the languages derived by the fully automatic approach are as semantically expressive as the handwritten ones.

A process that allows synthesizing a delta language from the grammar of a given base language, relying on an automatically generated language extension that can be manually adapted to meet domain-specific needs.