Lessons Learned in Model-Based Reverse Engineering of Large Legacy Systems

Laura García-Borgoñón[1(✉)], Miguel Angel Barcelona[1], Armando J. Egea[2], German Reyes[2], Alejandro Sainz-de-la-maza[2], and Adolfo González-Uzabal[2]

[1] Instituto Tecnológico de Aragón, Av/ María de Luna 7, 50018 Zaragoza, Spain
{laurag,mabarcelona}@itainnova.es

[2] NTT Data Spain Soluciones Tecnológicas SLU, Pl. de Antonio Beltrán Martínez 1, 50002 Zaragoza, Spain
{armando.javier.egea.moneva,german.reyes.munoz,
alejandro.sainz.de.la.maza.sainz.de.la.maza,
adolfo.gonzalez.uzabal}@nttdata.com

Abstract. Large technologies companies that offer software modernization and maintenance services for legacy software applications in diverse sectors such as banking, insurance, healthcare and public sector, face a significant challenge. Legacy systems were usually developed in old programming languages, often have outdated documentation and the processes used for software development were immature. Modernization and maintenance projects include tasks such as source code analysis with high effort and time costs, and an important risk of misunderstanding. In the literature, model-driven reverse engineering (MDRE) approaches promise to address these challenges successfully, but most of existing proposals are focused on a concrete technological stack. This paper aims to present the preliminary results and lessons learned when adopting MDRE in a large multinational company, providing a series of reflections and open issues to reduce the gap between academia and industry. It introduces STRATO, a corporate solution that proposes a MDRE approach focused on a high flexibility to incorporate new programming languages. It reads source code and through model-to-model transformations convert it into platform independent conceptual, persistence and business logic models. Preliminary outcomes, lessons learned and open issues concerning MDRE industry adoption are presented.

Keywords: Model Based Reverse Engineering · Large Legacy Systems · Industrial application

1 Introduction

NTT Data is a large multinational technology company that offers to its clients comprehensive business solutions covering all aspects of the value chain, from business strategy to systems implementation. It operates in the Telecommunications, Banking, Healthcare, Industry, Insurance, Media, Public Sector and Utilities sectors. Within its activities, the maintenance of legacy information systems

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2023
M. Indulska et al. (Eds.): CAiSE 2023, LNCS 13901, pp. 330–344, 2023.
https://doi.org/10.1007/978-3-031-34560-9_20

is an important line of business due to enterprises from all sectors have many applications which were developed in the past and are still operated today. Modernization and maintenance tasks of such legacy systems are complex because the applications were usually developed in old programming languages, documentation is often outdated and poor, and software processes used to develop them are not easily repeatable. That is the reason many organizations outsource modernization and maintenance services.

Model-driven engineering (MDE) is a development paradigm that uses high-level models and model transformations in order to produce software applications [1]. The Object Management Group (OMG) [2] establishes that "a model of a system is a description or specification of that system an its environment for certain purpose". MDE is the general term for all model-based principles and techniques that can be applied to both forward and reverse engineering [3]. In [4] authors define reverse engineering as the process of comprehending software and producing a model of it at a higher abstraction level than source code. The application of MDE principles and techniques in reverse engineering is called Model-Driven Reverse Engineering (MDRE) [4].

This paper aims at sharing our experience by creating and using a MDRE approach to systematize the modernization and maintenance services. After analyzing existing solutions and considering the needs of a multinational company that must cover a wide technological spectrum, we have developed a corporate solution, called STRATO, which focuses on facilitating extensibility when incorporating new programming languages. As a proof of concept we have evaluated it on real customer legacy systems. The main contribution of this article is to present the preliminary results and lessons learned when adopting MDRE in a large multinational company, providing a series of reflections and open issues to reduce the gap between academia and industry.

The paper is organized as follows. Section 2 reports the motivation of facing to the challenge of the MDRE in NTT Data real environments. Section 3 shows the related work and background in this research. Section 4 describes the conceptual approach proposed. STRATO platform as a tool is shown in Sect. 5. Section 6 exposes results with real legacy systems as a proof of concept evaluation. Section 7 summarizes lessons learned. Finally, Sect. 8 presents conclusions and future work.

2 Motivation

One of the business areas of the multinational NTT Data is the maintenance of large legacy applications, accounting for up to 40% of the business volume. Although the service offering covers all sectors, banking and utilities are the main ones in which there is a group of clients that maintain the core of their business with software developed decades ago. Sometimes these systems have gone through evolutions, carried out by various companies over the years, so estimating the effort required to understand and know how to evolve or correct these systems is a very complex task without having time to evaluate its source code.

Generally, these systems were developed with old technological stacks and the associated documentation is either non-existent or is not a true reflection of how the software has evolved, so the first task of the maintenance team is to be able to understand how a system was built by reading the source code. This work, although it is carried out by technical experts in the specific technologies, requires a great effort and entails a time period of several weeks and, in turn, is subject to potential errors caused by the manual work of reading, interpreting and documenting the software.

From the Innovation and Strategic Investments (ISI) area, in charge of leading corporate assets that allow the company to innovate and change the business, we have set ourselves the challenge of automating this first phase of discovering large legacy systems, based on MDRE principles, working on abstract models instead on source code. For this reason, we consider the creation of a technological platform that automates parsing source code and transforming it into models, to improve the productivity of technical teams during early stages of large legacy systems maintenance projects.

Although there may be some specific solutions that perform automatic code conversion from one language to another, this scenario is in practice very little demanded by clients, since in the modernization process there are always changes in functionality or change requests not to drag errors accumulated in the evolution of code for years. Additionally, our approach must meet two characteristics: i) be easily extensible to several languages, due to the great variability of clients and sectors that means that even for the same programming language we find more than 10 versions, and; ii) be a web-based collaborative platform in which large teams can work simultaneously on the same project.

Consequently, our hypothesis is that through an MDRE solution, we will be able to improve early stages of maintenance or modernization projects of large real legacy systems in a practical way, automating acquisition, generating technical documentation and guiding the team to identify problems and propose solutions. In this way we could reduce the effort and time required to maintain an existing system, increasing our competitiveness and being proactive in reducing its technical debt, which sums up our motivation from a business perspective.

3 Background and Related Work

In the field of MDRE there are some initiatives aimed at obtaining descriptive models from existing systems that have previously been developed [5], generally structured in two steps: i) obtaining a view, a model, of the legacy system analyzed from source artifacts, and; ii) the exploitation of the model to achieve a specific objective, for example, to document or re-engineer a system.

Regarding languages for reverse engineering, OMG Architecture-Driven Modernization (ADM) [6] initiative defined different models with the aim of supporting reverse engineering activities: i) the knowledge discovery metamodel (KDM) [7], which aims to define a shared and complete representation experience capable of guaranteeing the interoperability of different tools, efficiently

supporting maintenance, evolution, evaluation and activities of modernization; ii) the Abstract Syntax Tree (AST) Metamodel (ASTM) [8], which represents the AST of virtually any programming language, allowing parsing tools to target the metamodel rather than the specific language AST. The metamodel defines the generic AST, with definitions that can be applied recurrently to most programming languages, but also allows extensions, known as specialized ASTM, to be able to include the specific characteristics of a specific programming language; iii) the Consortium for Information and Software Quality contributed to the definition of metamodels integrated with ADM standards, adding the representation of different quality measures, for example, maintainability with the Automated Source Code Maintainability Measure (ASCMM) [9]; iv) the metamodel for the definition and description of patterns as used in the architecture, design and implementation of software systems, working with software flaws or security problems, and any situation in which a pattern is appropriately applied, known as Structured Patterns Metamodel Standard (SPMS) [10], and; iv) the metamodel to support the representation of software metrics applied to existing models (Structured Metrics Metamodel, SMM) [11], which defines the way to represent measurement information related to any structured information model, being extensible to exchange both measurements as measurement information about artifacts contained or expressed by structured models. In 2017 a systematic literature review (SLR) [3] of MDRE presented 15 different initiatives, supporting several programming languages (*java*, *cobol*, *structured query language (SQL)*, *javascript*, *php* or *delphi* among others), as well as metamodels and tools created from scratch or extending existing frameworks as MODISCO [12].

Regarding case studies, in 2007 Tonella et al. [13] performed a review of empirical studies in reverse engineering and identified the need to adopt a common framework for the execution of experiments. In 2019 Pa Pascal developed a study on the practical adoption of MDRE approaches [14] with these conclusions: i) MDRE for general purpose languages such as *java* was currently still a myth, and; ii) even when there were generic tools such as MODISCO, they were far from being able to raise understanding to an abstract level such as architecture. A SLR of automatic software refactoring was conducted in 2019 by Baqais and Alshayeb [15] including 41 primary studies with these conclusions: i) most research was done at code level (78%) so they encourage to do more research at model level; ii) there was a lack of tool support (only 29% of studies included a tool chain).

Regarding existing tool chains, there are several with a MDRE approach that proposes their own metamodels: i) Spoon [16] is an open source library for parsing, rewriting, and transforming *java* source code; ii) COALA with its CoAST (Universal and language-independent abstract syntax tree) metamodel [17] is a universal AST that makes it easy to analyze every programming language, and; iii) Grammar to Model Language (Gra2Mol) [18] is a domain-specific transformation language for defining relationships between grammar elements and metamodel elements. To build an MDRE solution from the definition of domain-specific languages, whether textual or graphical, we have also

reviewed the existing tools. Iung et al. conducted a systematic mapping study in 2020 [19] where 59 tools were exposed. To support web based interaction we have developed some proof of concepts using Web Generic Modeling Environment (WebGME) tool [20], AToMPM [21], Pyro [22], EMF in cloud [23] with different graphical modeling extensions like Sirius [24] or Graphical Language Server Platform (GLSP) [25], and finally using Theia [26] or reproducing existing solutions for data modeling [27]

After analyzing the state of the art we have evidenced that: i) there is no solution to automate reverse engineering for any programming language; ii) there are several metamodel proposals to follow an MDRE approach; iii) ASTM seems to be the OMG standard that would allow to cover the platform independent perspective we are pursuing; iv) there is not much practical evidence of the application of reverse engineering in industry in real cases to compare with.

4 Conceptual Approach

This section introduces conceptual MDRE approach proposed. Figure 1 shows a schematic of the general reverse engineering process, according to a bottom-up approach, in which top services are based on platform independent models, reducing the effort required to incorporate new languages. The new solution must be applied to various programming languages and technologies, but we have started with *java*, *cobol* and relational databases due to the volume of existing maintenance legacy systems in our business.

At the bottom there is the **legacy systems layer**. The information available from the legacy systems is used, whether they are *java* or *cobol* solutions, which can be accessed via a local folder, a compressed file with the sources, git repository credentials, or compiled code. Additionally, if the system stores persistent data, the Open Database Connectivity (ODBC) connection or the SQL of the relational database schema may be provided.

Then there is the **discovery layer**. In this phase, the source code is parsed using lexical and syntactic analyzers, according to concrete programming languages grammars, and their respective ASTs are obtained. Next, a model-to-model (M2M) conversion is carried out towards a platform independent AST model (PIM ASTM) that allows the rest of the functionalities and transformations to be reused independently of the Platform Specific Models (PSM). This way, the possibility of extending the MDRE coverage towards new languages is simplified because the rest of the upper layers would not require modifications.

The objective of the **understanding layer** is to transform the information from the ASTM to a set of more representative models of an information system. The solution includes three PIM metamodels: i) the domain model includes conceptual elements with attributes and relationships; ii) the persistence model includes the relational database schema with tables, columns and relationships; iii) the business logic model includes functions, services, their statements and relations to domain and persistence models.

At the top there is the **diagnostic and transformation services layer**. Once all the information of a legacy system is defined in the domain, persistence and business logic metamodels, there are services for technological debt diagnosis and automatic M2M transformation to reduce it.

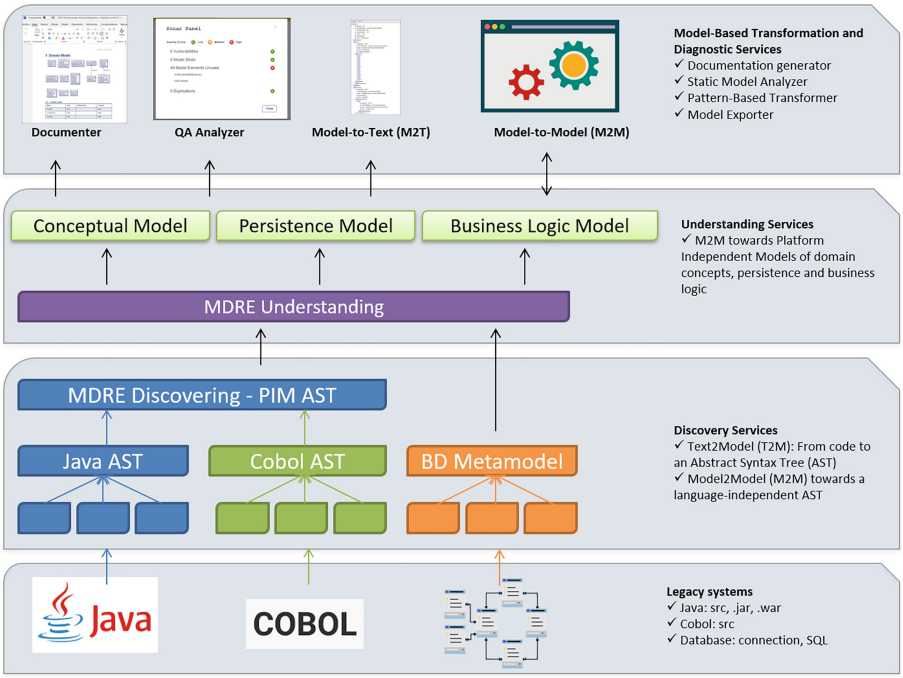

Fig. 1. Conceptual architecture for reverse engineering process

5 STRATO Platform

This section exposes the new corporate platform that, following MDRE principles, automates the discovery and understanding of legacy systems. At technical level, it is created as an extension on top of WebGME. The solution is composed by new metamodels and services for discovery, understanding, diagnosis, documentation and technical debt reduction. WebGME tool has been extended in functionality, some plugins have been developed and a new system architecture based on micro-services has been created as it is shown in the Fig. 2.

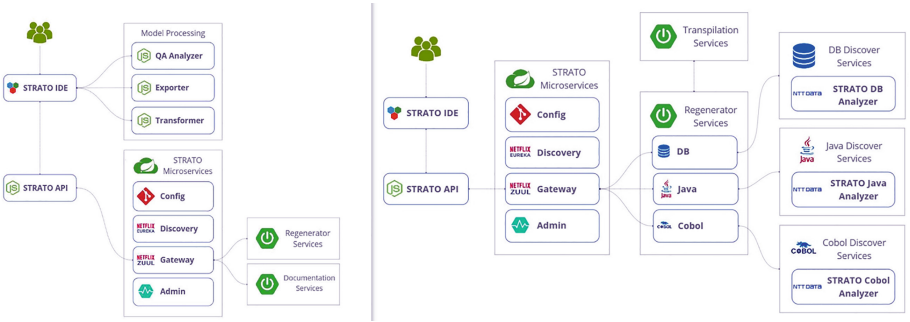

Fig. 2. Technical architecture of services

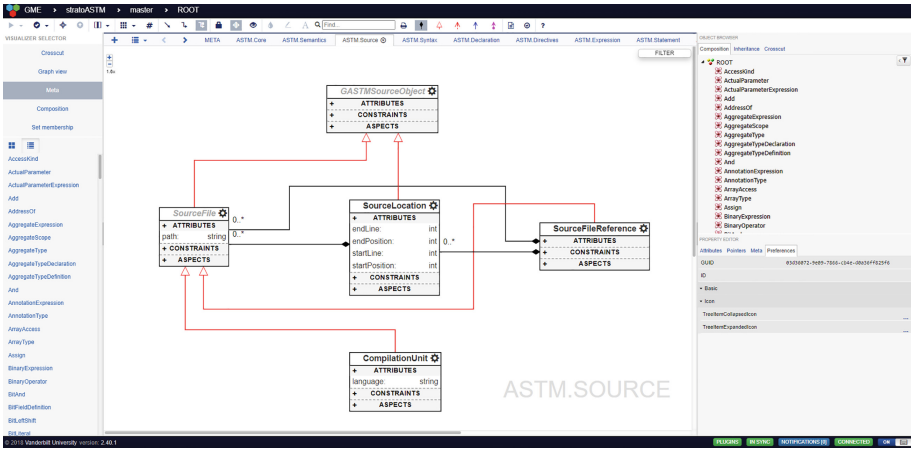

Fig. 3. View of ASTM meta visualizer in WebGME

The conceptual approach requires the generation of four platform independent metamodels: i) a domain conceptual metamodel for complex data structures, composed by 35 elements; ii) a persistence metamodel for relational databases that includes 39 metaelements; iii) a business logic metamodel for actions, services, statements and expressions, formed by 81 elements and; iv) an extension of the OMG ASTM standard composed by 196 metaelements. As the size of the metamodels is large and with WebGME we experienced problems when applying changes, we decided to create them using Essential Meta Object Facility (EMOF) [28]. We developed a tool to automate their transformation from ecore to WebGME. Figure 3 shows a view of ASTM source package metaelements into meta visualizer of WebGME.

Discovery services are responsible for parsing source code and convert it to ASTM model. It is the architecture layer closest to the concrete technology and the one that requires specific adaptation for each possible programming

language. These services are built from the AST obtained after parsing the specific grammar of each language, using the *antlr4* tool [29], as Spring Boot based *java* services. When we want to include a new programming language, we need to have its grammar and build the service that reads its AST and transforms it into the ASTM model.

Once the ASTM model is filled, all transformations and analysis are done on top of abstract models, so that adding new languages requires no changes. Understanding services are in charge of transform ASTM information into domain, persistence and business logic models. At the beginning they were developed using WebGME plugins, but due to the huge size of ASTM models, we experienced performance and memory issues, so that they were developed in *java* and interacts with models using a new Representational State Transfer (REST) Application Program Interface (API) created on top of existing core API [30]. Since ASTM is an intermediate step between the code and the abstract models, we implemented its integration into the platform for demonstration purposes, but in its current version the transformation is performed without the need to dump the extensive information into MongoDB which is used by WebGME.

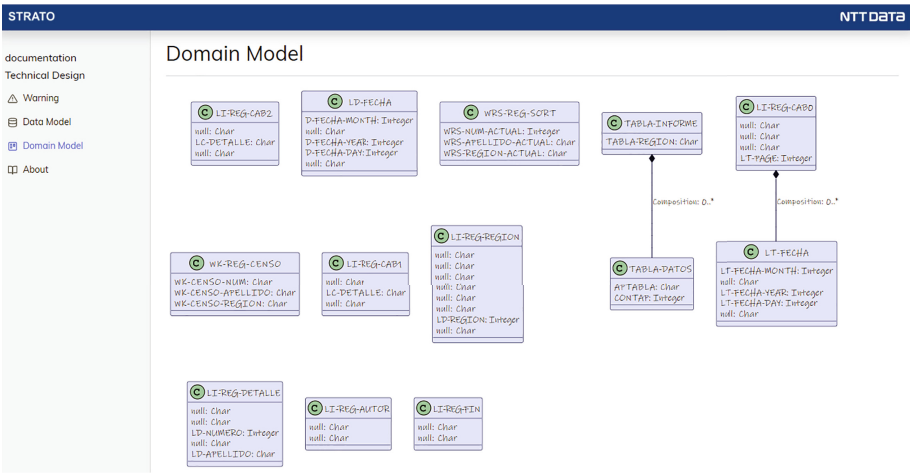

Fig. 4. Auto-generation of technical documentation from persistence model

The rest of features are implemented on the diagnostic and transformation services layer, launched by user as *javascript* plugins but all logic is included in external micro-services, so that we reduce the coupling with the WebGME tool and these services may be reused from different client tools. The first functionality is focused on the automated generation of technical documentation that allows visualizing the internal structure of a system and its call map, as an agile way of navigating between the components visually, but being able to access the developed code at the same time. This documentation may be generated in

several formats like word, powerpoint, pdf or html, and there are templates for analysis and design. Figure 4 shows an example of documentation design profile which is created from the domain model in html.

The second feature is a system quality analyzer service that identifies technological debt based on unused variables, repeated code blocks, lack of control in the possible passage of null parameters or exception handling blocks without content, among others. This detail of the quality is extended through a static analyzer of a system, always based on the models and not from the source code, which allows to graphically analyze how maintainable a system is and the main risks detected in the face of evolution or modernization, creating the equivalent to a model-based Sonarqube [31] dashboard panel. Figure 5 shows an analysis model with hyperlinks to the erroneous elements as well as a dashboard panel to quickly get an idea of its technological debt.

The third functionality is a technical debt reduction service that realizes a set of changes on business logic model according to various transformation patterns that can help to maintain the system more easily, among others: i) deletion of unused variables, function blocks, domain or persistence elements; ii) control of datatypes in parameter binding; iii) normalization of naming; iv) initialization of variables, and; v) conversion from complex data structures to equivalent domain model elements. Finally there is a service to export all models to *json* so that this models may be used by external code generators.

Fig. 5. Model based sonar and QA diagnosis from models

6 Evaluation

For the development of the platform we have followed an iterative and incremental methodology. We have started from a simplified set of legacy systems from real clients to validate each of the stages of the MDRE cycle during its development. According to the different potential purposes of a reverse engineering

empirical study [13], our main motivation was to evaluate the feasibility of the technical proposal with real large legacy systems, so that our evaluation method is a proof of concept. To validate the proposal, a set of real maintenance projects of large legacy systems have been used. In some of them we have not achieved 100% coverage of the source code because discovery services have limitations such as: i) Spring framework annotations are ignored ; ii) third party libraries are parsed only when included in maven pom.xml and accessible by maven central repository, or; iii) cobol code cannot include sql sentences. This way we only expose a summary of relevant pilot projects in which reverse engineering coverage was 100% without manual work and are shown in Table 1:

Table 1. Most relevant real projects used for validation.

Project	Legacy	Source	Size
P1	Banking transactional	Cobol	636 lines
P2	Banking batch	Cobol Micro Focus	603 lines + 17 functions
P3	Integration platform	Java 6	400 classes
P4	Online ecommerce	Java 6	110 classes
P5	Management system	Dump script SQL	1379 tables

Table 2 shows a summary of the reverse results obtained. The process time is detailed, which includes the reading and parsing of the grammar, its conversion to ASTM and its subsequent conversion to the domain, business logic and persistence models. Additionally, figures of the size of these models, expressed in metaelements, including all the nodes and the connections between them, are shown.

Table 2. Process time and model size generated from source code.

Project	Reverse process time	Domain Size	Business Logic Size	Persistence Size
P1	0:00:59	864	71283	0
P2	0:01:06	1062	92693	0
P3	0:04:39	444080	5124307	0
P4	0:03:09	1400	1319600	0
P5	0:02:08	0	0	1612781

Once the transition to technology independent models has been made, in all of them we have been able to: i) perform automatic generation of documentation in less than one minute in any format; ii) perform the analysis of the technological debt, both detailed and sonar panel, in less than two minutes; iii) generate a new

model that reduces the technological debt in less than three minutes. Due to confidentiality agreements with clients, we do not include details on the existing technical debt in the projects, but we do include details on the value that the solution provides to our maintenance teams. To this end, we have performed an analysis of the reduction in time and effort required to understand and document a legacy system, comparing it with the historical data available at the corporate level, as it is shown in Table 3:

Table 3. Summary of the effort and time reduction obtained.

KPI	Method	Average Value
Reduction of effort in analyzing a legacy system	Person hours required with STRATO vs Person hours required in historical data for similar size and complexity projects	75%
Reduction of time in analyzing a legacy system	Working period (days) required with STRATO vs Working period (days) required in historical data for similar size and complexity projects	87%
Reduction of effort for generating technical documentation	Person hours required with STRATO vs Person hours required in historical data for similar size and complexity projects	99%

7 Lessons Learned

In this section we are going to share some of the lessons learned when running this applied research project in a corporate environment with large real legacy systems. These thoughts are divided into four perspectives: industrial, technological, academic and finally we write down some reflections and open questions for the MDE community.

From a business perspective: 1) pilot projects have validated there is immediate business value when maintaining the same code base and a potential differential factor in the market: i) an average saving of 75% of the effort needed to understand how a legacy system is built and technological debt analysis, compared to historical maintenance team data; ii) elimination of interpretation errors when creating the legacy system documentation, which reduces the risk of misinterpretation when creating the work plan; iii) more than 40% of clients interviewed perceive added value in maintaining their legacy systems using these kind of automation tools, especially in banking sector. 2) the proposal is valid for any technology but depending on the project it may not be profitable. It is necessary to evaluate the cost of developing the discovery service of a technology stack when not only the grammar changes but also when there are proprietary libraries that condition the technical debt analysis. For example, when there is a

proprietary library that manages a transactional and should only be used from certain components. This requires new validation rules beyond the interaction between logical blocks and requires changes on current analyzers; 3) customers interviews have confirmed that business case for reverse engineering is not in automating technology modernization. They want to know how a legacy system was built so they can best incorporate it when building a new solution, not simply transform a system from one programming language to another, and; 4) in some projects a static analysis of the code is not enough, it is necessary to combine it with usage based logs to understand and improve its dynamic behavior. The lack of this functionality has been a barrier to its use in some pilots.

From a technological perspective: 5) we believe that it is feasible to perform a layered MDRE process using PIM ASTM, although it is necessary to perform a set of interpretations of how to map certain elements of each specific language, for example a Cobol *DISPLAY* statement, to the ASTM metaelements. We have not found a standard guide on how to map PSM AST to PIM ASTM; 6) although there are web-based MBSE tools like WebGME, their mechanisms for M2M transformations are not valid for very large systems. We have experienced memory or timeout issues when processing models with hundreds of thousands of nodes, so we have modified the architecture adding external services outside of the tool to ensure its scalability, and; 7) the effort to adapt the solution to new grammars is assumable, but to be able to analyze the technical debt at the architecture level, it is necessary to extend the business logic metamodel to be able to characterize the semantics of certain blocks, their relationships and constraints, as well as to be able to analyze runtime data or functional programming languages. We have not found any standard or existing technical approach for these needs.

At an academic level, understood by the ability to understand and reuse research results: 8) although we have extended the OMG ASTM standard, its formal specification in XSD format contains form errors and is not consistent with the official document specification, which has led us to question whether we are the first company to approach a real project with all the metaelements of the standard without reducing its scope; 9) one of the conclusions of the review done by Tonella et al. [13] was the need to adopt a common framework for the execution of experiments in reverse engineering. Many years later, we have not found a way to analyze and measure the value of reverse engineering or to compare with other academic or industry initiatives; 10) we have detected a gap between academia and industry, in the sense that many of the publications or open research projects are far from the possibility of being used in a practical way for real cases, due to the fact that the scope of the investigations is generally limited. As an example, we have not been able to parse complex *java* expressions with most of the existing proposals reviewed in the state of the art, and; 11) the existence of research articles that summarize and systematically analyze the publications or tools has allowed us to quickly obtain a global vision of the state of the art, so from the industry we highly value this type of contributions that allow us to identify research status and pending challenges. We have not been

able to find the same level of reviews at a practical level, who has tested what, how they have done it and what they have obtained, in order to have a summary at the level of research projects applied to the industry.

Finally, we write down a series of open questions for the community: 10) in our platform the code is transformed to models under three perspectives, the business logic, the database and the domain model, but the interactions among them are shown by the interconnections between metaelements. The component or block diagrams that can represent an architecture are not applicable to these monolithic systems either, so this question arises: what is the best graphic representation to express how a legacy system is built?; 11) at academic level there are many publications on the subject but they generally do not use real scenarios and the scope of the proposals is generally limited to hypothetical situations that are far from the problems that occur in the industry, so we ask ourselves: why validation cases in scientific publications are far from real scenarios? How can the industry contribute to raising the problems and needs to align the research results to the industry?; 12) at the standards level, OMG ASTM seems to be the most ambitious proposal to propose a systematic way for reverse engineering, but in our opinion it has remained at a theoretical level. In other areas, such as the specification of REST services, there are standards such as openAPI [32] that are complemented by a set of tools [33] that facilitate their practical adoption. Would it make sense that standards were complemented by the industry and the academy with tools that facilitate their use at a practical level?; 13) there are three areas in which we have not found any existing metamodel standard: i) is there any initiative to describe visual elements of interaction with the user independently of the technology?; ii) just as there are class diagrams for domain or relational diagrams for persistence, is there any standard to describe business logic independently of technology?, and; iii) is there any way to describe architectural constraints for a technological debt analysis?.

8 Conclusion and Future Work

In this article we have presented the experience of a multinational when addressing the technical approach to resolve the need to automate the process of understanding and maintaining large legacy systems. After analyzing the research results published by academia and the existing tools, we have concluded that there was no MDRE solution that can cover a variety of technology stacks and was designed for extensibility.

For this reason we have developed our own solution, created on top of WebGME but with an external micro-services architecture for transformations, which incorporates platform independent domain, persistence and business logic metamodels. Additionally, our proposal differs from other existing tools because the usage of an extension of the OMG ASTM standard, which allow us to separate the discovery phase from the rest of the layers to facilitate the future extension to new languages and technology stacks.

The use of this proposal in real projects has allowed us to confirm that: i) we can reduce the effort and time needed to understand a legacy system, by

automating the generation of documentation; ii) we are able to improve system maintenance by automatically detecting optimizations and improvements; iii) it allows us to approach technological transformation processes in a systematized and model-based manner, with a higher level of abstraction than code, and; iv) at the business level, clients have shown their interest in being able to undertake more ambitious modernization projects by being able to shorten deadlines and costs.

As lines of future work we have identified: i) the extension of the coverage of languages supported in the discovery layer, including new grammars and their transformation to the ASTM; ii) the inclusion of automatic code generation services to regenerate the code in various technologies based on the modified models, closing a technological modernization service; iii) the integration of code related to interaction with the user or semantics for architectural debt analysis; iv) the application of artificial intelligence techniques to automatically obtain a textual description of what a functional block does from its source code or its ASTM representation, and; v) the potential usage of software experimentation methods [34] to achieve a systematic evaluation of its value in real projects.

Acknowledgements. This work was supported in part by Centro para el Desarrollo Tecnológico Industrial (CDTI) under Grant IDI-20210948 (STRATO, nuevaS herramienTas para la modeRnizAción de sisTemas heredadOs).

References

1. Ruiz, F.: An approach for model-driven data reengineering (Doctoral dissertation, PhD dissertation, University of Murcia) (2016)
2. Object Management Group, Inc. Object Management Group (2012). http://www.omg.org
3. Raibulet, C., Fontana, F.A., Zanoni, M.: Model-driven reverse engineering approaches: a systematic literature review. IEEE Access **5**, 14516–14542 (2017)
4. Rugaber, S., Stirewalt, K.: Model-driven reverse engineering. IEEE Software **21**(4), 45–53 (2004)
5. Favre, J.M.: Foundations of model (Driven)(Reverse) engineering: models-Episode I: stories of the fidus papyrus and of the solarus. In Dagstuhl Seminar Proceedings. Schloss Dagstuhl-Leibniz-Zentrum für Informatik (2005)
6. Object Management Group, Inc. Arquitecture Driven Modernization Task Force (2022). https://www.omg.org/adm/
7. Pérez-Castillo, R., de Guzmán, I.G.-R., Piattini, M.: Knowledge discovery metamodel-ISO/IEC 19506: a standard to modernize legacy systems. Comput. Stand. Interf. **33**(6), 519–532 (2011)
8. Object Management Group, Architecture-Driven Modernization: Abstract Syntax Tree Metamodel (ASTM), OMG document number: formal/2011-01-05 (2011)
9. Object Management Group, Automated Source Code Maintainability Measure TM (ASCMM TM), OMG document number: formal/2016-01-01 (2016)
10. Object Management Group, Structured Patterns Metamodel Standard (SPMS), OMG document number: formal/2011-01-05 (2017)
11. Object Management Group, Structured Metrics Metamodel (SMM), OMG document number: formal/2018-03-01 (2018)

12. Bruneliere, H., Cabot, J., Dupé, G., Madiot, F.: Modisco: a model driven reverse engineering framework. Inf. Softw. Technol. **56**(8), 1012–1032 (2014)
13. Tonella, P., Torchiano, M., Du Bois, B., Systä, T.: Empirical studies in reverse engineering: state of the art and future trends. Empirical Softw. Eng. **12**, 551–571 (2007)
14. Pascal, A.: Case studies in model-driven reverse engineering. In: Proceedings of the 7th International Conference on Model-Driven Engineering and Software Development, pp. 256–263. SCITEPRESS-Science and Technology Publications, Lda (2019)
15. Baqais, A.A.B., Alshayeb, M.: Automatic software refactoring: a systematic literature review. Softw. Q. J. **28**(2), 459–502 (2020)
16. Pawlak, R., Monperrus, M., Petitprez, N., Noguera, C., Seinturier, L.: Spoon: a library for implementing analyses and transformations of java source code. Softw. Pract. Experience **46**(9), 1155–1179 (2016)
17. coAST coala Abstract Syntax Tree https://github.com/coala/coAST
18. Izquierdo, J.L.C., Cuadrado, J.S., Molina, J.G.: Gra2MoL: a domain specific transformation language for bridging grammarware to modelware in software modernization. In: Workshop on Model-Driven Software Evolution, pp. 1–8 (2008)
19. Iung, A., et al.: Systematic mapping study on domain-specific language development tools. Empirical Softw. Eng. **25**(5), 4205–4249 (2020). https://doi.org/10.1007/s10664-020-09872-1
20. Maróti, M., et al.: Next generation (meta) modeling: web-and cloud-based collaborative tool infrastructure. MPM@ MoDELS **1237**, 41–60 (2014)
21. AToMPM: a tool for multi-paradigm modeling. (n.d.). https://atompm.github.io/
22. Pyro: a collaborative, meta-model-driven, Web-based and graphical modeling environment. (n.d.). https://pyro.scce.info/
23. Eclipse foundation. (n.d.). Eclipse modeling framework in cloud. https://www.eclipse.org/emfcloud/
24. Eclipse foundation. (n.d.). Sirius web. https://www.eclipse.org/sirius/sirius-web.html
25. Eclipse foundation. (n.d.). Graphical language server platform for building web-based diagram editors. https://github.com/eclipse-glsp/glsp
26. Theia - cloud and desktop IDE platform. (n.d.). https://theia-ide.org/
27. Glaser, P.L.: Developing sprotty-based modeling tools for VS code (2022)
28. Object management group, Meta object facility (MOF) Core specification, version 2.5.1. OMG document number: formal/2019-10-01 (2016)
29. Parr, T.: The definitive ANTLR 4 reference. In: The Definitive ANTLR 4 Reference, pp. 1–326 (2013)
30. ISIS/Vanderbilt university, WebGME Documentation, Release 1.0.0 (2022)
31. Campbell, G.A., Papapetrou, P.: SonarQube in action. Manning Publications Co, Shelter Island (2013)
32. OpenAPI initiative, OpenAPI specification v3.1.0 (2021). https://spec.openapis.org/oas/v3.0.1
33. OpenAPI initiative, OpenAPI tools (2022). https://openapi.tools/
34. Juristo, N., Moreno, A.M.: Basics of Software Engineering Experimentation. Springer Science & Business Media (2013)