

Analysing Microsoft Access Projects: Building a model in a Partially Observable Domain

Santiago Bragagnolo, Nicolas Anquetil, Stéphane Ducasse, Abderrahmane Seriai, Mustapha Derras

▶ **To cite this version:**

Santiago Bragagnolo, Nicolas Anquetil, Stéphane Ducasse, Abderrahmane Seriai, Mustapha Derras. Analysing Microsoft Access Projects: Building a model in a Partially Observable Domain. ICSR 2020, Dec 2020, Hammamet, Tunisia. hal-02966146

HAL Id: hal-02966146
https://inria.hal.science/hal-02966146

Submitted on 15 Oct 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Analysing Microsoft Access Projects: Building a model in a Partially Observable Domain

Santiago Bragagnolo[1,2][0000−0002−5863−2698], Nicolas Anquetil[1][0000−0003−1486−8399], Stephane Ducasse[1][0000−0001−6070−6599], Seriai Abderrahmane[2], and Mustapha Derras[2]

[1] Université de Lille, CNRS, Inria, Centrale Lille, UMR 9189 – CRIStAL France
{name.lastname}@inria.fr
http://ww.inria.fr
[2] Berger-Levrault, France
{name.lastname}@berger-levrault.com

Abstract. Due to the technology evolution, every IT Company migrates their software systems at least once. Reengineering tools build system models which are used for running software analysis. These models are traditionally built from source code analysis and information accessible by data extractors (that we call such information *observable*). In this article we present the case of Microsoft Access projects and how this kind of project is partially observable due to proprietary storing formats. We propose a novel approach for building models that allows us to overcome this problem by reverse engineering the development environment runtime through the usage of Microsoft COM interface. We validate our approach and implementation by fully replicating 10 projects, 8 of them industrial, based only on our model information. We measure the replication performance by measuring the errors during the process and completeness of the product. We measure the replication error, by tracking replication operations. We used the scope and completeness measure to enact this error. Completeness is measured by the instrumentation of a simple and scoped diff based on a third source of information. We present extensive results and interpretations. We discuss the threats to validity, the possibility of other approaches and the technological restrictions of our solution.

Keywords: model-driven Engineering, Migration, Software Analysis, Software Reuse, Reverse Engineering, Re engineering

1 Introduction

With the fast evolution of programming languages and frameworks, companies must evolve their systems. This evolution may imply the full migration of their applications to new technological environments. Our work takes place in collaboration with Berger-Levrault, a major IT company selling information systems developed in Microsoft Access among others. Microsoft Access is ageing and not

able to respond to the architectural needs of modern times. This migration is critical, since Berger-Levrault has more than 90 Microsoft Access applications.

Migration has been a research topic for a long time. The scientific community has proposed many different ways of tackling down this problem [3,6,8,9,10,11,16,22]. Nevertheless, migration is hard, tightly related with its circumstances and therefore, still challenging and not completely solved [20].

Most of the programming language compilers use plain text files as input for the programs they should compile or for configuration, such as XML, YML, properties, etc. Therefore, reengineering tools [12] often use the same approach for producing their internal models [6,8,11,13,16,21,22]. Work has already been done to mix static and dynamic analysis[3] [7,15,18,19]. However, not all the languages are based on text files. Some of them, such as Microsoft Access (Access for short), Oracle Forms, Flash, Flex and many other Rapid Application Development (RAD), use some kind of binary format. In our particular case, we study the applications developed in Access. Access uses a proprietary binary format for storing the programs. Due to this policy and the lack of exporting capabilities, an Access application lacks full text representation.

Knowing that our migration involves the splitting of code in between front-end and back-end, the reengineering of the UI from desktop application to web application and the backend into good quality micro-services.

We propose the next three research questions to lead the research:

#RQ1: Can we obtain an application representation by querying the IDE run-time?

#RQ2: Are we able to re-engineer the meta-data sources into a model useful for migration?

#RQ3: Is the obtained model suitable for migration?

Using these questions as general guidelines and getting inspiration from previous work on a different domain [1,2], we propose a model fully built on binary sources by applying reverse engineering on the run-time of the Access development environment, and re-engineering to transform the available data into an unified model.

Section 2 details the background of our research and in particular we describe Access as a *partially observable* domain and stress its *opacity*. We overview available technologies for accessing binary information. We propose an approach and implementation (Section 4) based on reverse and re-engineering taking into account the underlying challenges. We validate our approach and implementation by fully replicating 10 projects, 8 of them industrial, based only on our model information. We measure the replication performance by measuring the errors during the process and completeness of the product. We measure the replication error, by tracking replication operations. We used the scope and completeness measure to enacte this error. Completeness is measured by the instrumentation of a simple and scoped diff based on a third source of information. (Section 5).

[3] For space reasons, we do not consider tools performing analyses of system runtime. Such approaches instrument applications and produce various traces [4].

2 Microsoft Access: a Partially Observable System

Access is a relational database management system (RDBMS) that besides offering the relational Microsoft Jet Database Engine, also offers a graphical user interface and software-development tools. We briefly present and stress the key problems to extract information about Access applications.

2.1 Access

Visual Basic for Applications (VBA) is provided as a programming language. VBA is an object-based [23] extension of Visual Basic. Access is a fourth-generation language (4GL), comparable with Oracle forms or Visual Fox-pro. With the same mission of easing the GUI creation.

Access proposes a hybrid paradigm that aims to tackle down GUI, data storing and processings in a fully controlled and centralized environment. A program developed in Access solves problems by the orchestration of its first-class citizens: forms, modules, class, tables, queries, reports and macros.

To ease the work of GUI development, Access provides a point and click GUI Builder. As many other GUI builders in the market, such as Android Studio, Eclipse or Microsoft Visual Studio, Access also has to face the problem of distinguishing the generated content from the hand-crafted content. Android Studio uses the R class [4] for scoping generated code, Visual Studio.Net uses partial classes [5], and JavaFX uses xml files and annotations.

In the case of Access, Forms and Reports are split into two parts: (1) the VBA code, produced and modified only by the developer, (2) the component internal structure, produced and managed by the IDE, accessible to the developer only through point and click.

As many other 4GL languages, and Microsoft products, Access uses a proprietary binary format. This format organisation is undocumented, implying that attempting to extract data directly from the file would require a huge reverse engineering effort. Furthermore, Access uses entity specific formats for each first-class-citizen type, and in some cases, such as forms and reports, it has two formats, to respond to the internal division, required for managing code generation, explained above. This variety of formats leads to a more complex problem, threatening the generalization of a solution.

2.2 Limited Exporting

Access provides a visual interface to export some entities by point and click. This process is time consuming and prone to error. It is not tractable for full applications and in addition not all the elements can be exported. Leading to what we call a *partially observable* domain, since, by the usage of given tooling we cannot obtain artefact to analyze.

[4] https://developer.android.com/reference/android/R

[5] https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/partial-classes-and-methods

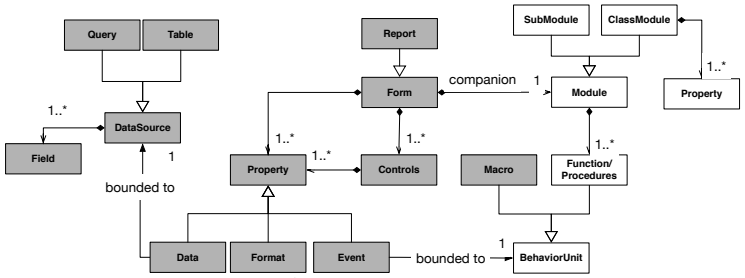


Fig. 1: Access simplified model

Figure 1 shows a simplified model of Access main elements. In grey we show the elements that **cannot** be exported from the GUI, in white those that can. Most of the structural entities are not available for export such as the table definitions, the query SQL definition, reports and forms structures not even the macros. The main GUI exporting features are related to the visual basic part of the project, including modules, class-modules, and the report or form companion-modules.The latter happens to be useless since their structure is not migrated.All analysis proposed over this partial content should be fully based on heuristics.

2.3 Programmatic Exporting

Access provides an undocumented function for programmatically exporting a text representation for all the first-class citizens. Using this function requires the implementation of specific software. This function is leveraged by third-party vendors that propose enhanced exporting features for control source version purposes. Solutions such as Oasis[6], Ivercy[7], MSAccess SVN[8], and others. We compare and extend on this subject in Section 7

2.4 Requirements & Constraints

Our work happens in the context of an industrial research on migration from Access to different kinds of technologies. We aim to migrate full applications from one monolith origin to a front-end and a micro service-based back-end (disregarding the database migration and restructuring since is out of scope).

The main features expected for the migration process are (1) selectiveness (the developer must be able to choose what he wants to migrate). (2) iterative (being able to propose short loops of migrations easy to verify and cheap to reject). (3) interactiveness (to be able to establish a dialog with the developer to better achieve the selective migration).

Such a migration process must coexist with an original project that is under maintenance and development. These requirements imply the following constraints: (1) The CPU and memory usage must be scoped to selective migrations

[6] https://dev2dev.de/index.php?lang=en
[7] https://ivercy.com/
[8] https://archive.codeplex.com/?p=accesssvn

(the migration solution must be able to run cheaply in the working environment of the developers without performance penalties). (2) The model must be able to supply as much data as possible. (3) The model must be able to supply up-to-date data (all modifications should be reflected immediately in the model).

Following the direction of [16,8] that work over the model of Oracle Forms, we recognize the importance of having a model based on the first-class-citizens of the language. Following the abstract idea behind [14], we propose a representation close to the language, that responds to Figure 1.

3 COM Technological Overview

In this Section we provide a technological overview that will be used to enumerate the challenges of a solution based on the usage of Access COM API.

3.1 Microsoft COM & Access

Through COM, Access exposes a large and powerful API, that allows high interoperability in between different applications.

Access documentation [9] is heterogeneous. It provides good quality content for the popular usages, but vague, superfluous or even nonexistent for less popular usages. We insist on a technological overview that will help us answer our #RQ1, and shed light on the challenges.

3.2 Data Access

For interacting with Access through COM we must interact with an object model, composed by the followings.

Remote handle. For interacting with remote Access entities COM provides remote memory addresses. We call these addresses handles.

Application. First instance to access through COM. This application object is bound to a running instance of Access. It exposes an explorable API, and it allows access to the project components, directly or indirectly.

DoCmd. (Do Command) is an object that reifies most of the available operations to apply on the application. It must be used for opening a project, databases and others. Most of the objects below have this object as a dependency.

References. This collection contains *Reference objects* describing a project's static dependencies.

CurrentProject. Depends on *DoCmd*. It holds basic metadata for each element in the project, by pointing to the collections *AllForms, AllReports, AllMacros, AllModules* that contains objects describing each form, report, macro and module correspondingly.

[9] https://docs.microsoft.com/en-us/office/vba/api/overview/access

CurrentData. Depends on *DoCmd*. It holds metadata for each element related with data structures. In this object the available collections are *AllTables, AllQueries* that contains objects describing each table and query correspondingly.

DbEngine. Depends on *DoCmd*. It is the main access point to the data model. It provides access to *workspaces*.

Workspace. Depends on *DbEngine*. Represent database schemes, and provides access to the scheme elements by pointing to the collections *QueryDefs* and *TableDefs*.

TableDef and QueryDef. Depends on *Workspace*. Each of these objects contains a description. For the TableDefs name and *fields*. For the QueryDefs name and SQL.

Forms, Reports and Modules. Depends on *DoCmd*. Finally, we have three main collections where we can find the Form, Report and Module objects with their inner composition. This internal definition includes composed controls (textbox, labels, etc.), properties (layout, naming, companion-module, etc) and VBA source code.

3.3 COM Model Re-engineer Challenges

To re-engineer COM data into an unified model has challenges:

The reading of a property of the COM entity, may give back another COM entity. In some cases, we are going to read native type or self-contained information. But in some other cases the value of an attribute may be another handle. For these cases, we have to map the read value with a model entity type.

One model entity may correspond to more than one COM entity. The COM model provides two different objects for representing each of the first-class citizens. By example, *AllForms* contains form's metadata, *Forms* contains a form internal representation.

One COM entity type may be mapped to different model entity types. Most of the objects in the COM model have properties represented with the same type, but to be able to structure the analysis (visiting, for example), we need them belonging to different classes. This implies that some of the entities with the same type may have to be mapped to different types in our model.

Loadable objects. The first-class citizen objects must be dynamically loaded to reach their internal information. For loading, they require access to many COM entry points. This implies that some objects require specific extra steps for acquiring the desired data.

Summary. The overview shows that COM delivers a large access to the binary model of an Access application. This remote binary representation (from now on **COM model**) is also a very low level model that responds to the need for interaction between applications. We also understand that an approach in this direction must respond at least to the traditional challenges of reverse and re-engineering processes.

4 Mixing Static and Internal Access information

To answer *Are we able to re-engineer the meta-data sources into a model useful for migration?*, (#RQ2), we propose the approach of online model from the point of view of a migration, followed by an implementation of the proposed approach and an explanation of how our approach and implementation address each constraint and challenge stated above.

4.1 Approach

As a general approach we propose to define our model as an online projection of the COM model. By **online** we mean that all the data is obtained by the COM bridge, therefore, any change done in the code impacts immediately our model. For achieving this we propose to let our model use COM as a back-end. Our model must conform to the meta model proposed in Figure 1. By delegating to COM we aim to get all the possible data to be gathered from the analyzed software on demand. We expect this strategy to give immediate feedback and to allow quick and agile modifications over the used information, without requiring to do further extractions reducing the need of planning the data (constraint stated in Section 2), and allows us to implement quick migration experiments.

4.2 Architecture Implementation

As general architecture we propose to create a model that uses the COM model as a back-end as shown in the Figure 2 We propose lazy access to the COM model

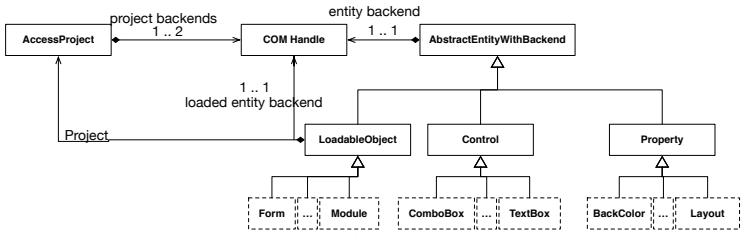


Fig. 2: Architecture

back-end, what will guarantee that we access and load only what is needed. This feature aims to limit the memory usage (constraint stated in Section 2) by construction. The lazy approach will also allow us to map each binary-model-entity to a model-entity one at a time. We also propose to cache the results, for reducing the COM calls and therefore CPU time and inter-process communication.

Regarding the mapping between the COM model entity-type and our model, we propose to use two kinds of mapping: by type and by attribute value. First-class citizen entities are represented by two COM models, and that is why all

of them subclass from a *LoadableObject* class, which maps two COM models instead of one.

For mapping the binary-model-entities to model-entity types, we propose to use factories. The mapping factory by type maps one binary-entity-type to one model-entity-type. The mapping factory by attribute value maps one binary-entity to one specific model-entity-type according to one specific binary-entity value.

4.3 Microsoft Access Model Implementation

Our model extends from the architecture implementation, and inherits the mapping to COM remote entities. This model is meant to be visited by a visitor pattern, in order to perform analysis. In order to define the structural composition to be visited, it relies on the usage of stateful traits. At the level of stateful traits we define, by example, the widget - control composition.

4.4 Meeting the Challenges and Constraints

The reading of a property of the COM entity, may give back another COM entity Each model type must know which readings will give back a COM model entity. In these reads we use a factory that maps the COM model type with a model type. After creating a new instance, it sets up the given COM model entity as a back-end.

One model entity may correspond to more than one COM entity There are two kind of objects with more than one back-end, the first-class citizens, subclass of LoadableObject, and AccessProject, that is a convenient class for managing the generality of COM usage. Since there are only two specific cases they are treated individually.

One COM entity type may be mapped to different model entity types While loading properties we use a factory pattern that defines the class to instantiate according to the name of the property. Since the only COM model entity with this kind of mapping is the property, we did not generalize this kind of mapping.

Loadable objects For loadable objects, we defined a specific branch in the hierarchy, that before accessing to remote properties related to the loaded object back-end, it ensures that the back-end has been loaded and bounded. For ensuring this, the Loadable Object subclass does a typed double dispatch with the AccessProject, which delegates to DoCmd.

Contain computational resources usage If we want to access all possible data, we risk having a model that is too big to be managed. For approaching a solution to this problem, we propose to specialize the access on demand in our implementation by using **lazy loading** and **cache**. Lazy loading scopes the memory usage to the effectively needed data. Cache scopes the inter-process communication and CPU time for remote access to one time per object.

Accessing all accessible data Our proposal is conceived to get data on demand. This is why the very nature and particularity of this model is to be connected to the Access development runtime. If the data is reachable by COM, therefore it should be accessible.

Accessing up-to-date data The online nature of our approach is the main key for ensuring up-to-date data, since the data obtained is meant to be obtained from the developer's Microsoft Access running instance.

5 Validation

Our validation is aligned with #RQ3: *Is the obtained model suitable for migration?* Since our model is meant to be used for migration, we propose to fully migrate some projects to the same technology. That is to say to replicate or clone. For doing so, we perform a replication of 10 different access projects. For this performance we use our model and the COM extensions to produce the replicated project programmatically.

5.1 Methodology

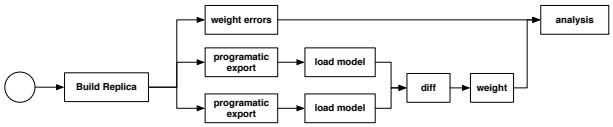


Fig. 3: Validation methodology overview

Chosen projects For this validation we used 10 different projects (described in Table 1). 8 of them are base libraries used by Berger-Levrault in all the Access projects. One is an open source example found in GitHub [10]. The last one is the Microsoft Northwind Traders (Northwind for short) [11] example project. This project is used for learning Access and it uses most of the standard techniques and available graphical features of the language.

Error tracking & weighting All errors happening during the replication process are tracked down for further analysis and correlation with the original / replica comparison. The error tracking composition responds to the same composition as the proposed model. In a nutshell, we count the operations required to replicate the project. We obtain a proportion of errors by contrasting this counting according to the outcome: successful or failure. More details including the formula are given in the annex Section 10.

[10] https://github.com/Access-projects/Access-examples
[11] https://docs.microsoft.com/en-us/powerapps/maker/canvas-apps/northwind-install

Project	Remote Table	Remote Table Query	Query	Module	Classes	Report	Forms
Northwind	0	20	0	27	6	2	15	34
CUTLCOMP	0	1	0	0	3	0	0	0
CUTL	7	3	0	1	26	62	0	8
CRIR	5	4	0	16	6	0	2	3
CPDI	0	1	0	0	2	0	0	0
CHABIL	11	2	0	27	8	1	1	10
CDDE	0	1	0	0	2	0	0	0
CAUNIT	0	1	0	0	4	15	0	1
ACCUEIL	25	7	0	13	6	67	5	33
Access Examples	0	10	2	14	11	1	8	13

Table 1: Projects descriptions

Programmatic export for obtaining file definitions For obtaining the file definition of each component we leverage the COM function named SaveAsText (this function is explained in Section 7). The output of this function differs by each type of entity as follow: For Modules and class-modules it offers VBA (.bas) files. For Queries it offers an SQL output. For tables it uses XML format. For Forms and Reports, offers an output that hybridizes the structure definition and the VBA code of the companion-module.

File diff & weighting We instrument a diff in between pairs of files of each original / replica exported project. The result of this diff is a differential graph expressing all the required operations for transforming the original project to the replica project. In a nutshell, we count the comparisons and contrast according to the outcome: added/removed element, modified value, exact replication. More details including the formula are given in the annex Section 11.

5.2 Results

Table 2 offers an overview of the process of replication of each of our projects. Most of the main elements are replicated. The tables and queries not replicated happen to be remote tables that we cannot access, since we lack access to the remote server.

Table 2: Export overview

	Reference	Table	Query	Module	Report	Forms	Total
#Elements	70	98	100	222	31	102	623
#Replicated	69	50	98	222	31	102	572
#Failures	1	48	2	0	0	0	51

Below compare the different aggregation of completeness with the aggregation of error tracking per project per type of element. We do not include modules nor macros because the result is **complete** in all the projects. Since both completeness and error series respond mostly to a hyperbolic distribution, we propose to measure the rate of success and error by using the median.

Table 3: Query comparison

Queries								
Projects	**Completeness**				**Failures**			
	Max	Min	Median	SDev	Max	Min	Median	SDev
Northwind	100	100	100	0	15	0	0	3.05
CUTLCOMP	–	–	–	–	–	–	–	–
CUTL	100	100	100	0	0	0	0	0
CRIR	100	88.46	100	2.88	0	0	0	0
CPDI	–	–	–	–	–	–	–	–
CHABIL	100	93.49	100	1.25	0	0	0	0
CDDE	–	–	–	–	–	–	–	–
CAUNIT	–	–	–	–	–	–	–	–
ACCUEIL	100	95.76	100	1.17	0	0	0	0
Access Examples	100	0	100	25.81	100	0	0	34.15

Table 4: Table comparison

Tables								
Projects	**Completeness**				**Failures**			
	Max	Min	Median	SDev	Max	Min	Median	SDev
Northwind	100	0	100	23.72	22	0	0	5.14
CUTLCOMP	100	0	100	27.62	0	0	0	0
CUTL	100	0	98	5041	10	0	0	46.43
CRIR	100	0	99	46.04	100	0	0	44.42
CPDI	100	0	100	27.62	0	0	0	0
CHABIL	100	0	0	50.65	100	0	0	50.38
CDDE	–	–	–	–	–	–	–	–
CAUNIT	100	98	100	0.75	0	0	0	0
ACCUEIL	100	0	0	49.6	100	0	100	50.63
Access Examples	100	0	99.5	32.34	90	0	0	17.65

Table 3 show some very good results. We fully replicate most of the queries. Table 4 has also very good results, since most of the failures happen on tables that are remote. **CUTL**, **CRIR**, **CHABIL**, **ACCUEIL** and **Access Examples**, all of them high standard deviations, and all of them have remote tables. There are some cases where there are no errors during the process, but we don't meet full completeness, such as the cases **CUTLCOMP**, **CAUNIT** and **CPDI**. These cases happen because system tables are not replicated, and the replication targets a newer Access version.

Table 5: Form comparison

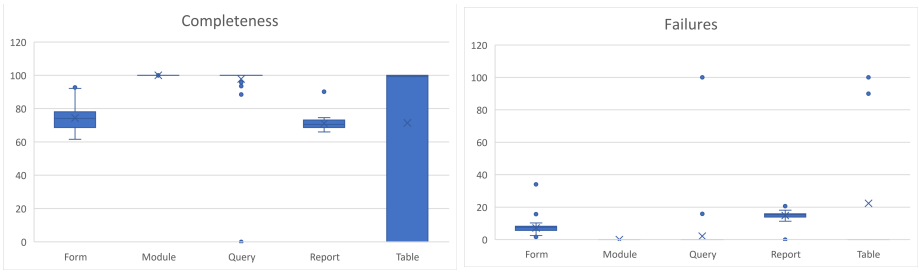
Forms								
Projects	**Completeness**				**Failures**			
	Max	Min	Median	SDev	Max	Min	Median	SDev
Northwind	86.24	61.55	69	4.72	8.68	5.32	9	0.73
CUTLCOMP	–	–	–	–	–	–	–	–
CUTL	86.73	63.46	82	8.45	7.62	1.66	5	1.79
CRIR	74.73	73.06	74	0.96	8.31	8.18	9	0.064
CPDI	–	–	–	–	–	–	–	–
CHABIL	78.3	65.89	70.5	4.14	8.95	5.44	6	1.134
CDDE	100	98	100	0.83	0	0	0	0
CAUNIT	79.96	79.96	79.96	0	5.27	5.27	5.27	5.27
ACCUEIL	92.71	70.01	78	6.22	15.71	2.49	6	2.7
Access Examples	88.52	61.91	76	8.35	34.05	6.6	9	7.25

Table 6: Report Comparison

Reports								
Projects	**Completeness**				**Failures**			
	Max	Min	Median	SDev	Max	Min	Median	SDev
Northwind	71.55	65.97	70	1.5	16.57	11.33	16	1.3
CUTLCOMP	–	–	–	–	–	–	–	–
CUTL	–	–	–	–	–	–	–	–
CRIR	73.13	71	73	1.5	20.64	15.64	18.5	3.54
CPDI	–	–	–	–	–	–	–	–
CHABIL	71.21	71.21	71.21	0	13.87	13.87	13.87	0
CDDE	–	–	–	–	–	–	–	–
CAUNIT	–	–	–	–	–	–	–	–
ACCUEIL	74.57	73.34	74	0.53	14.16	13.87	14	0.13
Access Examples	90.14	66.1	72.5	7.48	18.15	13.87	16	1.51

In the particular case of reports and forms (Table 5, Table 6) we see less interesting outcomes from the point of view of completeness, but we can observe an inversely proportional relation with the errors. There is also a restriction of implementation, many values that are stored in byte array structures, even if we can read them, we cannot write them. This makes impossible the replication of printing configuration, custom controls based on ActiveX or OCX technologies and image contents. These properties do not figure in between the errors, because they are avoided by construction of the process. Finally, Figure 4 provides an idea of the confidence interval of the measures. Both sides show a correlated existence of isolated measures. In the case of the *Tables* the completeness confidence is too large. We can relate it with the scattered error measures. Forms and Reports completeness show shorter intervals, that we can correlate with the error intervals and the distance between the isolated cases. Finally, modules and queries have

a really good interval. The centre is placed almost in 0 in the failures plot, even having some isolated cases themselves.

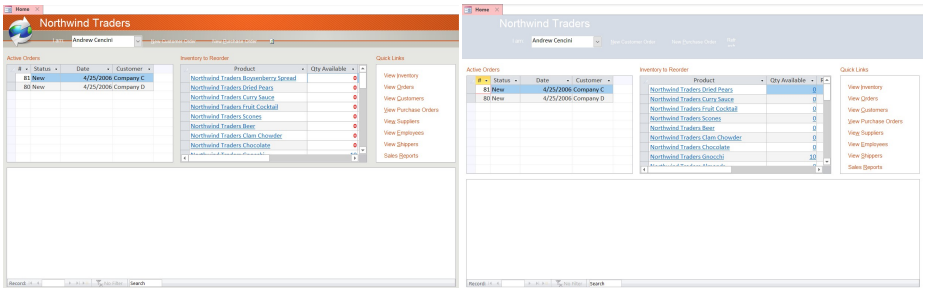


(a) Completeness (b) Failures

Fig. 4: Confidence

5.3 Human Insight and Opinion

We check each of the replicas and compare manually with the original, and also with the extracted data. Most of the meaningful parts of the applications were properly replicated, even when some of the most appealing graphical features are not maintained because of the impossibility of writing this kind of data. Nevertheless, we spent time specially on the execution of many functionalities of the project Northwind and found out that most of the behaviors are maintained, since all the macros and source code has been correctly replicated and bound to the proper structures. After migrating all the example data available from the original to the replica, we can observe that the login works as expected in all the tests we manually checked. Figure 5a and Figure 5b give material evidence of the outcome, by exposing the most complex form in the replicated system. All this insight is highly positive. Our most positive but opinionated insight is that we achieve to develop the validation faster than we expected, thanks to the model that we are presenting. We got constant assessment from it getting fast feedback and understanding of the replication process target.



(a) Original Home screen (b) Replicated Home screen

6 Threats to Validity

Empirical study. Our validation is based on the replication of ten projects. This gives about 534 first-class-citizen components, thousands of controls and table fields. There may exist many kinds of projects that are not represented by those that we have.

Multiple versions. We have seen how the non-replication of system tables available in other versions of access came out as a difference in between the original and replicate code because of the policy of non-exporting system tables. This does not happen to be a false positive. And we did not find any false positive or false negative, but we cannot completely ensure yet with this validation this cannot happen in other projects.

Undocumented features leveraging. For allowing the file comparison done for our validation, we had to leverage some undocumented functions (widely explained in Section 7). This function could change or not be accessible anymore, threatening the reliability of the process.

7 Discussion

Source version control-oriented solution. As we pointed out in Section 2, there are third party solutions for source control that could be helpful for solving this problem. This software produces different text formats able to reproduce the exact same project. The available tools developed for source control are based on the usage of **SaveAsText** undocumented function provided by Access DoCmd command. This function exports each entity to a text format, producing XML for the tables, VBA for the source code and an Access specific DSL for defining forms and reports. Their stability is tested already for many years by the market, meaning that could be a good starting point for software analysis.

Undocumented features. Even taking into account the fact that these tools have been in the market for a long time, we do not really know how they have changed during their lifetime. As we pointed just above, these solutions are based on the usage of the undocumented functions SaveAsText and LoadFromText. This presents two risks: (1)Microsoft may change their behavior, or even make them unavailable in the future,(2) the format of the exported text has no documentation either, which means that different versions could have singularities.

Context and performance. Besides that, our approach of software migration is based on augmenting the developer. For this reason, we see it more useful to be able to see exactly what the developer is actively working on, to have more context and insight. Finally, we see that delegate the management of the information to the same access and using the IDE as a database has a high potential for reducing memory consumption and model complexity, allowing us to develop tools that can run on a working environment, without requiring extra infrastructure.

What our validation does not validate. The exported files have the minimal amount of information required to build again the whole application. This is a lot. But it does not include default values. While in a file, a complex component may define about ten different properties, when accessed through COM, we have access to more than 100 properties per control. This means that the text representation reveals to be incomplete, *partially observable*. From the point of view of software analysis and migration, to have systematic access to default values without having to manually specify them is a great asset.

8 Related Works

OMG in [13] and [21], in the context of Architecture driven modernization, proposes successive transformations over the extracted Abstract Syntax Tree model (ASTM) [14], which is obtained by parsing source code. [6] Claims an efficient implementation of model-driven Engineering (MDE) with models obtained by parsing source code and obtaining an ASTM. [16] proposes to reverse engineer GUI Layouts from Oracle Forms. They export the Form structure as XML and use EMF [12] tools for generating models. [11] for the particular case of analyzing flex UI, it proposes the usage of Adobe Wallaby [13] for transforming Flash's SWF binary proprietary format files into HTML, and then parsing. [8] proposes to parse all the files representing an Oracle Forms for obtaining a model the article does not specify the kind of file they used for analyzing But according to [16] and to the existence of Oracle exporting tools [14] from Form to XML we suspect that they follow the same path. [22] proposes the usage of Famix [5]. Famix extraction for Java applications is achieved by using VervaineJ [15]. This library transforms the source code to a Family of Languages representation. One of the main differences with an AST is that it is a Graph with scoping and it binds all the static relationships. To reach this deeper knowledge, it analyses the application by parsing its files in the context of the eclipse java compiler. To the best of our knowledge, at least within the MDE based approaches: (1) there is no other model extraction technique but by parsing: either the source code of the source application or the exportation from a binary format to a text format (for example XML/HTML). (2) The approaches are based on batch processing instead of online access. Some of these model extracting tools VervaineJ, Proleap [16], SMACC [17], ANTLR [18], and many others. [19] overviews several dynamic (based on run-time analysis) reverse engineering techniques and their challenges in the context of the software migration and evolution. [18] Proposes a hybrid analysis approach for reverse-engineering web applications, obtaining a model

[12] http://www.eclipse.org/modeling/emf/.
[13] https://en.wikipedia.org/wiki/Adobe_Wallaby
[14] https://blogs.oracle.com/apex/forms-to-apex-converting-fmbs-to-xml
[15] https://github.com/NicolasAnquetil/VerveineJ
[16] https://github.com/uwol/proleap-vb6-parser
[17] https://refactory.com/smacc/
[18] https://www.antlr.org/

by crawling the widgets from the run-time, and augment the results by parsing the event handlers code and recognize what are the possible navigation options. [17] Points out the complexity of accurate GUI analysis by code interpretation. Extracts a technology-agnostic UI model by crawling the application run-time using AOP, for enabling portability to android.

9 Conclusion & Future Work

Contribution. In this article we explained the problem of opacity in Access. We enunciate three research questions for our work. We offered a technological overview for using COM as a bridge for accessing data, aiming to answer #RQ1. We offered the novel approach and implementation of model to shape a COM model into an application model, answering #RQ2. We conducted an exhaustive and detailed validation process guided by and answering #RQ3. We offered a compendium of the threats to validity that we found during our experiments. We compared our solution with related work and proposed a discussion on why to choose the usage of COM over exported files. We position our work with the state of the art on the part of software migration and how software is analyzed towards this goal.

Future. From this point we have several paths opening. Adapt parsing techniques over the modules, class-modules and companion-modules, for being able to build a full AST on demand, and being able to control its creation on demand, without losing reference information. Find the minimal migration from Access to Angular/TypeScript based in our online metamodel.

References

[1] Bragagnolo, S., Marra, M., Polito, G., Boix, E.G.: Towards scalable blockchain analysis. In: 2019 IEEE/ACM 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB). pp. 1–7 (2019)
[2] Bragagnolo, S., Rocha, H., Denker, M., Ducasse, S.: Smartinspect: solidity smart contract inspector. In: 2018 International Workshop on Blockchain Oriented Software Engineering (IWBOSE). pp. 9–18 (mar 2018). Electronic ISBN: 978-1-5386-5986-1
[3] Brant, J., Roberts, D., Plendl, B., Prince, J.: Extreme maintenance: Transforming Delphi into C#. In: 2010 IEEE International Conference on Software Maintenance. vol. Software Maintenance (ICSM), 2010 IEEE International Conference on, pp. 1–8 (2010)
[4] De Pauw, W., Jensen, E., Mitchell, N., Sevitsky, G., Vlissides, J.M., Yang, J.: Visualizing the execution of java programs. In: Revised Lectures on Software Visualization, International Seminar. pp. 151–162. Springer-Verlag, London, UK (2002)
[5] Ducasse, S., Anquetil, N., Bhatti, U., Cavalcante Hora, A., Laval, J., Girba, T.: MSE and FAMIX 3.0: an Interexchange Format and Source Code Model Family. Tech. rep., RMod – INRIA Lille-Nord Europe (2011)
[6] Fleurey, F., Breton, E., Baudry, B., Nicolas, A., Jezéquel, J.M.: Model-Driven Engineering for Software Migration in a Large Industrial Context. In: Engels, G., Opdyke, B., Schmidt, D.C., Weil, F. (eds.) Model Driven Engineering Languages and Systems. vol. 4735, pp. 482–497. Springer Berlin Heidelberg, Berlin, Heidelberg (2007)
[7] Francesca, A.F., Fabrizio, P., Claudia, R., Stefano, R.: Behavioural design pattern detection through dynamic analysis. In: Proceedings of 4th PCODA at the 15th Working Conference on Reverse Engineering (WCRE 2008). pp. 11–16 (2008)
[8] Garcés, K., Casallas, R., Álvarez, C., Sandoval, E., Salamanca, A., Viera, F., Melo, F., Soto, J.M.: White-box modernization of legacy applications: The oracle forms case study. Computer Standards & Interfaces pp. 110–122 (Oct 2017)
[9] Govin, B., Anquetil, N., Etien, A., Ducasse, S., Monegier Du Sorbier, A.: Managing an Industrial Software Rearchitecting Project With Source Code Labelling. In: Complex Systems Design & Management conference (CSD&M). Paris, France (Dec 2017)

[10] Govin, B., Anquetil, N., Etien, A., Monegier Du Sorbier, A., Ducasse, S.: How Can We Help Software Rearchitecting Efforts ? Study of an Industrial Case. In: Proceedings of the International Conference on Software Maintenance and Evolution, (Industrial Track). Raleigh, USA (Oct 2016)

[11] Hayakawa, T., Hasegawa, S., Yoshika, S., Hikita, T.: Maintaining web applications by translating among different RIA technologies. GSTF Journal on Computing p. 7 (2012)

[12] Kienle, H.M., Müller, H.A.: The tools perspective on software reverse engineering: Requirements, construction, and evaluation. In: Advanced in Computers, vol. 79, pp. 189–290. Elsevier (2010)

[13] Newcomb, P.: Architecture-driven modernization (adm). In: 12th Working Conference on Reverse Engineering (WCRE'05). pp. 237–237 (2005)

[14] Object Management Group: Abstract syntax tree metamodel (ASTM) version 1.0. Tech. rep., Object Management Group (2011)

[15] Richner, T., Ducasse, S.: Recovering high-level views of object-oriented applications from static and dynamic information. In: Yang, H., White, L. (eds.) Proceedings of 15th IEEE International Conference on Software Maintenance (ICSM'99). pp. 13–22. IEEE Computer Society Press, Los Alamitos CA (Sep 1999)

[16] Sánchez Ramán, O., Sánchez Cuadrado, J., García Molina, J.: Model-driven reverse engineering of legacy graphical user interfaces. In: Proceedings of the IEEE/ACM International Conference on Automated Software Engineering. pp. 147–150. ASE '10, ACM (2010)

[17] Shah, E., Tilevich, E.: Reverse-engineering user interfaces to facilitate porting to and across mobile devices and platforms. In: Proceedings of the compilation of the co-located workshops on DSM'11, TMC'11, AGERE! 2011, AOOPES'11, NEAT'11, \& VMIL'11. pp. 255–260. ACM (2011)

[18] Silva, C.E., Campos, J.C.: Combining static and dynamic analysis for the reverse engineering of web applications. In: Proceedings of the 5th ACM SIGCHI Symposium on Engineering Interactive Computing Systems. p. 107. ACM Press (2013)

[19] Stroulia, E., Systä, T.: Dynamic analysis for reverse engineering and program understanding. SIGAPP. Applied Computing Review **10**(1), 8–17 (2002)

[20] Terekhov, A.A., Verhoef, C.: The realities of language conversions. IEEE Software **17**(6), 111–124 (Nov 2000)

[21] Ulrich, W.M., Newcomb, P.: Information systems transformation: architecture-driven modernization case studies. Morgan Kaufmann (2010)

[22] Verhaeghe, B., Etien, A., Anquetil, N., Seriai, A., Deruelle, L., Ducasse, S., Derras, M.: GUI migration using MDE from GWT to Angular 6: An industrial case. In: 2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER). Hangzhou, China (2019)

[23] Wegner, P.: Dimensions of object-based language design. In: Proceedings OOPSLA '87, ACM SIGPLAN Notices. vol. 22, pp. 168–182 (Dec 1987)

10 Annex 1: Error tracking & weighting

Error tracking COM is not intensively used to create projects programmatically. Many standard procedures, after running, make Access unstable and easy to fail in any next attempt of modification. Failures that may imply from non created widgets to missing properties. During the whole process we track down all the errors happened during this process, for being able to plot alongside with the results on the comparisons. This error is tracked down at the level of replication operation and typified.(1) ChildCreatedSuccessfully (2) FailureToCreateChild (3) FailureToWriteProperty (4) PropertyWrittenSuccesfully. The tree of error tracking composition responds to the same composition as the proposed model.

Error weighting We measure the failure of a replication process, by the weighting and summarization of the tree of operations.

Let o be the result of an operation of replication. Let c_o be the children creation operation under the scope of the operation o. Let p_o be the properties creation operation under the scope of the operation o.

$$F(o) = \begin{cases} 1 & o \in \{Failure\} \\ 0 & o \in \{Success\} \wedge |c_o| = 0 \wedge |p_o| = 0 \\ (0.9\frac{\sum_{i=1}^{|c_o|} F(c_o i)}{|c_o|} + 0.1\frac{\sum_{i=1}^{|p_o|} F(p_o i)}{|p_o|})0.5 & o \in \{Success\} \wedge |c_o| > 0 \\ \frac{\sum_{i=1}^{|p_o|} F(p_o i)}{|p_o|} & o \in \{Success\} \wedge |c_o| = 0 \end{cases} \quad (1)$$

This recursive function calculates the proportion of error in terms of errors in terms of composed errors. For our work those elements that are composed of elements (by example, the controls inside a form) are specially represented by their children. This is why one formula branch uses coefficients: 10% based on the component properties, and 90% on the children completeness.

11 Annex 2: File x File Diff & weighting

File diff For being able to diff each pair of files we used different techniques Modules, Macros, and Queries are loaded as nodes with name and plain text content. Tables are loaded as XML trees including name, indexes and fields with their name and type. Forms, Reports are loaded with a simple parser that produces a tree of report/form with their controls and properties.

Each of these entities are loaded from original and replica. For each pair we calculate the differential tree expressing all the needed operations for transforming the original graph into the replica graph. We define the following operations: (1) Add (2) Remove (3) Same (4) ModifyChild (5) ModifyProperty.

Diff weighting We measure the completeness of each of the elements on the differential graph. Let u be the result of comparing an element from the original project with its equivalent of the replica.

$$Completeness(u) = (1 - M(u)) * 100) \quad (2)$$

Magnitude $M(u)$ is the weighting of the difference in between two elements. Let u_o and u_r being respectively original and replica side of u. Let c_u be the set of children that belong to the u. Let p_u be the set of properties that belong to the u.

$$M(u) = \begin{cases} 1 & u \in \{Add, Remove\} \\ 0 & u \in \{Same\} \\ (0.9\frac{\sum_{i=1}^{|c_u|} M(c_u i)}{|c_u|} + 0.1\frac{\sum_{i=1}^{|p_i|} M(p_u i)}{|p_u|})0.5 & u \in \{ChildModif\} \wedge |c_u| > 0 \\ \frac{\sum_{i=1}^{|p_u|} M(p_u i)}{|p_u|} & u \in \{ChildModif\} \wedge |c_u| = 0 \\ u_r - u_o & u \in \{PropertyModif\} \wedge u_r, u_o \in \{Native\ type\} \end{cases} \quad (3)$$

This recursive function calculates the magnitude of the difference in terms of the composed differences. The coefficients used in this formula respond to the same explanation as those used on the error weighting formula explained above.