Mainframe Migration based on Screen Scraping

Sergio Flores-Ruiz, Christoph Domann, Simona Puica

itestra GmbH
Destouchesstr. 68, 80796, München, Germany
[flores | domann | puica]@itestra.de

Ricardo Perez-Castillo

Information Systems and Technology Institute (ITSI)
University of Castilla-La Mancha
Paseo de la Universidad 4, 13071, Ciudad Real, Spain
ricardo.pdelcastillo@uclm.es

Abstract—Companies possess a history and large array of legacy information systems that consume a great part of their IT budget in operations and maintenance. These systems are mission-critical, and they cannot be fully discarded since they retain business rules and provide information that is not available anywhere else. Unfortunately, decades-old legacy systems cannot easily withstand modification. Mainframes specifically conglomerate most of these legacy systems. Although there are some white-box solutions for migrating mainframe systems, such solutions lack systematicity and do not provide mechanisms for verifying business rules preservation. Hence, this paper presents a black-box solution (ignoring the internal structure of COBOL programs) which uses a screen scraping technique for migrating mainframe systems toward JavaFX and relational databases. Together with this solution, this paper provides an automatic verification technique to check if the recreated system reflects all the embedded business logic. This proposal has been designed and developed in the context of an industrial project, in which the solution has already migrated 43,000,000 mainframe screens from four systems. The main implication for researchers and practitioners is that screen scraping has proved to be feasible for migrating mainframe systems in large-scale projects within a manageable time-frame while preserving business.

Keywords—Screen Scraping; Mainframe; Legacy Information Systems; COBOL; Migration; Verification.

I. INTRODUCTION

Major corporations usually possess a history and large array of Legacy Information Systems (LIS) that consume a great part of their IT budget in operations and maintenance [1]. These systems have been extended over many years to support business processes. Other times, business processes have been adapted over time to work efficiently with an existing system and the workforce is used to the established process. A lot of business knowledge is implemented in these mission critical systems, up to the point that certain aspects of processes are not documented elsewhere but in the source code [2, 3]. Additionally, these systems are the foundations and building blocks for newer systems [4]. Since LIS are essential from the business point of view, modification of these systems has a high risk. Furthermore, LIS are very expensive to be fully discarded. Thus, organizations must consider modernizing these legacy systems to remain viable [5].

LIS' challenges have a larger impact in some specific business domains. For example, insurance companies have LIS which must manage life insurance contracts for decades. According to the law in many countries, the contracts have to

be stored for several years [6, 7]. The lifecycle of LIS, managing such contracts over the years, covers a lot of changes in the technology landscape, programming languages and paradigms and even staff turnover [3].

LIS are thus an undesired yet unavoidable reality for many of these companies. COBOL (Common Business-Oriented Language) programs that run in mainframes support the core business logic of large insurance and financial companies [8]. Mainframes force organizations to incur extremely high costs due to payments for processing power [9] and hiring programmers that master such technologies.

Many techniques and methods have been proposed in the literature for migration, replacement or modernization of mainframe/COBOL systems to modern technologies [8, 10-12]. Unfortunately, these techniques cannot be applied extensively in a systematic way due to two main limitations. First, and especially in mission-critical systems, there is a lack of integrity assurance controls to validate that after migration the whole business logic and all legacy data are preserved without defects. Second, even when integrity and quality assurance mechanisms exist, these are time-consuming and, in practice, not feasible to be executed in a manageable time.

To mitigate these issues, black-box modernization approaches have been alternatively proposed in the literature [5]. Black-box modernization involves examining the input and output of a legacy system within an operating context to gain an understanding of the system interfaces. In contrast with white-box modernization, black-box does not require reverse engineering of LIS. Therefore, the scope of verification is also reduced, requiring less verification efforts compared to white-box approaches.

This paper presents a black-box migration method and an automatic verification solution proposed in the context of an industrial project for a German insurance company. The migration method is specially built to migrate a mainframe system in COBOL and an IMS database in PL/I toward a JavaFX system plus a relational database in DB2. The migration method consists of a screen scraping technique that systematically navigates and interacts over all available mainframe screens and collects data in a relational schema. Then, the mainframe screens are reconstructed in a JavaFX system which consumes the collected data and faithfully recreates all screens. This paper also introduces a technique to be applied after migration so that logic and data can be verified. This technique consists of a robot that detects inconsistencies by automatically and in parallel navigating

through the old screens in the LIS and the recreated ones in the new system.

The proposed method has to be applied to different mainframe subsystems and, although the industrial project is still underway, preliminary results show that millions of insurance contracts can be migrated without loss of information.

As a result, the main contributions of this paper are the following: (i) design and development of a screen scraping solution for migrating mainframe systems; (ii) design and development of a migrating verification solution to automate the business rule preservation check; (iii) demonstration of the applicability of the aforementioned solutions to a large-scale, real-life project.

The remainder of this paper is organized as follows: Section II briefly introduces the terminology employed in this paper. Section III shows the related work. Section IV introduces the mainframe migration project as the context of the proposed solution. Section V presents in detail the screen scraping technique, while Section VI shows the automatic screen verification method. Section VII provides some preliminary results of the migration and verification in the underlying industrial project. Lastly, Section VIII gives a summary of the conclusions reached.

II. TERMINOLOGY

This section provides a definition of the core concepts and elements used throughout the paper.

Dialog: it is a model that defines the kind of information shown by the mainframe system after selection. Dialogs have unique names and specific purposes. Dialogs are instantiated for different policy contracts with specific information.

Screen: it is the instantiation of a dialog. The screen provides part of the information about a specific policy contract. The terminal can display only one screen at a time.

Field: fields define the dialog structure depending on the type of information contained and their coordinates (row and column number). Dialogs are composed by fields.

Field definition: it represents the abstract definition of a field in the specific dialog. This definition consists of (i) a label, (ii) the type of the data, (iii) length of the data (digits or characters), and (iii) the coordinates in the dialog.

Field value: the specific value of a field contained in the screen instantiated for a certain dialog.

Contract: represents a unique insurance contract in the system. Contracts are represented in several screens with different dialogs depending on the nature of the contract. The mainframe system supports different types of insurance contracts.

III. RELATED WORK

This section presents work related to the migration of legacy mainframe systems and screen scraping techniques.

A. Mainframe migration

Various proposals for migrating mainframe systems to modern languages exist in the literature. *Sneed et al.* [10] migrated legacy COBOL programs to Java. This work provided a technique for reengineering COBOL code prior to migration. The goal of the reengineering was to free the code running on an IBM-AS400 from proprietary dependencies and to reduce them to the pure business logic. *Brito et al.* [13] reported an experience on reengineering project for migrating a Natural/Adabas bank system. This approach consists of code analysis, re-documentation and reimplementation by combining commercial and *ad hoc* tools. *Fujiwara et al.* [12] proposed a reverse engineering technique to generate C code from COBOL assembly programs. This technique is based on dataflow analysis, recognition of control structures, and code pattern matching.

Other proposals provided wrappers and other integration mechanisms, without modifying mainframe systems, to integrate them into Service-Oriented Architecture (SOA) environments. For example, *Koschel et al.* [14] provided wrappers for the integration of Natural systems into SOA environments. *Canfora et al.* [15] used a similar wrapping approach which focuses mainly on interactive functionality. *Rodriguez et al.* [8] analyzed the differences between bottom-up and top-down approaches concerning web service generation from COBOL systems.

Other kind of proposals focused on extracting business rules and re-documenting mainframe systems so that such information can be used in modernization projects. For instance, *Erlikh* [16] proposed *RescueWare*, a method and supporting tool that uses knowledge mining to extract reusable business or e-components from LIS based on the COBOL program call map. *Cosentino et al.* [17] also provided a business rule extraction tool for COBOL systems. Starting from a COBOL program, this technique derives a model-based representation of the source code and provides a set of model transformations to identify and visualize the embedded business rules. A supporting tool facilitates the definition of an application vocabulary and the identification of relevant business variables. Alternatively, *Dorninger et al.* [11] built an *ad hoc* tool prototype to generate COBOL documentation. This approach supports re-documentation in COBOL to Java migration projects and creates the foundations to continuously generate up-to-date documentation for the new software system.

B. Screen scraping

Screen scraping allows extracting the content from user interfaces. This technique is commonly used for obtaining data when there is no useful API [18]. The usual context of screen scraping is a program wanting to extract information from another program without any API or direct access to the data source.

Screen scraping techniques are used in a lot of domains, like for example in web crawling. *Mahto et al.* [19] used screen scraping for web page indexing in the context of web crawling. *Fernández-Villamor et al.* [20] provided a semantic scraping

model for web resources applying linked data to web page screen scraping.

In mainframe systems modernization, screen scraping has been used as a black-box approach since it only considers input and output without looking inside the mainframe components [5]. In fact, screen scraping is commonly used when the data source is too complex or the maintenance unviable, which is the case in many large and complex mainframe systems. For example, *Durand* [21] presented a case study describing the internationalization of a mainframe application without changing the mainframe code by using screen scraping techniques.

Recent applications of screen scraping demonstrated that it can be used to record developers' behavior as they work on software development tasks. Such behavioral data is then used to understand and improve software engineering practices [22].

IV. MAINFRAME MIGRATION PROJECT

HistAuskunft (*Historische Auskunft*) is a commercial project for a German insurance company, customer of *itestra GmbH*. The project name means historical information in German. The motivation for the replacement in *HistAuskunft* is strongly derived from the current operating costs and heterogeneity of the architecture and the systems around the LIS. The project includes the involvement of the management from the customer and *itestra GmbH*.

The idea generation of *HistAuskunft* began in 2014. This idea was followed by a proof of concept in 2015. After the implementation, user test and screen scraping of all relevant data, *HistAuskunft* is expected to go live in 2018.

The main goal is to replace a LIS consisting of z/OS and a hierarchical database with IMS. The target system is to be coded using JavaFX and a relational database, IBM DB2. The mainframe system managed historical information which is nowadays used in a static way, i.e., for querying historical data if required. These needs are related to the German data protection regulation (GoBD) [6], which specifies the boundaries on data modification, format change or the shape and content of the migration process documentation.

A. Initial challenges

The customer requests the recovery of the data from the mainframe and the preservation of screens that contain relevant information. This process is not a simple migration as there are several challenges:

- The hierarchical database contains encoded data that cannot be migrated one to one. The EBCDIC 273 encoding requires specific processing to obtain character encodings compatible with current technologies. EBCDIC 273 is a character encoding for Germany and Austria dated from 1986.
- Functional requirements existing at the time of the design and development of the legacy hierarchical data structure led to a complex system.

- Data modeling aspects such as primary or foreign keys are not evident. Relational database concepts were not considered in the initial design of the hierarchical database, as the paradigms were different at that time.
- The size and numerous maintenance tasks on the database and the LIS as a whole have increased the complexity during several decades of life.
- The vast amount of content acquired in the LIS over the years makes the validation of the replacement a challenge by itself. The content is stored in different formats and sometimes not stored but calculated by functions trapped inside the LIS.

B. Alternative solutions

Before the screen scraping solution was designed and developed, alternative solutions aimed at recovering the historical data in the LIS were preliminarily considered. Alternative solutions were discussed and assessed together with the company's customer from a technical point of view but also considering a cost-benefit analysis. The most remarkable solutions identified during the solution design are the following:

- Keep the mainframe LIS: discarded because of increasingly high infrastructure and operational costs.
- Use the LIS in a virtual machine: only allows the retirement of the obsolete hardware. The software license and operational costs do not improve.
- Reimplementation of the LIS to new hardware: this solution allows replacing the hardware and solves part of the problem. Nevertheless, the solution keeps notable software license, operation and maintenance costs. Furthermore, the system would require possibly expensive special components for a replica of the legacy infrastructure (e.g.: IMS /DLI). In addition, this solution requires very high implementation and test costs.
- Migrate by converting the data for a new system: data migrations are very expensive, especially when the contents are partly unknown and require many modifications. This is very often the case with mainframe-based LIS.
- Transfer the content to a neutral format like RDBMS or XML allowing manual search: this solution requires complete understanding of the legacy data and some data migration. Additionally, the users cannot navigate the application as usual and must be trained in how to read the data content. The calculations of the original application present another limitation for this approach.
- Static storage of all possibilities for information output: this solution covers the storage of all relevant dialogs for the business information. In this case the calculation logic of the application is no longer required. However, there is a huge amount of data to store and the connection between dialogs would be

lost. It is unclear how all relevant dialogs and business objects would be visited by the users in this approach. This solution would require appropriate tool support for each application contained in the LIS.

After considering the mentioned alternative solutions, screen scraping was chosen due to it being software-based rather than hardware-based, with optimum logic preservation and a manageable cost and time. The main benefits are the following: (i) all screens are preserved with the same information that they had in the original system; and (ii) no user training is required. This is because the new system has an identical user interface; (iii) core data is transformed into a relational schema that can be easily migrated to other systems in the future; (iv) low effort for implementation and testing. Business logic does not have to be re-implemented, not even fully understood; and (v) other LIS can be efficiently added reusing the same components.

V. SCREEN SCRAPING TECHNIQUE

Fig. 1 shows the high-level architectural view of the solution. The LIS is structured with different dialogs. Each dialog has its own template or way of showing data. Dialogs have a title or header field that allows recognizing dialogs to use the appropriate template.

Each instance of a given dialog has different dialog content and behavior depending on the insurance contracts stored in the LIS. The legacy system contains different business objects.

Fig. 1. Architectural view of the project

Fig. 2. Contents of any contract in different screens

These business objects are entities formed by different attributes shown in different places of the LIS (in this project, mainly insurance contracts).

The dialog templates, content, behavior and business objects need to be extracted from the LIS. The *Content Extractor* is the module whose purpose is to extract those elements and to store them into the relational database. This database is to be used by the *Screen Viewer*.

The *Content Extractor* is divided into four main submodules: *screen navigation engine*, *mainframe connector*, *screen fields extractor*, and *screen persistency agent*. These sub-modules are explained in detail in the next sections.

The *Screen Viewer* is the replacement of the LIS. The users interact with the *Screen Viewer* as they would with the mainframe system. The *Screen Viewer* connects to the database that contains the information extracted from the LIS by the *Content Extractor*. The *Screen Viewer* reuses the screen specification from the *Content Extractor*.

A. Screen navigation engine

The LIS contains thirteen relevant dialog types. Out of those thirteen, six contain the information that should be extracted and incorporated to the new system. The remaining dialogs provide functionality to navigate the system and access the information. Each contract has a certain number of screens, and each screen is related to a dialog type (see Fig. 2).

Some of these dialogs have different screens showing different information depending on the year. For example, the 'account summary' dialog has the summary for different years, so there would be at least one screen of this dialog for every year for that contract. Each dialog can have one or several pages. Thus, each screen would belong to a specific year and page tied to that dialog. The screens have a variable number of fields defined by the dialog specification.

The LIS supports many different input commands. Some commands give access to the information contained in the system. Those commands are especially important as they provide the means for the navigation and extraction of all the relevant information contained in the LIS. Fig. 3 illustrates part of the navigation model for one of the LIS subsystems.

Since some screens can only be accessed by following a specific path, a navigation strategy has to be established to ensure the exhaustive extraction of every screen and with a minimal computation time (i.e., avoiding visiting the same screen many times if not necessary). This navigation strategy was devised by the project team with the input of the experts in the usage of the particular LIS.

Additionally, the navigator engine iterates through all available business objects (i.e., all the contract numbers).

B. Mainframe connector

Fig. 4 shows a class diagram that depicts classes used to connect to the mainframe systems from the *Content Extractor*. Each parallelized *ExtractionTask* makes use of a *ContentProvider* that implements the specific ways to log in and send commands to the customer mainframe system.

Fig. 3. Example of screen navigation model in the LIS

Fig. 4. Connection to the 3270 terminal from the extractor

Fig. 5. Entity-relationship diagram for storing screens and fields

The name of the customer has been anonymized. This structure supports reusing modules adapting the system to the specifics of different applications.

The implementation of this model considers a virtual user representing a regular user that uses the 3270 terminal to input commands and receive content to allow the extraction of content from the content provider.

C. Screen fields extractor

Parsing the screens of a certain dialog type for the extraction requires defining the dialog structure. This definition involves specifying the fields with the following data: (i) name of the field depending on their label or perceived business role; (ii) the data type of each field including the length; and (iii) the position of the field on the screen. This is important, since the position is always the same in the mainframe screens.

This structure is modeled using a plain text template with the static characters that appear in every screen of that dialog. The template is complemented with a definition of the format and length of the fields with variable content. This template is then used by the screen parser.

A parser built for the pre-processing of the dialog structure provides a draft of the dialog template. The draft is then polished by experts. In this way, the definition of dialog types requires less time, also reducing the total effort required to write screen scrapers for the different dialog types.

There are two challenges that were considered in this module: screen definition for exceptional cases and extraction performance:

- **Edge cases and exceptions.** Although the extractor needs the definition of the dialogs to parse them correctly, there was no formal definition of them. Thereby, such specifications were rebuilt from analyzing the content in the screens. Many of the screens share similar structure but many dialogs have unused fields or fields that only have a value in some of the screens for that dialog. Those fields are hidden until they are discovered in a specific contract that has a value for them. After those fields are discovered, they are included in the dialog definition so that the *Content Extractor* can extract those contracts. Other exceptions are not in content, but in navigation functionality. Locating the special cases is the first step in analyzing and learning how to implement and extract those contracts.
- **Performance.** Most of the time spent during the extraction (> 90%) corresponds with the input and output exchanged with the mainframe. This input and output covers processing the input in the display terminal, the net latency between the terminal and the mainframe, the processing time in the mainframe and showing the output in the terminal. The rest of the time is spent in extraction tasks such as parsing the screens. The optimal extraction strategy focuses on reducing the calls to the mainframe and the screen navigation as those operations take much more time than analyzing already fetched screens. The navigation strategy is therefore refined in the aforementioned module. Furthermore, the extraction is parallelized using multithreaded programming. At the beginning of the project, a pilot study was conducted in order to figure out the maximum number of threads until performance is stabilized and even gets worse. This threshold was established between 8 and 12 threads.

D. Screen persistency agent

Fig. 5 shows the entity-relationship diagram that represents the storage of fields and screens in a relational database. This schema is designed to be as simple as possible, in order for it to be valid for all screens types and subsystems in *HistAuskunft*. The *screens* entity has *screenId* as key and contains information about the *entityId*, *dialog*, *subId*, *page* and *date* of extraction. The *fields* entity contains the *screenId* of the screen that contains that field, the *index* on the screen for that field and the *value* of the field. The index in *fields* refers to the position on the visual screen depending on the row and column of the field. The information persisted in this schema together with the screen specification (dialog templates) can be used to recreate screens in the target system.

The interaction of the extractor with the LIS must be logged to keep track of the exchanges with the original system. This kind of record is defined by ISO/IEC 27000:2014 as “evidence of the results achieved” [23]. These records are evidence of the activities and part of the documentation. These records can also be necessary to prove that the process complies with the laws and regulations. Records are also helpful to keep track of what has already been done and what is yet to be done.

E. Screen viewer

After all screens have been scraped, a new system in JavaFX is implemented to recreate old mainframe screens. The implementation of this new user interface is semi-automatic, making use of the parsed dialog templates (in a similar way as for screen extractor). Fig. 6 shows the *Darlehensanzeige* dialog in both the legacy mainframe (left) and the new system (right) as example. Fields that could show personal data have been blurred. Notably, both systems have almost the same look to ease the transition from one system to the other for the users.

There are a couple exceptions though, for example, the available commands hint is different. That is because along with the changes in the environment of the LIS through the years, some features are no longer relevant for the business of the customer. Those features were not included in the new system.

Modeling the structure of the dialogs allows the navigation and extraction of the different screens. The *Screen Viewer* reuses that structure and includes the data read from the database. This task also covers the logic behind the filtering and navigation through the system.

Examples of this navigation are the usage of commands like next/previous year, next/previous page or going from the main dialog to the account summary.

Apart from screen recreation, there are some improvements in the new system implying differences in the system behavior:

Subsystem integration improvement. The *Screen Viewer* is developed considering the different subsystems. As such, the navigation between subsystems is supported natively as an improvement compared to the LIS. As requested by the customer, the new system provides a homogeneous access control, helping users to switch between subsystems easily. After all, the subsystems show different, but related information about the contracts. This was possible thanks to the new technologies used in this replacement project (e.g.: WebSphere Liberty integration, a common relational database or the Spring framework, among others). The user access to the different parts is managed by providing different user roles, allowing for a unified login that also avoids repetitive or redundant actions for the users. Integrating all systems in the same application makes the user interaction easier and reduces time and overhead from the different logins and intermediate screens to access the contents of each subsystem.

Fig. 6. Mainframe screen (left) compared to the recreated screen in the new viewer system (right)

- **Error handling improvement.** The system must provide feedback to the user's interaction, even when the actions are not supported by the system. Any field that contains invalid input is marked in red to give more visual feedback to the user. The cursor is also automatically placed on that field to help the user amend the field value. Some examples of exceptional situations are: invalid input, last year issue when trying to go further in the past, no data for the contract number, among others. As a difference with the mainframe behavior, the errors shown by the new system have been abstracted and homogenized. Equivalent situations show the same error message. For example, trying to reach the next page after reaching the last page shows the same message in all dialogs.

VI. AUTOMATIC SCREEN VERIFICATION

The screen verification aims to find defects during the development stage. This verification runs in parallel with the development of the other modules. The findings help to bring a polished product to the production stage. If the defects had reached the final product, it would have been close to impossible to solve them as the mainframe would have already been retired. If those defects had gone unnoticed, they could have affected the business operations of the customer.

This verification solution contributes to the domain field of automatic verifications of software migrations. The reason is that this approach can be applied to the migration of other LIS from different domains.

The information contained in the LIS has to be extracted without errors. This information contains insurance contracts and personal information protected by data protection laws. Verification measures are needed to ensure the correct replacement of the LIS. Thereby, recall and precision should be ensured in the target system.

- Recall means that loss of data is not allowed, i.e., some missing screens for a certain contract is absolutely not acceptable. Neither is it allowed for a value to be missing in a field for a concrete screen.
- Precision means that no data is extracted incorrectly e.g., a field value in a certain screen of the new systems differs from the actual value in the legacy one.

Due to such needs, manual verification seemed at first the only possible verification solution to cover the recall and precision demands. Nevertheless, it is unviable and therefore was almost directly discarded. The number of already extracted screens is around 43,000,000 and increasing, with 20 to 70 field values per screen on average. Furthermore, there are multiple particularities and combinations of characteristics that make the number of representative screen types still too high for manual testing.

A. Automatic verification solution

The automatic verification of the extracted screens is based on a robot that automatically navigates in parallel over the

original screens in the mainframe system and the recreated screens in the new system after the data has been extracted. This approach is inspired by similar proposals that have been used for the verification of Web applications [24, 25].

The robot types the same commands in both systems and performs the same navigation between dialogs using the same values. The robot synchronizes both interactions waiting for the results in both systems. This is because the response times are not the same for both systems and they also depend on the network. In each synchronization step, the solution checks that both screens are the same and contain the same values. The robot then compares the output in both systems in order to detect inconsistencies.

The navigation for the verification has a different approach than the navigation for the extraction. A different approach prevents the verification tool from repeating potential mistakes from the screen extraction approach. Incorporating random input, the verification tool can discover missing parts if the navigation model for the extraction was incomplete. The random input also helped discover additional content that was not found during the initial design of the screen extraction strategy.

Fig. 7 represents the architecture of the tool that allows the automatic verification of the mainframe migration. The data extractor makes use of specialized *Scrapers* that can interact with the mainframe terminal (IBM3270) or the new *Screen Viewer* (*HistViewer*).

Some dialogs were intentionally recreated with some differences in order to improve the user interaction. The verification solution can be customized to take such differences into account in order to filter the false positives from the posterior analysis of the verification result. The verification robot also applies appropriate conversions or transformations to compare values and determine if the mainframe and the recreated screens are equivalent (see data comparer module in Fig. 7). These steps are necessary because even though the information content is the same, the format can be different.

Fig. 7. Architectural view of the supporting tool for the verification

The tool performs two levels of comparison for each visited screen. One strict check in which the screen content is compared as a whole, screen to screen. The other check compares the screen content field by field. Only performing the field by field check could lead to invalid results because of potential mistakes in the field definition. On the other hand, the whole screen to screen check would not provide enough information. The field by field check instead, gives more semantic and meaningful results i.e., identifying which fields did not match. If the screen to screen check reported errors and the field by field did not, the field definition used in the

extraction has to be corrected. If both the screen to screen check and the field by field check report errors, the field by field report will give the most valuable information with the field identifier, type and values in both systems. If no check reported errors, the screens would be completely equivalent.

B. Challenges specific to the verification

There are some verification-solution specific challenges that were addressed:

- With the new technologies and paradigms in the new system, the response times are much better than in the LIS. More importantly, the network latency is variable. This motivates the definition of a synchronization mechanism. The tool needed a specific module that allowed identifying if the screens compared were the expected comparable screens, or if one of the systems had not updated the data content yet. If the verification is too fast, the systems under test could become unresponsive. This could affect the quality of the results and is another justification for an explicit check of whether the received output from the systems under test is valid.
- As it also happens with the extraction of content for the replacement, the verification solution does not have a native entry point offered by the systems. The verification solution accesses the user interface and has as input means the mouse and keyboard like a normal user would have.
- Additional layers of abstraction might allow more generic verification; nevertheless, it is necessary to find a sensible ground considering the available resources and the complexity increase associated with abstraction.
- Each dialog has a different field structure. The number and location of input and output fields is specific to each dialog. The verification solution needs to know how to enter the input arguments in the correct fields. Without that information, the verification solution could get stuck or lead to unexpected screens after writing input into invalid fields.
- The new system and the LIS show some information in a different way. This is a design decision as the new *Screen Viewer* should be not only a replacement but an improvement of the previous system. Special comparison patterns for those specific fields were defined to avoid false positives of non-matching fields.

VII. PRELIMINARY RESULTS

This section provides some running data and some valuable information concerning the migration process in production.

A. Screen scraping results

This paper presents an approach that can be reused in the migration of different mainframe systems. So far, the migration has been completed for four subsystems contained in the LIS. TABLE I. shows preliminary results for these four subsystems, which provides:

- The total number of migrated contracts, for which several screens are available. The total number of contracts in bottom row of TABLE I. (highlighted with an asterisk) shows the total number of unique contracts as the same contract might appear in several subsystems.
- The total number of screens.
- The number of screens per contract on average.
- The total number of extracted field values.
- The number of fields per screen on average.
- The extraction time in hours. This time is approximated since some interruptions happened due to errors found in production. Special cases in production, not found in the development data, lead to fixes and re-extraction of some contracts or screens for a specific dialog type.
- The extraction speed measured as the number of processed contracts per hour.
- The extraction speed measured as the screens per hour extracted in production.

These results show that the different subsystems have very different amounts of contracts, screens and fields. As an example, there are many more screens and fields per screen in the Loan (4) subsystem than in the Pension (3) subsystem. This means that the structure varies greatly between subsystems. In addition, the navigation through the subsystems is also different. Each subsystem has its own dialog types which support different features such as navigation between years, pagination or nested screens. This fact leads to a different number of screens per contract on average. These values vary from 189 screens per contract in the case of Loan (subsystem 4) to 41 in the case of Payments (subsystem 2).

TABLE I. SCREEN MIGRATION RESULTS FOR THE FOUR SYSTEMS MIGRATED

Id	Subsystem	#contracts	#screens	#screens per contract	#fields	#fields per screen	Extraction time (hours)	Contracts per hour	Screens per hour
1	Life	300,640	22,467,776	74.7	1,590,872,685	70.8	1466.6	205.0	15319.1
2	Payments	467,419	19,258,055	41.2	417,777,482	21.7	1497.8	312.1	12857.1
3	Pension	7,350	383,680	52.2	7,598,258	19.8	35.4	207.7	10843.3
4	Loan	7,289	1,377,949	189.0	85,267,186	61.9	313.8	23.2	4390.2
Total		(*) 679,871	43,487,460	64.0	2,101,515,611	48.3	3313.7	205.2	13123.5

As for the extraction times, these results prove that a large volume of contracts with many screens and fields can be extracted in a manageable time assumable by *itestra GmbH*'s customer. Even then, the time spent on the four subsystems was also variable.

Furthermore, the end users already tested three systems migrated with this approach. The users also validated the obtained results. The legacy applications in the LIS will be hidden for the users as they already moved to the new system. These legacy applications will be switched off once the manual validation is completed by the company's customer.

B. Verification results

In order to compare automatic verification performance against a manual one, experts (in the usage of the first system to be migrated) manually verified one hundred screens by considering all the dialog types. The verification of this sample took slightly more than two hours. In contrast, automatic verification with the verification tool verified the same one hundred screens in less than five minutes. The comparison of performance did not account for the registry of non-matching fields which would accentuate the difference even more in favor of the automatic tool. The reasoning is that manual logging of found issues in the comparison takes much more time.

This means that the supporting tool reduced the required time by more than 96% of the time taken for a manual verification. Furthermore, the tool makes the verification task autonomous, meaning that human effort can be spent in other tasks.

Apart from the efficiency analysis, we think the effectiveness of the tool is clearly better. Manual verification, owing to its inherent nature, is more error-prone than the automatic one. In addition, mistakes can be unnoticed due to the expert's fatigue if manual verification is carried out for long time. Thereby, avoiding human errors is one of the most valuable contributions of the supporting tool. Even with *diff* tools some errors could be unnoticed and format differences between the old and the new system could raise false positives.

The automated test tool also serves as a stress test. With automated interaction with the system, it is possible to simulate continuous use of the system or fast interaction. This allows finding memory leaks or performance issues in the new system.

Finally, the verification solution found that the performance of specific calls to the new system was unsatisfactory. This motivated the review and modification of the queries used for the data access to improve performance in those cases.

VIII. CONCLUSIONS

This paper presented a solution based on a screen scraping technique for migrating mainframe systems toward JavaFX and a relational database. In contrast with white-box approaches, this solution ignores the internal structure of the COBOL programs and focuses on extracting the screen as is in the legacy systems to be later recreated in the target system.

Furthermore, new technologies (JavaFX and DB2) update the LIS providing better maintainability and efficiency. In the new system, current approaches provide the users with additional functionality that increases efficiency in their daily operations. An example for this claim is the unified login and access by user role. In contrast, the LIS used numerous individual login portals for each section, hampering user productivity.

In this kind of software migration, it is very important that the information contained in the new system is equivalent to the content of the LIS. Because of that, both the screen scraping and the automatic verification techniques were designed and developed for verifying that no missing or incorrect data has been migrated. This verification allowed the detection of specific unexpected behaviors of the *Screen Viewer* in the new system. These could probably go unnoticed in other tests and pass the validation tests.

This solution is being applied in a large-scale, real-life project for a German insurance company. So far, more than 40 million screens have been successfully extracted for four mainframe subsystems, and more millions of screens are currently being extracted for other two subsystems. Thereby, in our opinion, the most valuable contribution (apart from the developed solutions) is the application to a large-scale project in a real-life mainframe environment. This case study demonstrates the feasibility and applicability of the proposed black-box migration solution.

The main implication is that a mainframe migration problem is shifted to a screen scraping one. This makes it possible to deal with the problem from a black-box perspective, which entails several advantages:

- There is no need for managing and rewriting COBOL code. The risk of finding people highly skilled in this programming language is mitigated, and the associated cost is reduced. However, the experts who can operate the mainframe are still necessary.
- The business knowledge preservation is addressed from a data checking perspective (screen equivalence) rather than considering the internal structure with all business transactions and interactions. This reduces risks due to the absence of people in the company's customer who comprehend the business logic. Nevertheless, the participation of the end users during the building of the new system was crucial to ensure the end product satisfies the needs of the organization.
- A confident, fast, repeatable verification can be performed since the focus is on equivalence of extracted data between the legacy screens and the new, recreated ones. Automating the verification of the migration meant that the results were more reliable and human effort could be spent in other tasks.

Further implication concerns the benefits of using the verification solution. On the one hand, the verification raised confidence in the replacement of the system. In fact, the verification found issues that could be fixed before the productive extraction of information from the LIS. Specific

fields that were not extracted would have implied information lost in the LIS and not transferred to the new system. That is an example of issues prevented by the verification solution. Repeating productive extractions could incur increased costs in the form of extended extraction time close to a year. More extraction time also implies higher computational costs for the usage of the LIS. On the other hand, the randomness of the verification solution allowed finding screens and ways to show information in the LIS that were not considered in the extraction strategy. Those screens were missed as they were found only in specific contracts. The feedback from the verification tool allowed the extension of the extraction to include those missed screens before productive extraction. The verification solution also presented different output to the same input for both systems in some cases. This helped to show the customer special behaviors enabling asking whether it was a relevant requirement for the new system to also mimic those behaviors.

We have planned some future work framed in the context of the current research. First, there are additional legacy subsystems contained in *itestra GmbH's* customer's mainframe environment that require migration. Adapting the *Screen Fields Extractor* and then the *Screen Viewer* to support those subsystems is the end goal. From there, other systems for other customers with different technologies can be migrated with this technique. Another line requiring future work is the evolution of the verification solution to handle those newly migrated applications.

ACKNOWLEDGMENT

This work was primarily supported by *itestra GmbH*. The work was additionally supported by the financial backing which Dr. Ricardo Pérez-Castillo has received from the Regional Government of Castilla-La Mancha in their program for retaining and bringing talent back, carried out in accord with RIS3 objectives.

REFERENCES

- [1] Pérez-Castillo, R., B. Mas, and M. Pizka. Understanding legacy architecture patterns. in *Evaluation of Novel Approaches to Software Engineering (ENASE)*, 2015 International Conference on. 2015. IEEE.
- [2] Sommerville, I., *Software Engineering*. 9 th Edition ed. 2010: Pearson. 792.
- [3] Kirchmayr, W., M. Moser, L. Nocke, J. Pichler, and R. Tober. Integration of Static and Dynamic Code Analysis for Understanding Legacy Source Code. in *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 2016.
- [4] Mulcahy, J.J. and S. Huang. Autonomic Software Systems: Developing for Self-Managing Legacy Systems. in *2014 IEEE International Conference on Software Maintenance and Evolution*. 2014.
- [5] Comella-Dorda, S., K. Wallnau, R.C. Seacord, and J. Robert. A survey of black-box modernization approaches for information systems. in *Proceedings 2000 International Conference on Software Maintenance*. 2000.
- [6] Finanzen, B.d. Grundsätze zur ordnungsmäßigen Führung und Aufbewahrung von Büchern, Aufzeichnungen und Unterlagen in elektronischer Form sowie zum Datenzugriff (GoBD). 2014 14. November 2014 [cited 2018 22/03/2018]; Available from: http://www.bundesfinanzministerium.de/Content/DE/Downloads/BMF_Schreiben/Weitere_Steuertemen/Abgabenordnung/Datenzugriff_GoBD_U/2014-11-14-GoBD.pdf?__blob=publicationFile&v=3%C3%A7.
- [7] Verbraucherschutz, B.d.J.u.f. Abgabenordnung (AO) § 147 Ordnungsvorschriften für die Aufbewahrung von Unterlagen. 2014 [cited 2018 05/04/2018]; Available from: https://www.gesetze-im-internet.de/ao_1977/_147.html.
- [8] Rodriguez, J.M., M. Crasso, C. Mateos, A. Zunino, and M. Campo. Bottom-Up and Top-Down Cobol System Migration to Web Services. *IEEE Internet Computing*, 2013. 17(2): p. 44-51.
- [9] The Standish Group, *Modernization. Clearing a pathway to sucess.* 2010, The Standish Group International, Inc.
- [10] Sneed, H.M. and K. Erdoes. Migrating AS400-COBOL to Java: A Report from the Field. in *2013 17th European Conference on Software Maintenance and Reengineering*. 2013.
- [11] Dorminger, B., M. Moser, and J. Pichler. Multi-language re-documentation to support a COBOL to Java migration project. in *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 2017.
- [12] Fujiwara, D., N. Ishiura, R. Sakai, R. Aoki, and T. Ogawara. Reverse Engineering from Mainframe Assembly to C Codes in Legacy Migration. in *2016 5th IIAI International Congress on Advanced Applied Informatics (IIAI-AAI)*. 2016.
- [13] Brito, K.d.S., V.C. Garcia, and S.R.d.L. Meira. Experiences from a Brazilian Bank Reengineering Project. in *2010 14th European Conference on Software Maintenance and Reengineering*. 2010.
- [14] Koschel, A., C. Kleiner, and I. Astrova. Mainframe Application Modernization Based on Service-Oriented Architecture. in *2009 Computation World: Future Computing, Service Computation, Cognitive, Adaptive, Content, Patterns*. 2009.
- [15] Canfora, G., A.R. Fasolino, G. Frattolillo, and P. Tramontana. A wrapping approach for migrating legacy system interactive functionalities to Service Oriented Architectures. *J. Syst. Softw.*, 2008. 81(4): p. 463-480.
- [16] Erlikh, L., Leveraging legacy system dollars for e-business. *IT Professional*, 2000. 2(3): p. 17-23.
- [17] Cosentino, V., J. Cabot, P. Albert, P. Bauquel, and J. Perronnet. Extracting business rules from COBOL: A model-based tool. in *2013 20th Working Conference on Reverse Engineering (WCRE)*. 2013.
- [18] Newman, K. A plausible ancestor generator an experiment with heuristics, screen scraping and geolocs. in *2013 International Conference on Current Trends in Information Technology (CTIT)*. 2013.
- [19] Mahto, D.K. and L. Singh. A dive into Web Scraper world. in *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*. 2016.
- [20] Fernández-Villamor, J.I., J. Blasco-García, C.Á. Iglesias, and M. Garijo. A semantic scraping model for web resources: Applying linked data to web page screen scraping. in *ICAART 2011 - Proceedings of the 3rd International Conference on Agents and Artificial Intelligence*. 2011.
- [21] Durand, C., Internationalizing mainframe applications through screen scraping, in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. 2009. p. 228-235.
- [22] Bao, L., J. Li, Z. Xing, X. Wang, and B. Zhou. Reverse engineering time-series interaction data from screen-captured videos. in *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering, SANER 2015 - Proceedings*. 2015.
- [23] ISO/IEC, ISO/IEC 27000:2014. Information technology -- Security techniques -- Information security management systems -- Overview and vocabulary. <https://www.iso.org/standard/63411.html>. 2014, ISO/IEC.
- [24] Choudhary, S.R., H. Versee, and A. Orso. WEBDIFF: Automated identification of cross-browser issues in web applications. in *2010 IEEE International Conference on Software Maintenance*. 2010.
- [25] Taneja, K., N. Li, M.R. Marri, T. Xie, and N. Tillmann. MiTV: multiple-implementation testing of user-input validators for web applications, in *Proceedings of the IEEE/ACM international conference on Automated software engineering*. 2010, ACM: Antwerp, Belgium. p. 131-134.