A Hybrid Approach to Re-Host and Mix Transactional COBOL and Java Code in Java EE Web Applications using Open Source Software

Philipp Brune
Neu-Ulm University of Applied Science, Wileystraße 1, 89231 Neu-Ulm, Germany

Keywords: Web Services, Service Orientation, Transaction Processing, COBOL, Java EE, Open Source Software, Mainframe Computing.

Abstract: Despite the common notion of mainframe-based transactional COBOL applications being an outdated technology, in many companies they continue to serve as the IT backbone. Therefore, in the era of big data and cloud services, these applications need to be transformed towards open, service-oriented architectures to preserve their value. This challenge has been tackled by different strategies so far, ranging from adding web service layers to existing mainframe applications to various products providing emulation on non-mainframe platforms. In contrast, in this paper this transformation is considered not as a mere COBOL re-hosting issue, but from the perspective of integrating COBOL in Java EE-based web applications. An open framework is demonstrated for executing existing transactional COBOL programs as part of Java EE application servers. It is build on established Open Source Software (OSS) components and executes on any Un*x-like operating system, in particular also on the mainframe itself.

1 INTRODUCTION

Despite COBOL is considered an outdated language by many (Khadka et al., 2014), it still plays an important role in enterprise application development, in particular on the mainframe platform (Lämmel and De Schutter, 2005; Vinaja, 2014) (the term "mainframe platform" here refers to IBM's S/390 architecture and its descendants, as it is commonly understood). There exist multiple reasons why it is neither possible nor desirable to replace it completely (Suganuma et al., 2008; Sagers et al., 2013; Farmer, 2013; Kiefer, 2017).

Mainframe-based transactional COBOL programs typically handle the core, value-generating business processes of many companies and store the related data. As so-called "Systems of Record" they are typically hosted on-premise by many companies (Moore, 2011). In the era of big data and cloud-based services (Hashem et al., 2015), to preserve their value, the challenge frequently is to convert these mission-critical applications into open, service-oriented backends (Khadka et al., 2015). which could be more easily integrated e.g. in cloud-based "Systems of Engagement" like mobile apps and web frontends, or into distributed big data processing applications (Moore, 2011; Tommy et al., 2015).

While various approaches to tackle this challenge have been proposed and used in research and practice over the years, the re-engineering of existing, large-scale transactional COBOL applications remains an ongoing topic for most mainframe-centered IT organizations. Depending on the strategic role of the mainframe platform within an organization, this is usually either achieved by migrating and re-hosting these applications to other non-mainframe platforms (e.g. Linux) or by modernizing them on the mainframe platform, thereby typically making use of the power and unique features of modern mainframes (Sagers et al., 2013; Farmer, 2013; Vinaja, 2014).

In any case, transactional COBOL applications require a transaction processing monitor (TPM) middleware such as IBM's CICS[1] (Malaika and Park, 1994) or Fujitsu's openUTM[2] to run in, which therefore needs to be either present or emulated on any target platform used for hosting such applications. Therefore, previous approaches consider the problem mainly from the COBOL perspective, trying to provide a replacement or emulation for the traditional mainframe TPM software (Talati and Lackie, 1999; White, 2000).

[1] https://www.ibm.com/software/products/de/cics-tservers
[2] http://www.fujitsu.com/de/products/software/middleware/openseas-oracle/openutm/

Brune, P.
A Hybrid Approach to Re-Host and Mix Transactional COBOL and Java Code in Java EE Web Applications using Open Source Software.
DOI: 10.5220/0006943402390246
In *Proceedings of the 14th International Conference on Web Information Systems and Technologies (WEBIST 2018)*, pages 239-246
ISBN: 978-989-758-324-7

In contrast, in this paper the topic is looked at from the perspective of re-using transactional COBOL code as part of modern Java Enterprise Edition (EE) web applications. Therefore, in this paper an open framework build purely on established Open Source Software (OSS) is proposed and demonstrated, which allows to execute existing transactional COBOL programs as part of Java EE application servers. The proposed framework supports all Un*x-like or Linux operating systems, therefore also the mainframe itself. It is evaluated using a proof-of-concept implementation to demonstrate its feasibility.

Recently, Java EE has been handed over to the Eclipse Foundation to manage its future development, and therefore re-labeled as Jakarta EE[3]. However, for sake of simplicity in this paper still only the term Java EE is used to denote both Java EE and Jakarta EE.

The rest of this paper is organized as follows: In section 2 the related work is analyzed in detail, while section 3 describes the software architecture of the proposed solution. The latter is evaluated by a proof-of-concept (PoC) Java EE application presented in section 4 and tested in a demo experiment illustrated in section 5. We conclude with a summary of our findings.

2 RELATED WORK

The modernization of existing transactional COBOL-based applications by modularization and encapsulation (Sellink et al., 1999) over the years has been pursued using different strategies, and numerous approaches have been proposed, which can be grouped into three main categories:

- Modernize the existing applications on the mainframe platform (Sellink et al., 2002), typically by wrapping COBOL transaction programs by web service facades or behind modern Web user interfaces (Sneed, 2001; Lee et al., 2001) and integrating them in distributed service-oriented application architectures (Calladine, 2004; Ferguson and Stockton, 2005; Rodriguez et al., 2013; Mateos et al., 2017). This approach is typically adopted when the unique features of mainframes such as extremely high availability and outstanding transaction throughput are inevitably required for an application (Farmer, 2013; Vinaja, 2014), and well supported by various software tools from the mainframe vendors (like e.g. the current versions of their respective TPM products) (Bainbridge et al., 2001) as well as from multiple third-party vendors.
- Re-hosting and subsequent modernization of the existing COBOL applications on other non-mainframe platforms (including cloud services), mainly to reduce the perceived high operating costs of the mainframe platform (Lancia et al., 2007; Khadka et al., 2015). This typically requires either the original mainframe TPM middleware (Malaika and Park, 1994) to be available for these non-mainframe platforms (which is the case for the major mainframe TPM products) or a kind of third-party emulation technique to be used to mimic the functionality of the TPM (Talati and Lackie, 1999; Apte et al., 2017). The latter has been tried to achieve over the years by numerous attempts, ranging from vintage computing enthusiasts' hobby projects[4] to high-end commercial offerings[5]. However, some of these approaches are not feature-complete, lack development activity, or rely on proprietary technology. Also legal issues may affect their success, as many TPM-retated technologies have been patented over the decades (Talati and Lackie, 1999; White, 2000; Lymer et al., 2001; Apte et al., 2017).
- Extraction of the business rules and logic from the existing COBOL code (e.g. by using special analysis tools), and their subsequent re-implementation using other languages and platforms (Sneed, 1992; Huang et al., 1998; Sneed, 2001; Bodhuin et al., 2002; Lancia et al., 2007; Suganuma et al., 2008; Zhou et al., 2010; Mainetti et al., 2012; El Beggar et al., 2014). This approach is widely discussed in the scientific literature on legacy systems modernization, but may be too expensive or riskful for companies in many cases, since existing mainframe applications are typically highly critical (Suganuma et al., 2008).

In general, the mainframe platform offers various unique features like the support for high availability, vertical scalability and security (Vinaja, 2014). Therefore, a re-hosting or re-implementtai on other platforms is not be feasible or useful in all cases (Farmer, 2013; Vinaja, 2014). In addition, completely re-writing existing transactional COBOL applications using modern languages and platforms might be much too costly compared to modernizing them (Kanter and Muscarello, 2005).

In modern enterprise application development, the role of COBOL has been overtaken by Java to a large

[3]https://jakarta.ee/about/

[4]http://www.kicksfortso.com

[5]https://www.lzlabs.com/

extend, and Java Enterprise Edition (EE)[6] application servers provide functionalities similar to classical TPM middleware to the Enterprise Java Bean (EJB) components deployed in them (Bainbridge et al., 2001; Lancia et al., 2007). In particular, Java EE application servers support distributed transactions and the 2-phase-commit (2PC) protocol through the Java Transaction API (JTA)[7].

Since Java EE application servers are similar to classical TPM middleware to some extend, it could be an interesting strategy for moderninzing transactional COBOL appplications to make these programs run integrated in a Java EE application server as part of a JTA tranasction. In this scenario, most of the TPM functionality required to run these programs (such as transaction handling, resource access, access to message queues, user interfaces, etc.) wouldl be provided by the Java EE application server out of the box through the default Java EE features. First, a dedicated TPM middleware for executing transactional COBOL programs is not necessary in this approach, thereby avoiding any dependency on proprietary software components. Only a thin "glue component layer" is needed to manage the native COBOL execution for the Java EE application server. Second, since Java EE is portable and available on most platforms, in particular also on mainframes, this approach also supports mainframe-to.mainframe re-hosting, e.g. on Linux[8].

Since such an approach has not been discussed in the literature so far, in this paper the question is addressed how an open software architecture for a respective approach could look like, which uses only open-source software (OSS) components, and how feasible it is in practice.

3 DESIGN OF THE SOFTWARE ARCHITECTURE

In figure 1 an overview of the software architecture of the proposed approach is shown. We refer to it as the Quick Web-Based Interactive COBOL Service (QWICS). Its full source code described in the following is available as OSS on GitHub[9].

As the major components for the design of its software architecture, the following OSS solutions have been selected following a "best of breed" strategy:

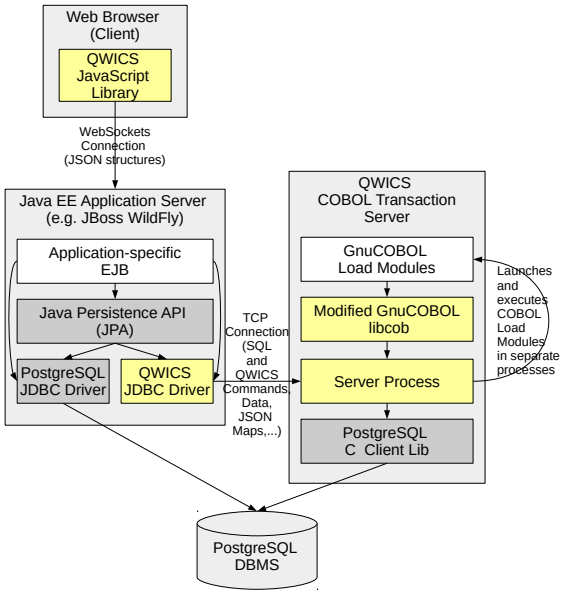

Figure 1: Overview of the software architecture of the proposed approach (called QWICS). The arrows denote usage/invocation relationships. Yellow boxes describe the QWICS-specific components and white boxes the application-specific COBOL or Java code. The integration between the Java and the COBOL code is achieved using a specific JDBC driver calling the COBOL server via its own protocol over a TCP connection.

- The *GnuCOBOL compiler*[10], since it is the most complete and mature OSS COBOL compiler available. In fact, it is a COBOL frontend to the GNU C compiler, translating COBOL to C first and then compiling the result using the standard gcc with its high optimization. Therefore, it supports the same target architectures as gcc, which indeed covers most existing platforms, including the mainframe.
- The *PostgreSQL relational database system*[11] as replacement for the mainframe databases used by typical transactional COBOL applications. PostgreSQL has been selected since it fully supports 2PC and distributed transactions (different from other OSS databases), which is necessary for enterprise applications. Also, its BSD license allows a completely free commercial use, which might be important for future applications in practice.
- The *JBoss WildFly application server*[12] as Java EE runtime, since it is probably the most established and mature OSS Java EE implementation with a long-time history, used in many critical enter-

[6] http://www.oracle.com/technetwork/java/javaee/overview/index.html
[7] http://www.oracle.com/technetwork/java/javaee/jta/index.html
[8] https://www.openmainframeproject.org/
[9] https://github.com/pbrune1973/qwics
[10] https://sourceforge.net/projects/open-cobol
[11] https://www.postgresql.org
[12] http://wildfly.org

```
int (*performEXEC)(char*, void*) = NULL;

void display_cobfield(cob_field *f, FILE *fp) {
    display_common(f,fp);
}


void
cob_display (const int to_stderr,
   const int newline, const int varcnt,
   ...)
{
        FILE            *fp;
        cob_field       *f;
        int             i;
        int             nlattr;
        cob_u32_t       disp_redirect;
        va_list         args;

// BEGIN OF EXEC HANDLER
        va_start (args, varcnt);
        f = va_arg (args, cob_field * );
        if (strstr((char*)f->data,
             "TPMI:")) {
           char *cmd
                = (char*)(f->data+5);
            if (varcnt > 1) {
                f = va_arg (args,
                      cob_field * );
            }
           (*performEXEC)(cmd,(void*)f);
           va_end (args);
           return;
         }
        va_end (args);
// END OF EXEC HANDLER
```

Figure 2: Necessary modification to `termio.c` of GnuCOBOL's `libcob` runtime library. Only the lines shown need to be added, no further modifications are necessary. This code adds an interception to `DISPLAY` statements of the form `DISPLAY "TPMI:...`, which are used to execute the `EXEC`-macros in the original COBOL source.

prise applications in practice.

For the integration of the transactional COBOL programs in the Java EE container, three main "glue components" have been designed illustrated in figure 1. These are kept as lightweight as possible, delegating most of the work (e.g. transaction handling, database connectivity, ...) to the existing and well-approved components listed above. This "glue layer" consists of:

- *COBOL Transaction Server*: A runtime container to dynamically load and execute the COBOL load modules (i.e. executable binaries in mainframe terminology) created by the GnuCOBOL compiler. It is written in C and for each client session executes the COBOL load modules in a separate thread. It is linked against a slightly modified version of GnuCOBOL's `libcob` library, which now intercepts COBOL `DISPLAY` statements of the form `DISPLAY "TPMI:...` and calls a function in the runtime container for these. The code fragment in figure 2 shows the respective necessary modification to libcob's `termio.c` source file. By means of this mechanism, the original `EXEC ... END-EXEC` macros in the code (Malaika and Park, 1994) are converted to `DISPLAY` statements by the preprocessor, and then interpreted by the runtime container. This mechanism provides maximum flexibility compared to the alternative solution of converting the macros to dedicated library calls, since e.g. the preprocessor needs not to be modified for supporting further macro statements.

  To execute the SQL in the `EXEC SQL` statements and the statements send directly by the client, the runtime container also manages pooled connections to the PostgreSQL database by means of PostgreSQL's C client library. The `EXEC` statements are either transmitted to the client (JDBC driver in the Java EE application server) or the database for interpretation. Only statements invoking other COBOL modules are directly executed by the transaction server.
- *JDBC Driver*: A Java Database Connectivity(JDBC)[13]-compliant driver supporting Non-XA and XA datasources to handle distributed transactions. It connects to the COBOL Transaction Server via a TCP connection, acting as a client to the COBOL programs. The execution of COBOL programs is controlled by the application server (e.g. from an EJB) via the JDBC driver by means of special callable statements and result sets.
- *JavaScript Library*: Implements an (optional) simple web user interface based on the original TPM's map defintions (Malaika and Park, 1994), which are converted beforehand to JavaScript Object Notation (JSON) structures by the preprocessor (see below). It uses the popular Bootstrap.js framework[14] for providing a responsive and modern look-and-feel suitable for mobiles, and WebSockets to communicate in a stateful manner with the application server. A stateful session here is required to mimic the behaviour of the original text-based terminal sessions.

In addition, a Java EE application (e.g. consisting of EJB, servlets, JPA, ...) is needed to actually use

[13]http://www.oracle.com/technetwork/java/javase/jdbc/ index.html

[14]https://getbootstrap.com/

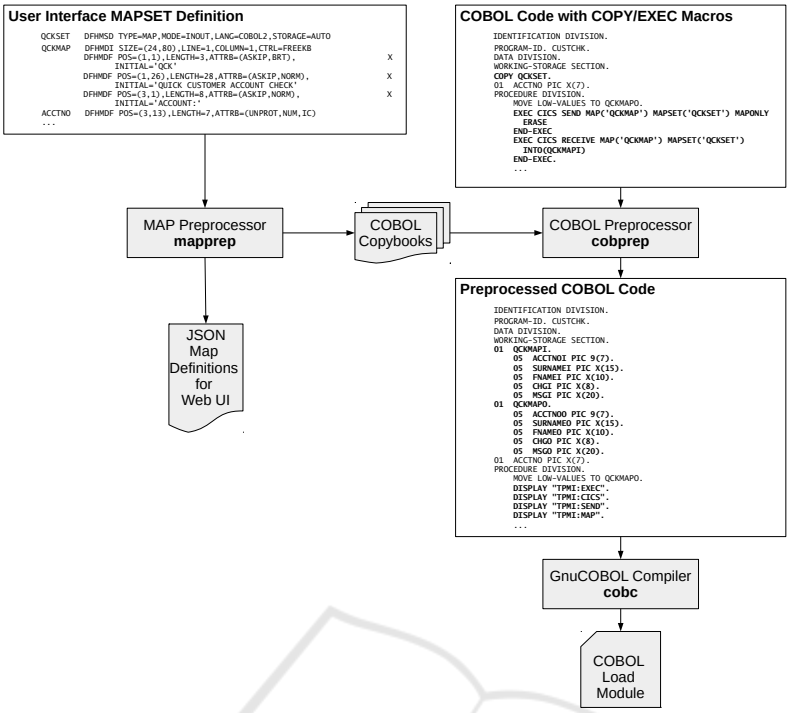

Figure 3: Process of preprocessing the original COBOL source files and map definitions for use by QWICS.

and invoke the COBOL programs within (distributed) transactions via the JDBC driver. Depending on the customer requirements, these applications can be of different kind, e.g. implementing web services to be called by other applications or full web frontends (e.g. using the above JavaScript library).

Transactional COBOL programs in traditional mainframe environments are loaded and executed by a TPM. All input/output (I/O) operations performed by these programs need to be handled by the TPM, so it can always keep track of them to handle the (distributed) transactions. Therefore, all necessary I/O operations (like e.g. executing SQL statements, sending or receiving data from the screen, etc.) are embedded in the COBOL code (or that of any other supported host language) in terms of TPM- and SQL-specific `EXEC ... END-EXEC` macros. These macros are preprocessed and translated to TPM API calls before the COBOL program is passed to the COBOL compiler (Malaika and Park, 1994). In addition, the terminal UI screen definitions (so-called maps) references by these macros are translated beforehand to COBOL variable declarations (so-called copybooks), which need to be copied into the COBOL code before compilation during this preprocessing stage as well.

Therefore, to be able to re-use the unmodified COBOL source codes including these macros, the presented approach needs to implement an analogous preprocessing step. The purpose of this COBOL preprocessing first is to convert the `EXEC` macro statements into real COBOL code (in the proposed approach, i.e. to the special `DISPLAY "TPMI:...` statements described above), and second to insert the necessary variable declarations from the respective COBOL copybooks before compilation.

The overall process is illustrated in figure 3: Two preprocessors have been implemented for that purpose, `cobprep` for COBOL and `mapprep` for the map definitions. Both are written in the C programming language. First, the original map definitions are preprocessed, generating the corresponding COBOL copybooks as well as a JSON representation of the maps, later usable for web frontends. Currently, the preprocessor only supports `EXEC SQL` and `EXEC CICS` macros, but it easily could be adapted to support another TPM's command syntax. Second, the COBOL code is preprocessed, including the previously generated copybooks. The resulting COBOL code then is passed through the GnuCOBOL compiler to obtain the final executable load modules.

4 PROOF-OF-CONCEPT JAVA APPLICATION

To test and demonstrate the functionality and interplay of the described components, a Proof-of-

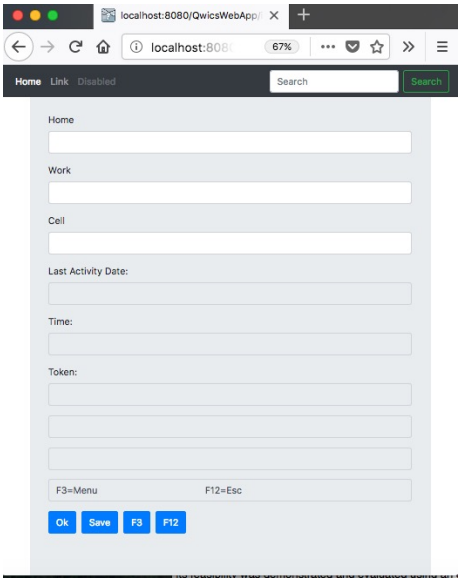

Figure 4: Screenshot of a map screen converted to JSON and displayed in web page by the JavaScript library.

Concept (PoC) Java EE application was implemented and deployed in the JBoss WildFly application server.

It consists of a single stateful session enterprise Java bean (EJB), accessing the COBOL code using a Java Transaction API (JTA) XA datasource (to support distributed transactions) and the JDBC driver described above. The stateful session bean acts as websocket backend invoked from the JavaScript library in the web browser. A stateful session bean is required, since websockets provide a session-based connection between the browser and the web server. This is necessary to make the UI behave similar to the traditional terminal screens. The EJB uses bean-managed transctions, since websockets require a separate EJB method call for each new message, corresponding to UI interactions here. Since transactions in a TPM may span multiple screen interactions, this needs to be recovered here using transactions spanning multiple method calls of an EJB. Therefore, JTA's UserTransaction interface is used to control the begin and end of transactions by the EJB.

5 EXPERIMENTAL EVALUATION

To evaluate the feasibilty of the proposed approach, in a first experiment an existing transactional COBOL application has been ported to the QWICS environment. To avoid any bias, a third-party application was required, written for a real mainframe TPM by developers not related or known to the authors. Therefore, a demo COBOL CICS application written by a consulting company for training purposes and being available online was selected (SimoTime Technologies and Services,).

Of the demo programs provided there (SimoTime Technologies and Services,), the COBOL programs MN1APP, MN1SQL and CQ2UPD and their respective map definitions were used for the experiment, since they allow to test UI menu navigation, program calls and SQL database access.

First, the COBOL source codes and the respective map definitions were processed by the preprocessor programs,. Afterwards, the COBOL code was compiled using GnuCOBOL.To make the code run properly, the embedded SQL statements in the original COBOL code had to be slightly adjusted to conform to the PostgreSQL SQL syntax. Beside this, the code could be used without further modifications.

The experiment was carried out two times with identical results, first by running it completely on an Apple MacBook Air developer laptop using MacOS X 10.11.6, thus, a BSD Un*x- derivative as operating system, and second on a IBM zBC12 mainframe computer running Linux for z Systems to demonstrate the potential of the approach for a true mainframe-to-mainframe migration and modernization of existing transactional COBOL programs. In both cases, all components as well as the preprocessing were running on the respective target platforms.

For the experiment, the COBOL programs were just used keeping the original transactional structure and with the UI maps just converted 1:1 automatically to a web UI using the JavaScript library. This was straightforward and the resulting application works correct and is usable. Figure 4 shows a screenshot of the converted UI map of the program CU2UPD in a web browser, as it is rendered by the JavaScript library. Due to the responsiveness of the underlying Bootstrap.js framework, the UI could be also used on a mobile device.

However, in a real-world scenario one probably would adopt a more sophisticated re-hosting approach, e.g. by transforming the UI to a modern, real web-like GUI or web service API manually or by implementing additional business functionality in Java and mixing it with the legacy COBOL code or replacing parts of it. But since all this is provided by the Java EE application server, it will be feasible without using further non-standard components.

While other existing approaches for re-hosting and modernizing transactional COBOL applications focus mainly on moving mainframe workloads to other (commodity) platforms by achieving full binary[15] or at least source-level compatibilty (Talati and

[15]https://www.lzlabs.com/

Lackie, 1999; Apte et al., 2017), this is not the case in the present approach. It does not intend to provide a full emulation of mainframe TPM environments and requires a partial adaption and recompilation of the existing sources. Instead, the focus here lies on integrating existing COBOL code into Java EE applications and modern web technologies.

The performance and scalability of the proposed approach still need to be evaluated. The runtime performance of the load modules generated by the GnuCOBOL compiler and the PostgreSQL database compared to their original mainframe counterparts has not ben investigated so far, but it presumably will be lower. Therefore, the presented approach is probably best suited for smaller and less critical applications.

Nevertheless, further research is needed to actually evaluate all this in a real-world case study.

6 CONCLUSION

In conclusion, in this paper an open approach was presented to modernize and re-host existing transactional COBOL applications within the context of Java EE application servers using solely Open Source Software (OSS) components. In addition, the appproach allows also to onvert the existing terminal-based user-interface maps into modern web UI and to mix COBOL and Java code while extending the application.

Its feasibility was demonstrated and evaluated using an existing third-party COBOL demo application originally written for the IBM CICS application server. However, the described approach should work for similar transaction processing monitors with some adaptations as well.

The OSS-only approach allows to host the applications on any common Un*x-like platform, including Linux on the mainframe. Therefore, a mainframe-to-mainframe migration of existing COBOL applications is possible, preserving the unique features of the mainframe platform.

While the feasibility of the approach needs to be further evaluated and demonstrated in practice, the presented first evalution indicates its potential. Further research is needed to empirically evaluate the approach and to bring the described proof-of-concept software artifacts to production quality.

REFERENCES

Apte, A., Negi, A., Rao, V., Pundeer, A., Kulkarni, S., Ladha, P., Moghe, S., Rallabandi, V., Shankar, R., Dhal, L., et al. (2017). Method and apparatus for migration of application source code. US Patent App. 15/397,473.

Bainbridge, A., Colgrave, J., Colyer, A., and Normington, G. (2001). CICS and Enterprise JavaBeans. *IBM Systems Journal*, 40(1):46–67.

Bodhuin, T., Guardabascio, E., and Tortorella, M. (2002). Migrating COBOL systems to the web by using the mvc design pattern. In *Reverse Engineering, 2002. Proceedings. Ninth Working Conference on*, pages 329–338. IEEE.

Calladine, J. (2004). Giving legs to the legacyweb services integration within the enterprise. *BT Technology Journal*, 22(1):87–98.

El Beggar, O., Bousetta, B., and Gadi, T. (2014). Getting objects methods and interactions by extracting business rules from legacy systems. *Journal of Systems Integration*, 5(3):32.

Farmer, E. (2013). The reality of rehosting: Understanding the value of your mainframe.

Ferguson, D. F. and Stockton, M. L. (2005). Service-oriented architecture: Programming model and product architecture. *IBM Systems Journal*, 44(4):753–780.

Hashem, I. A. T., Yaqoob, I., Anuar, N. B., Mokhtar, S., Gani, A., and Khan, S. U. (2015). The rise of big data on cloud computing: Review and open research issues. *Information Systems*, 47:98–115.

Huang, H., Tsai, W.-T., Bhattacharya, S., Chen, X., Wang, Y., and Sun, J. (1998). Business rule extraction techniques for COBOL programs. *Journal of Software: Evolution and Process*, 10(1):3–35.

Kanter, H. A. and Muscarello, T. J. (2005). Reuse versus rewrite: An empirical study of alternative software development methods for web-enabling mission-critical COBOL/CICS legacy applications. *CICS Legacy Applications: Fujitsu Software*.

Khadka, R., Batlajery, B. V., Saeidi, A. M., Jansen, S., and Hage, J. (2014). How do professionals perceive legacy systems and software modernization? In *Proceedings of the 36th International Conference on Software Engineering*, pages 36–47. ACM.

Khadka, R., Shrestha, P., Klein, B., Saeidi, A., Hage, J., Jansen, S., van Dis, E., and Bruntink, M. (2015). Does software modernization deliver what it aimed for? a post modernization analysis of five software modernization case studies. In *Software Maintenance and Evolution (ICSME), 2015 IEEE International Conference on*, pages 477–486. IEEE.

Kiefer, C. (2017). COBOL as a modern language. https://digitalcommons.northgeorgia.edu/honors_theses/17/. Accessed: 2018-07-27.

Lämmel, R. and De Schutter, K. (2005). What does aspect-oriented programming mean to COBOL? In *Proceedings of the 4th international conference on Aspect-oriented software development*, pages 99–110. ACM.

Lancia, M., Puccinelli, R., and Lombardi, F. (2007). Feasibility and benefits of migrating towards JEE: a real life case. In *Proceedings of the 5th international sym-*

posium on Principles and practice of programming in Java, pages 13–20. ACM.

Lee, M.-S., Shin, S.-G., and Yang, Y.-J. (2001). The design and implementation of Enterprise JavaBean (EJB) wrapper for legacy system. In *Systems, Man, and Cybernetics, 2001 IEEE International Conference on*, volume 3, pages 1988–1992. IEEE.

Lymer, S. F., Starkey, M., and Stephenson, J. W. (2001). System for automated interface generation for computer programs operating in different environments. US Patent 6,230,117.

Mainetti, L., Paiano, R., and Pandurino, A. (2012). Migros: a model-driven transformation approach of the user experience of legacy applications. In *International Conference on Web Engineering*, pages 490–493. Springer.

Malaika, S. and Park, H. (1994). A tale of a transaction monitor. *IEEE Data Eng. Bull.*, 17(1):3–9.

Mateos, C., Zunino, A., Misra, S., Anabalon, D., and Flores, A. (2017). *Migration from COBOL to SOA: Measuring the Impact on Web Services Interfaces Complexity*, pages 266–279. Springer International Publishing, Cham.

Moore, G. (2011). Systems of engagement and the future of enterprise IT: A sea change in enterprise IT. AIIM whitepaper.

Rodriguez, J. M., Crasso, M., Mateos, C., Zunino, A., and Campo, M. (2013). Bottom-up and top-down cobol system migration to web services. *IEEE Internet Computing*, 17(2):44–51.

Sagers, G., Ball, K., Hosack, B., Twitchell, D., and Wallace, D. (2013). The mainframe is dead. long live the mainframe! *AIS Transactions on Enterprise Systems*, 4:4–10.

Sellink, A., Sneed, H., and Verhoef, C. (1999). Restructuring of COBOL/CICS legacy systems. In *Software Maintenance and Reengineering, 1999. Proceedings of the Third European Conference on*, pages 72–82. IEEE.

Sellink, A., Sneed, H., and Verhoef, C. (2002). Restructuring of COBOL/CICS legacy systems. *Science of Computer Programming*, 45(2-3):193–243.

SimoTime Technologies and Services. The CICS connection, sample programs for CICS. http://www.simotime.com/indexcic.htm. Accessed: 2018-02-21.

Sneed, H. M. (1992). Migration of procedurally oriented cobol programs in an object-oriented architecture. In *Software Maintenance, 1992. Proceerdings., Conference on*, pages 105–116. IEEE.

Sneed, H. M. (2001). Wrapping legacy COBOL programs behind an XML-interface. In *Reverse Engineering, 2001. Proceedings. Eighth Working Conference on*, pages 189–197. IEEE.

Suganuma, T., Yasue, T., Onodera, T., and Nakatani, T. (2008). Performance pitfalls in large-scale java applications translated from COBOL. In *Companion to the 23rd ACM SIGPLAN conference on Object-oriented programming systems languages and applications*, pages 685–696. ACM.

Talati, K. and Lackie, C. W. (1999). Virtual software machine for enabling CICS application software to run on unix based computer systems. US Patent 6,006,277.

Tommy, R., Ravi, U., Mohan, D., Luke, J., Krishna, A. S., and Subramaniam, G. (2015). Internet of things (IoT) expanding the horizons of mainframes. In *IT Convergence and Security (ICITCS), 2015 5th International Conference on*, pages 1–4. IEEE.

Vinaja, R. (2014). 50 th aniversary of the mainframe computer: a reflective analysis. *Journal of Computing Sciences in Colleges*, 30(2):116–124.

White, J. W. (2000). Portable and dynamic distributed transaction management method. US Patent 6,115,710.

Zhou, N., Zhang, L.-J., Chee, Y.-M., and Chen, L. (2010). Legacy asset analysis and integration in model-driven SOA solution. In *Services Computing (SCC), 2010 IEEE International Conference on*, pages 554–561. IEEE.