An Open Source Approach for Modernizing Message-Processing and Transactional COBOL Applications by Integration in Java EE Application Servers

Philipp Brune[1,2](✉)

[1] Neu-Ulm University of Applied Science, Wileystraße 1, 89231 Neu-Ulm, Germany
Philipp.Brune@hs-neu-ulm.de
[2] QWICS Enterprise Systems, Taunustor 1, 60310 Frankfurt, Germany
Philipp.Brune@qwics.de
https://qwics.de,
https://qwics.org

Abstract. Modernization of monolithic "legacy" mainframe COBOL applications to enable them for modern service- and cloud-centric environments is one of the ongoing challenges in the context of digital transformation for many organizations. This challenge has been addressed for many years by different approaches. However, the possibility of using a pure Open Source Software (OSS)-based approach to run existing transactional COBOL code as part of Java EE-based web applications has just recently been demonstrated by the author. Therefore, in this paper, an overview of the previously proposed Quick Web-Based Interactive COBOL Service (QWICS) is given and its new extension to run message-processing COBOL applications via JMS is described. QWICS runs on Un*x-like operating systems such as Linux, and therefore on most platforms, but in particular on the mainframe itself. This enables a mainframe-to-mainframe re-hosting, preserving the unique features of the mainframe platform like superior availability and security.

Keywords: Web services · Message processing · Transaction processing COBOL · Java EE · Open Source Software · Mainframe computing

1 Introduction

In the recent and ongoing discussions about digital transformation and its impact for companies, in particular in the banking and financial service industries, it has been recently pointed out by various authors that the mainframe is by no means an outdated technology and probably will remain an important part of the entrprise IT landscape for a long time [16,28,42].

M. J. Escalona et al. (Eds.): WEBIST 2018, LNBIP 372, pp. 244–261, 2019.
https://doi.org/10.1007/978-3-030-35330-8_12

However, the traditional monolithic "legacy" COBOL enterprise applications are too inflexible and not open enough to fulfill the requirements of the digital age [1,38]. And despite its age, COBOL still plays a major role in enterprise application development on the mainframe [1,20,40] (with "mainframe" in this paper denoting IBM's S/390 platform and its descendants) and will continue to do so for a long time due to various reasons [1,9,18,30,36].

Therefore, the challenge of modernizing the existing COBOL applications is to convert them into service-oriented backends with well-defined APIs and using open technologies [17,19] while preserving their inherent value, making them accessible for cloud-based "Systems of Engagement" like mobile apps and web frontends [12] or distributed big data processing applications [27,39].

Migrating to Open Source Software (OSS) has recently been suggested as one approach for the banking industry to reach this goal [11]. Therefore, the recently proposed Quick Web-Based Interactive COBOL Service (QWICS) [6] discussed in this paper follows this direction. It is built on top of well-established, enterprise-ready OSS components such as e.g. the PostgreSQL relational database[1] [15].

Most of these legacy mainframe COBOL programs are online transaction-processing (OLTP) applications, which require a so-called transaction processing monitor (TPM) middleware to run in, like e.g. IBM's CICS[2] [25] or Fujitsu's openUTM[3]. Therefore, any modernization approach needs to take into account the inherent dependency of the transactional COBOL code on these TPM environments [6].

Most previous approaches [6] thus focus either on adding modern features such as RESTful APIs or Java EE support to the mainframe TPM middleware itselve, thus making use of the unique features of modern mainframes [9,30,40], or on providing a complete (mostly proprietary) replacement or emulation for the traditional mainframe TPM on other, non-mainframe platforms[4] [37,41].

In contrast, QWICS adopts another perspective on modernizing transactional COBOL applications, as it provides an open framework to run COBOL code in the context of any modern Java Enterprise Edition (EE) application server[5], using the latter to provide the TPM functionality [6]. QWICS is built using established OSS components and runs on Un*x and Linux derivates, in particular on the mainframe itself. Therefore, it combines the unique features of the modern mainframe with the platform independence and openess of Linux and OSS [6].

To further extend the capabilities of QWICS, in this paper its extension to support transactional message-processing COBOL programs is described. This is an important feature, as asynchronous, message-oriented data processing is

[1] https://www.postgresql.org.

[2] https://www.ibm.com/software/products/de/cics-tservers.

[3] http://www.fujitsu.com/de/products/software/middleware/openseas-oracle/openutm/.

[4] See e.g. https://www.lzlabs.com/.

[5] In this paper, for convenience still the term Java EE is used, despite it has been officially re-labeled recently to Jakarta EE (see https://jakarta.ee/about/).

an important concept for many enterprise applications [4]. Consistent with the original idea of QWICS, this is achieved by using the Java Message Service (JMS)[6] functionality of the Java EE application server.

The rest of this paper is organized as follows: In Sect. 2 the related work is analyzed in detail, while the software architecture of QWICS proposed in [6] is summarized in Sect. 3. Section 4 describes its new extension to support message-oriented transaction processing in COBOL using JMS. The experimental evaluation of QWICS and its results are illustrated in Sect. 5. We conclude with a summary of our findings.

2 Related Work

As described in [6], over the years the challenge of modernizing transactional "legacy" COBOL applications by modularization and encapsulation [31] has been addressed by numerous approaches, which my be classified into three major categories:

- *Modernization on the mainframe itself* [32]*:* Making use of new technologies such as web services and Java EE supported by the current versions of the "classical" mainframe TPM products to wrap COBOL transaction programs by web service facades or web user interfaces [19,22,35] and integrate them into service-oriented architectures [7,10,26,29]. This is well supported by various software tools from the mainframe vendors [3] as well as by third parties, and allows to make use of the unique features of modern mainframes such as extremely high availability and outstanding transaction throughput [9,40].
- *Migration to non-mainframe platforms (including cloud services)* [17,21]*:* Usually driven by the expectation to reduce the perceived high operating costs of the mainframe platform and the related vendor lock-in [17,21], this typically requires to use either the original mainframe TPM middleware [25] on these non-mainframe platforms (which is the case for the major mainframe TPM products) or an emulation mimicking the functionality of the TPM [2,37]. In particular, the latter has been addressed over the years by various hobbyist approaches[7] as well as professional commercial offerings (See footnote 4). However, these approaches have different drawbacks, since they frequently do not achieve the same transactional throughput as the original mainframes, rely on on proprietary emulation techniques (thereby creating a new vendor lock-in) or may suffer from patent-related and licensing issues [2,23,37,41].
- *Conversion of the program code to other languages and platforms:* This involves either the (automatic) conversion of the COBOL code to other, more modern programming languages and/or the extraction of the business rules and logic from the existing code (e.g. by using special analysis tools) and their subsequent re-implementation (either manually or by code generators)

[6] https://javaee.github.io/jms-spec/.
[7] http://www.kicksfortso.com.

using other languages and platforms [5,8,13,21,24,34–36,43]. Being closely related to model-driven development, this approach has gained wide interest in the scientific community on legacy systems modernization, but has been rarely used in practice since it may be too expensive or riskful for companies in many cases [36].

Despite numerous attempts for re-writing [14] or re-hosting mainframe "legacy" applications on other platforms, mainframe-based organizations after various failed migration projects [1,9,40] realized that the mainframe platform offers unique features like the support for high availability, vertical scalability and security [40], which could not always be recovered on other platforms [28,42]. Therefore, to enable the digital transformation [38], in the last years the focus shifted again to the first approach, namely the modernization on the mainframe itself [42].

Since Java has overtaken the role of COBOL in enterprise application development to a large extend, Java Enterprise Edition (EE)[8] application servers for the Enterprise Java Bean (EJB) components provide functionalities similar to those classical TPM middleware does for COBOL [3,21]. These includes support for distributed transactions and the 2-phase-commit (2PC) protocol through the Java Transaction API (JTA)[9] as well as transactional message processing via Message-driven Beans (MDB) and JMS.

Due to this analogy between Java EE application servers and classical TPM middleware, the previously proposed QWICS framework [6] demonstrated an approach to execute transactional COBOL programs integrated in a Java EE application server as part of a JTA transaction, such that the required TPM functions (such as transaction handling, resource access, access to message queues, user interfaces, etc.) are realized by the Java EE application server out of the box.

QWICS has been implemented as pure OSS, built using mature, enterprise-ready OSS components, since openess and OSS have been identified recently as cornerstones for enterprise application modernization in the age of digital transformation [11,15]. Since QWICS adds only a thin "glue component layer" to the established OSS components to manage the native COBOL execution for the Java EE application server [6], it is fully portable among various Un*x- and Linux derivatives, with a strong focus on "re-hosting" COBOL applications to Linux on the mainframe itself using OSS[10] [6].

While the feasibility of this approach has been demonstrated already in [6], QWICS so far lacked the support for transactional message-processing. Since this is an important feature [4], in this paper together with a summary of its previously described architecture and functionality the question is addressed, how an extension of QWICS to support message processing in COBOL via JMS could be designed and implemented.

[8] http://www.oracle.com/technetwork/java/javaee/overview/index.html.
[9] http://www.oracle.com/technetwork/java/javaee/jta/index.html.
[10] https://www.openmainframeproject.org/.

3 Design of the Software Architecture

Figure 1 shows an overview of the software architecture of QWICS, which has been presented in [6] and will be summarized in this section. Its full source code is available as OSS on GitHub[11].

As described in [6], QWICS is based on well-established, mature and enterprise-ready OSS components selected following a "best of breed" strategy: *GnuCOBOL*[12] for compiling the COBOL sources on the respective target platform, the *PostgreSQL relational database system* (See footnote 1) replacing the original mainframe DBMS and the *JBoss WildFly application server*[13] as the Java EE runtime. Here, in particular PostgreSQL has a wide recognition by practitioners as a reliable, scalable enterprise-quality OSS database management system [15].

To execute the transactional COBOL programs in the context of the Java EE container, as the core concept the integration by a special Java Database Connectivity(JDBC)[14]-compliant driver is used. This QWICS JDBC driver provides Non-XA and XA datasources to handle distributed transactions. It acts as a TCP client for the *COBOL Transaction Server*, which actually loads and executes the COBOL binary programs, the so-called load modules. Thus, the execution of these COBOL programs is invoked and controlled from the Java code (e.g. from an EJB) by special callable statements and result sets implemented by the QWICS JDBC driver. Thus, in QWICS always a Java EE application (e.g. consisting of EJB, servlets, JPA, Web Services, ...) is needed to call the COBOL programs via the JDBC driver [6].

The *COBOL Transaction Server* itself is a separate program implemented in C, which dynamically loads and executes the COBOL load modules created by the GnuCOBOL compiler. It also serves as a client for the PostgreSQL database to execute SQL statements embedded in the COBOL programs as well as those directly send from the Java side via the JDBC driver, being able to mix both in one transaction. The transaction server uses a slightly modified version of GnuCOBOL's `libcob` library, which has an intercept added to handle special `DISPLAY` statements of the form `DISPLAY "TPMI:...` and call a function in the transaction server for these. The necessary modification is shown in Fig. 2. This mechanism allows to handle the original `EXEC ... END-EXEC` macros in the code [25], which therefore have to be converted to these special `DISPLAY` statements in a subsequent preprocessing step [6].

To keep track of the state of all running transactions, a TPM needs to handle all input/output (I/O) operations performed by the programs it executes. Therefore, transactional COBOL programs running inside a TPM may not use the normal COBOL I/O statements. Instead, all necessary I/O operations (like e.g. executing SQL statements, sending or receiving data from the screen, etc.)

[11] https://github.com/pbrune1973/qwics.
[12] https://sourceforge.net/projects/open-cobol.
[13] http://wildfly.org.
[14] http://www.oracle.com/technetwork/java/javase/jdbc/index.html.

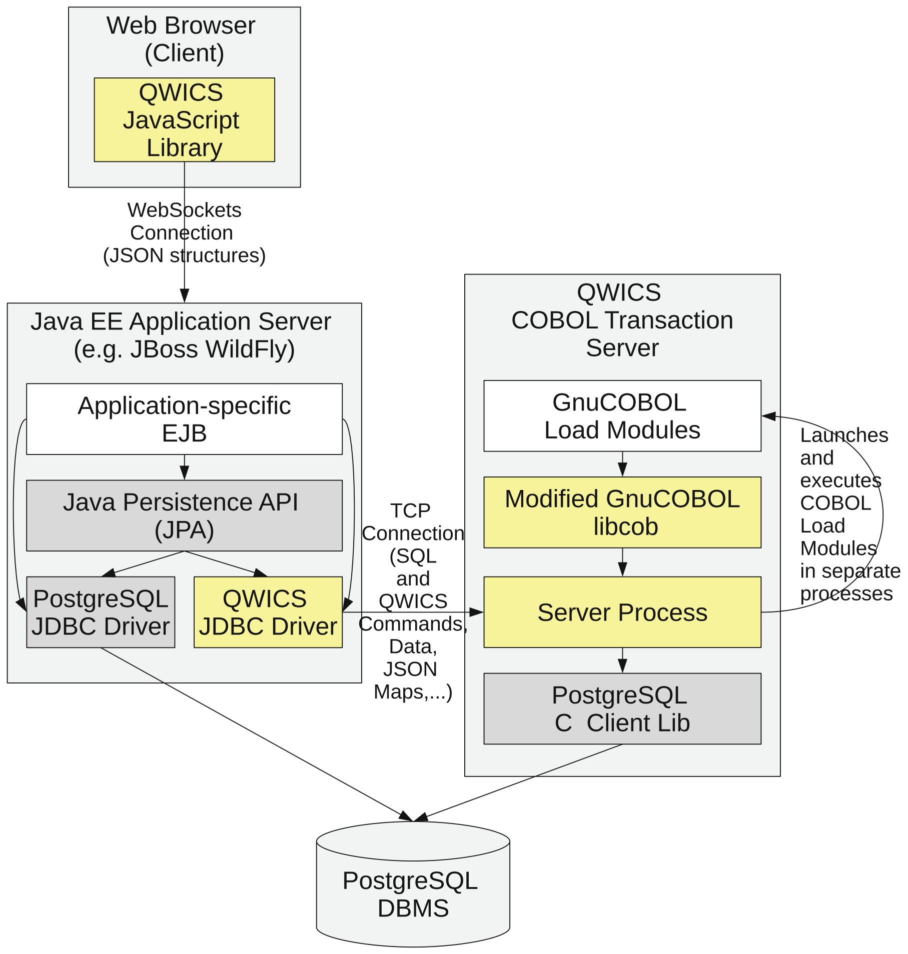

Web Browser (Client)
QWICS JavaScript Library
WebSockets Connection (JSON structures)
Java EE Application Server (e.g. JBoss WildFly)
Application-specific EJB
Java Persistence API (JPA)
PostgreSQL JDBC Driver
QWICS JDBC Driver
TCP Connection (SQL and QWICS Commands, Data, JSON Maps,...)
QWICS COBOL Transaction Server
GnuCOBOL Load Modules
Modified GnuCOBOL libcob
Server Process
PostgreSQL C Client Lib
Launches and executes COBOL Load Modules in separate processes
PostgreSQL DBMS

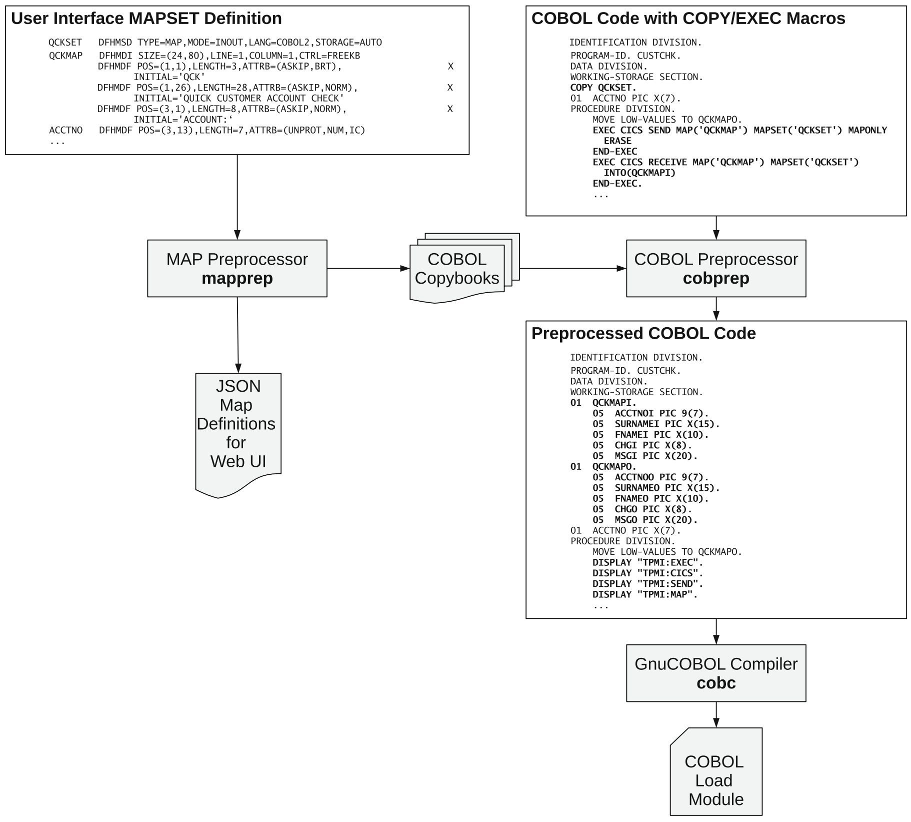
Fig. 1. Overview of the software architecture of QWICS. The arrows denote usage/invocation relationships. Yellow boxes describe the QWICS-specific components and white boxes the application-specific COBOL or Java code. The integration between the Java and the COBOL code is achieved using a specific JDBC driver calling the COBOL server via its own protocol over a TCP connection. Reprinted from [6]. (Color figure online)

are usually coded in COBOL (or that of any other supported language) by using TPM- and SQL-specific `EXEC ... END-EXEC` macros. These macros are then translated into TPM API calls in an additional preprocessing step before actually compiling the COBOL code [25]. In addition, the terminal UI screen definitions (so-called maps) referenced by these macros also need to be translated to stored COBOL code fragments (called copybooks) containing the necessary variable declarations. These copybooks are then inserted in the COBOL code during preprocessing as well [6].

```
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
```

Fig. 2. Necessary modification to `termio.c` of GnuCOBOL's `libcob` runtime library. Only the lines shown between `// BEGIN...` and `// END ...` 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. Reprinted from [6].

This preprocessing needs to be mimicked by QWICS to allow to re-use the unmodified COBOL source codes. In Fig. 3, the overall process implemented in QWICS by two preprocessors written in C, `cobprep` for COBOL and `mapprep` for the map definitions, is illustrated. After the preprocessing, the resulting COBOL code is compiled to an executable load module using the GnuCOBOL compiler [6].



```
QCKSET   DFHMSD TYPE=MAP,MODE=INOUT,LANG=COBOL2,STORAGE=AUTO
QCKMAP   DFHMDI SIZE=(24,80),LINE=1,COLUMN=1,CTRL=FREEKB
         DFHMDF POS=(1,1),LENGTH=3,ATTRB=(ASKIP,BRT),                    X
               INITIAL='QCK'
         DFHMDF POS=(1,26),LENGTH=28,ATTRB=(ASKIP,NORM),                 X
               INITIAL='QUICK CUSTOMER ACCOUNT CHECK'
         DFHMDF POS=(3,1),LENGTH=8,ATTRB=(ASKIP,NORM),                   X
               INITIAL='ACCOUNT:‘
ACCTNO   DFHMDF POS=(3,13),LENGTH=7,ATTRB=(UNPROT,NUM,IC)
...
```

```
IDENTIFICATION DIVISION.
PROGRAM-ID. CUSTCHK.
DATA DIVISION.
WORKING-STORAGE SECTION.
COPY QCKSET.
01  ACCTNO PIC X(7).
PROCEDURE DIVISION.
    MOVE LOW-VALUES TO QCKMAPO.
    EXEC CICS SEND MAP('QCKMAP') MAPSET('QCKSET') MAPONLY
      ERASE
    END-EXEC
    EXEC CICS RECEIVE MAP('QCKMAP') MAPSET('QCKSET')
      INTO(QCKMAPI)
    END-EXEC.
    ...
```

```
IDENTIFICATION DIVISION.
PROGRAM-ID. CUSTCHK.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  QCKMAPI.
    05  ACCTNOI PIC 9(7).
    05  SURNAMEI PIC X(15).
    05  FNAMEI PIC X(10).
    05  CHGI PIC X(8).
    05  MSGI PIC X(20).
01  QCKMAPO.
    05  ACCTNOO PIC 9(7).
    05  SURNAMEO PIC X(15).
    05  FNAMEO PIC X(10).
    05  CHGO PIC X(8).
    05  MSGO PIC X(20).
01  ACCTNO PIC X(7).
PROCEDURE DIVISION.
    MOVE LOW-VALUES TO QCKMAPO.
    DISPLAY "TPMI:EXEC".
    DISPLAY "TPMI:CICS".
    DISPLAY "TPMI:SEND".
    DISPLAY "TPMI:MAP".
    ...
```

Fig. 3. Process of preprocessing the original COBOL source files and map definitions for use by QWICS. Reprinted from [6].

Last but not least a *JavaScript Library* has been implemented for QWICS, which supports the (optional) implementation of web user interfaces by converting the original TPM's map defintions [25] to JavaScript Object Notation (JSON) structures beforehand using the preprocessor [6].

4 Extension to Transactional Message-Processing

Asynchronous, transactional message-processing is an important concept of large-scale enterprise applications [4]. In traditional mainframe applications this functionality is typically provided by a message-oriented middleware invoked from COBOL programs running in a TPM. A similar concept exists for Java EE application servers through the Java Message Service (JMS).

Following the general approach of QWICS, transactional message-processing was therefore implemented by using JMS from the COBOL code by extending the QWICS JDBC driver. To do so, the COBOL program calls for getting and

```
// BEGIN OF CALL HANDLER
void* (*resolveCALL)(char*) = NULL;
// END OF CALL HANDLER

void *
cob_resolve_cobol (const char *name, const int fold_case, const int errind)
{
        void    *p;
        char    *entry;
        char    *dirent;

// BEGIN OF CALL HANDLER
        p = resolveCALL(name);
        if (p != NULL) {
           return p;
        }
// END OF CALL HANDLER
```

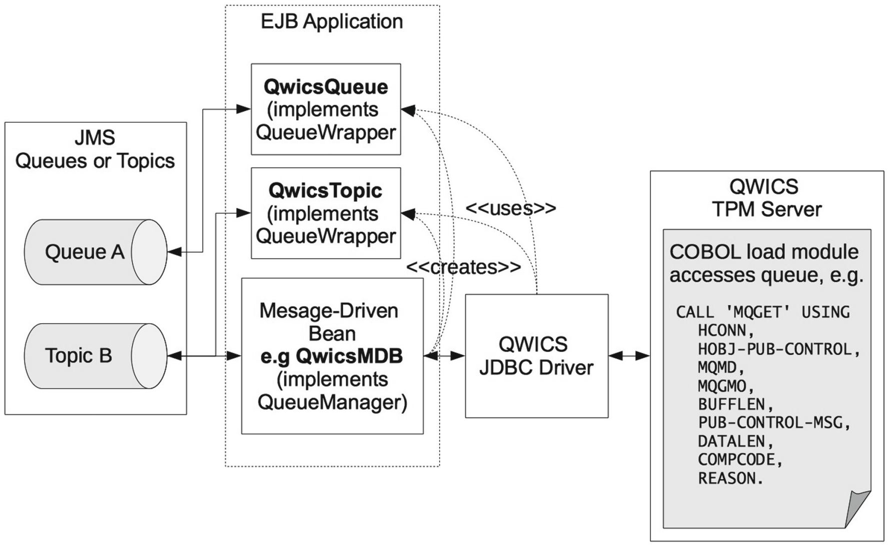
Fig. 4. Necessary modification to `call.c` of GnuCOBOL's `libcob` runtime library. Only the lines shown between `// BEGIN...` and `// END ...` need to be added, no further modifications are necessary. This code adds an interception to COBOL `CALL` statements, which executes the function denoted by the pointer `resolveCALL(..)` if it is set.

putting messages from and into queues or topics [4] need to be intercepted by the QWICS transaction server. Therefore, again a patch has been added to the GnuCOBOL `libcob` library to intercept and redirect the respective `CALL` statements to corresponding C functions provided by the transaction server. Figure 4 shows the respective modification to the `libcob` library.

The transaction server communicates with the QWICS JDBC driver to send and receive messages via JMS. Figure 5 shows a conceptual overview of the interplay between the components involved in this process. Since JDBC datasource management, JTA and JMS are independent subsystems of Java EE, a JDBC driver should not depend on any JMS API or call it directly. Therefore, the integration of COBOL with JMS requires a custom Java EE application using message-driven beans (MDB).

While the JDBC driver only offers interfaces representing abstract wrappers for queues or topics (`QueueWrapper`) and a factory for creating them (`QueueManager`), the Java EE application implements these interfaces to actually access the JMS functionality. It registers a `QueueManager` implementation with the JDBC driver, so the latter could create instances of these classes. The UML class diagram in Fig. 6 illustrates the relevant Java classes of the JDBC driver and the example Java EE application and there relations.

For every JMS queue or topic for which messages should be processed within a transaction by a COBOL program, a corresponding MDB needs to be implemented, for which the example `QwicsMDB` may serve as a blueprint. When such a MDB receives a message, its method `onMessage(Message message)` is invoked by the EJB container, wrapped into a distributed JTA transaction. This method


EJB Application
QwicsQueue (implements QueueWrapper
QwicsTopic (implements QueueWrapper
JMS Queues or Topics
Queue A
Topic B
<<uses>>
<<creates>>
Mesage-Driven Bean e.g QwicsMDB (implements QueueManager)
QWICS JDBC Driver
QWICS TPM Server
COBOL load module accesses queue, e.g.

```
CALL 'MQGET' USING
   HCONN,
   HOBJ-PUB-CONTROL,
   MQMD,
   MQGMO,
   BUFFLEN,
   PUB-CONTROL-MSG,
   DATALEN,
   COMPCODE,
   REASON.
```

Fig. 5. Conceptual overview of the integration between COBOL and JMS via the QWICS framework. The actual access to JMS is implemented within an EJB application and not inside the QWICS JDBC driver to keep the different Java EE subsystems independent from each other. A message-driven bean listens at a queue or topic and upon reception of a message triggers the corresponding COBOL program using the JDBC driver. The COBOL program accesses the JMS queues via the JDBC driver and the EJB application.

now uses the QWICS JDBC driver via a XA datasource to trigger the respective COBOL program.

Figure 7 shows an excerpt from the source code of the example `QwicsMDB` to illustrate this. The COBOL program invoked here is a demo program called `QPUBCBL`. By the statement `maps.updateObject(‘‘QMGR’’, this);`, the MDB registers itself as the `QueueManager` implementation with the JDBC driver.

The invoked COBOL load module now may access the JMS queues itself by executing `CALL` statements to the (emulated) routines “MQOPEN”, “MQCLOSE”, “MQGET” and “MQPUT”. Intercepted by the transaction server as described above, these calls are forwarded to the JDBC driver, which again uses the implementation classes `QwicsTopic` or `QwicsQueue` provided by the Java EE application to access the respective JMS artifacts.

With this mechanism, analogous to the previously described QWICS functionality, COBOL programs will be triggered by JMS messages are able to send and receive messages via JMS.

5 Experimental Evaluation

To evaluate the functionality and usefulness of QWICS, in a first step an existing transactional COBOL application was ported to the QWICS evironment [6]. To avoid a bias, a representative COBOL application written for training purposes using the IBM CICS TPM a consulting company was used for this [33], since its developers are not related or known to the author [6].

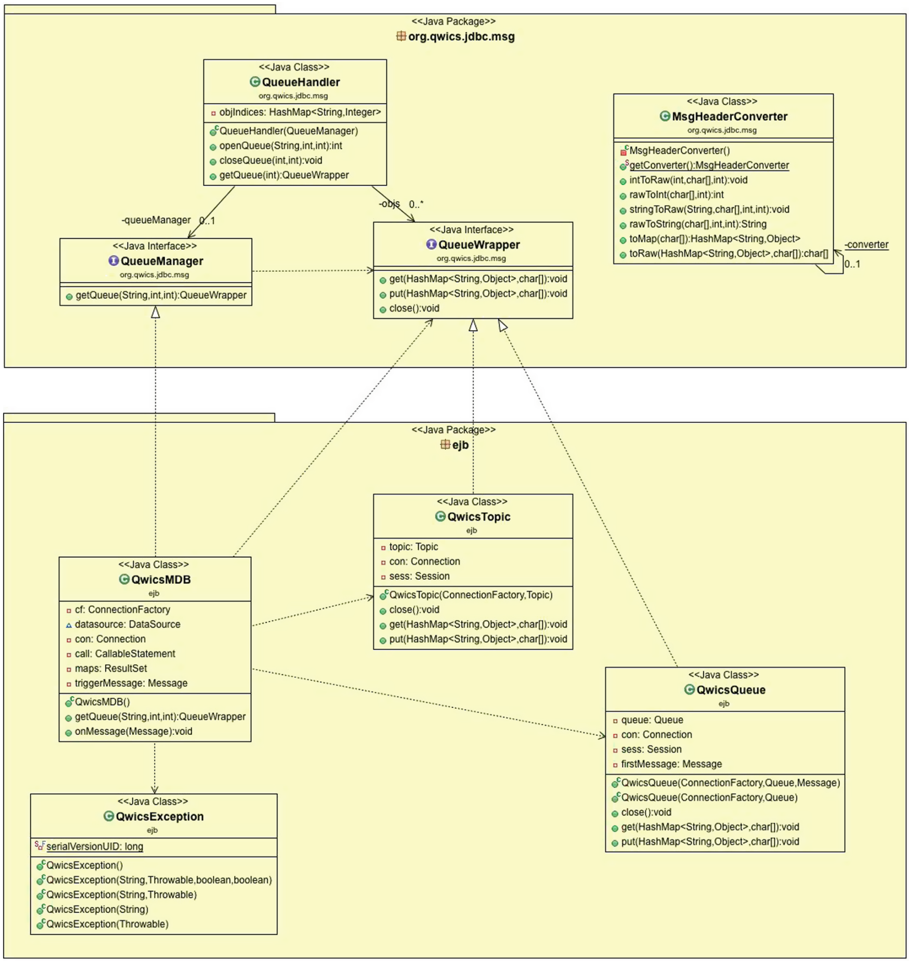

<<Java Package>>
org.qwics.jdbc.msg
<<Java Class>>
QueueHandler
org.qwics.jdbc.msg
objIndices: HashMap<String,Integer>
QueueHandler(QueueManager)
openQueue(String,int,int):int
closeQueue(int,int):void
getQueue(int):QueueWrapper
<<Java Class>>
MsgHeaderConverter
org.qwics.jdbc.msg
MsgHeaderConverter()
getConverter():MsgHeaderConverter
intToRaw(int,char[],int):void
rawToInt(char[],int):int
stringToRaw(String,char[],int,int):void
rawToString(char[],int,int):String
toMap(char[]):HashMap<String,Object>
toRaw(HashMap<String,Object>,char[]):char[]
-converter
0..1
-queueManager
0..1
-objs
0..*
<<Java Interface>>
QueueManager
org.qwics.jdbc.msg
getQueue(String,int,int):QueueWrapper
<<Java Interface>>
QueueWrapper
org.qwics.jdbc.msg
get(HashMap<String,Object>,char[]):void
put(HashMap<String,Object>,char[]):void
close():void
<<Java Package>>
ejb
<<Java Class>>
QwicsTopic
ejb
topic: Topic
con: Connection
sess: Session
QwicsTopic(ConnectionFactory,Topic)
close():void
get(HashMap<String,Object>,char[]):void
put(HashMap<String,Object>,char[]):void
<<Java Class>>
QwicsMDB
ejb
cf: ConnectionFactory
datasource: DataSource
con: Connection
call: CallableStatement
maps: ResultSet
triggerMessage: Message
QwicsMDB()
getQueue(String,int,int):QueueWrapper
onMessage(Message):void
<<Java Class>>
QwicsQueue
ejb
queue: Queue
con: Connection
sess: Session
firstMessage: Message
QwicsQueue(ConnectionFactory,Queue,Message)
QwicsQueue(ConnectionFactory,Queue)
close():void
get(HashMap<String,Object>,char[]):void
put(HashMap<String,Object>,char[]):void
<<Java Class>>
QwicsException
ejb
serialVersionUID: long
QwicsException()
QwicsException(String,Throwable,boolean,boolean)
QwicsException(String,Throwable)
QwicsException(String)
QwicsException(Throwable)

Fig. 6. UML class diagram describing the Java-side implementation of an example Message-Driven Bean (`QwicsMDB`) using the newly added message-queueing functionality of the QWICS JDBC driver. The MDB class listens for and reacts to received messages from a JMS queue or topic and provides the driver with appropriate `QueueWrapper` implementation classes (labeled `QwicsQueue` and `QwicsTopic` here) for accessing the respective JMS objects.

```
@MessageDriven(...,
        messageListenerInterface = MessageListener.class)
public class QwicsMDB
        implements MessageListener, QueueManager {
        ...
        @Resource(mappedName="java:jboss/datasources/QwicsDS")
        DataSource datasource;

        private Connection con;
        private CallableStatement call;
        private ResultSet maps;
        private Message triggerMessage = null;
        ...
        public void onMessage(Message message) {
           try {
                triggerMessage = message;
                con = datasource.getConnection();
                call = con.prepareCall("PROGRAM QPUBCBL");
                maps = call.executeQuery();
                maps.updateString("QNAME","MyQueue");
                maps.updateString("ENVDATA","MyStatQueue");
                maps.updateObject("QMGR", this);
                maps.next();
                try {
                        String ac = maps.getString("ABCODE");
                        throw new QwicsException("ABEND "+ac);
                } catch (Exception e) {
                }
           } catch (Exception e) {
                throw new QwicsException(e);
           }
        }
}
```

Fig. 7. Excerpt of the example Java EE message-driven bean (MDB) listening for messages on a JMS queue or topic. Upon reception of a respective message, the method `onMessage(Message message)` is invoked by the container, wrapped into a distributed XA transaction. The message is only removed from the queue of this transaction is commited successfully. The method invokes a sample message-processing COBOL program via the QWICS JDBC driver.

The original sources were passed through the preprocessors as described above, compiled using GnuCOBOL and then run in the QWICS environment. The screenshot shown in Fig. 8 illustrates how this COBOL application running in QWICS may appear to user in a web browser window. This original evaluation was repeated two times, first on an Apple MacBook Air laptop running MacOS X 10.11.6 (thus, a BSD Un*x- derivative), and second on a IBM zBC12 mainframe computer running Linux on Z. Both tests worked smoothly, delivered identical results regarding the functionality and thus demonstrate the feasibilty of the approach in principle [6].

Subsequent to the original publication of these evaluation results [6], in a second step a public community website (https://qwics.org) including a free demonstration and testing environment for QWICS has been set up, to further evaluate the QWICS framework and move it towards production readiness.

This website provides further information on QWICS, links to the source code repository and after a free online registration offers everyone the possibilty to run a personal QWICS environment in a Virtual Machine (VM) based on a Docker container[15]. Besides the actual QWICS framework, this environment offers the user a web-based administration console for the PostgreSQL database (using the OSS phpPgAdmin[16]) and a web-based source code editor to write COBOL code and edit UI screens online (using the Codiad IDE[17]). Figure 9 shows screenshots of the dashboard and the online code editor of this environment. As can also be seen from the screenshot, the QWICS online environment currently still runs on an Intel x86-based server running Ubuntu Linux. It is planned also to provide a demo running on the mainframe under Linux on Z in the future.

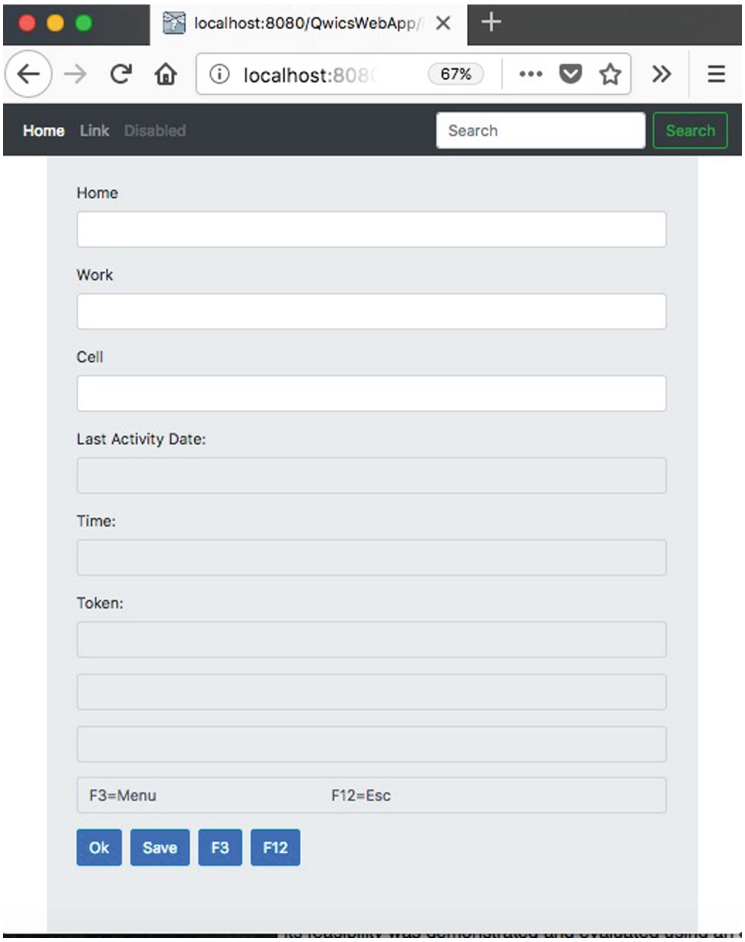
The QWICS server VM offered on https://qwics.org to the public already contains a simple but ready-to-run transactional COBOL example written by the author for demonstration purposes, in the form of a small guestbook web app. This online version of QWICS not only allows to demonstrate and explore the framework, but also may serve as an easy to use and free opportunity to learn and practice transactional COBOL programming online.

The availability of this QWICS online trial has been promoted via a press release and internationally via social media so far. Until now, a small number of interested persons from different countries have registered, but it is yet too early to obtain results from the analysis of user feedback.

It has been pointed out before that QWICS is different from other mainframe modernization approaches with respect to its focus on using OSS to integrate existing COBOL code into Java EE applications to use modern web technologies with COBOL [6]. Therefore, it requires a partial adaption and recompilation of the existing sources instead of achieving full binary[18] or source-level compatibilty [2,37] on other (commodity), non-mainframe platforms [6]. Instead, it enables customers to modernize transactional COBOL applications on the mainframe itself using Linux.

The next research steps will focus on evaluating the potentials of QWICS in real-world case studies, on the one hand by an extended Proof-of-Concept using a real "legacy" COBOL application in a company, on the other hand by further

[15] https://www.docker.com/.
[16] http://phppgadmin.sourceforge.net/doku.php.
[17] http://codiad.com/.
[18] https://www.lzlabs.com/



Home

Work

Cell

Last Activity Date:

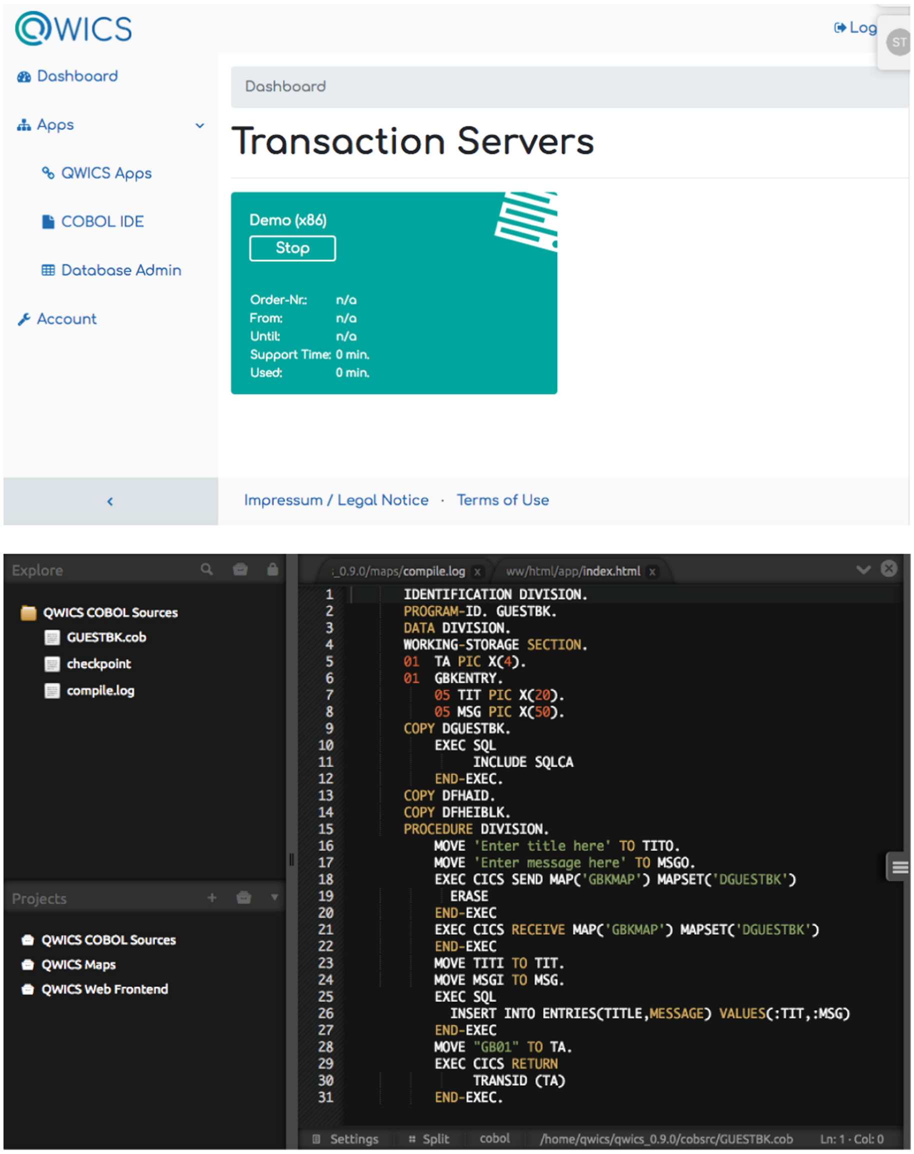
Time:

Token:

F3=Menu F12=Esc

Fig. 8. Screenshot of a map screen converted to JSON and displayed in web page by the JavaScript library. Reprinted from [6].

exploring the use of the QWICS online platform for COBOL programming education. The first will also need include an extended analysis of the performance and scalability of the approach compared to the original mainframe TPM. As described in [6], there definitely will be a performance tradeoff due to the design of the used OSS components, but it remains an open issue how big it will be in practice.



```
IDENTIFICATION DIVISION.
PROGRAM-ID. GUESTBK.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  TA PIC X(4).
01  GBKENTRY.
    05 TIT PIC X(20).
    05 MSG PIC X(50).
COPY DGUESTBK.
    EXEC SQL
         INCLUDE SQLCA
    END-EXEC.
COPY DFHAID.
COPY DFHEIBLK.
PROCEDURE DIVISION.
    MOVE 'Enter title here' TO TITO.
    MOVE 'Enter message here' TO MSGO.
    EXEC CICS SEND MAP('GBKMAP') MAPSET('DGUESTBK')
      ERASE
    END-EXEC
    EXEC CICS RECEIVE MAP('GBKMAP') MAPSET('DGUESTBK')
    END-EXEC
    MOVE TITI TO TIT.
    MOVE MSGI TO MSG.
    EXEC SQL
      INSERT INTO ENTRIES(TITLE,MESSAGE) VALUES(:TIT,:MSG)
    END-EXEC
    MOVE "GB01" TO TA.
    EXEC CICS RETURN
         TRANSID (TA)
    END-EXEC.
```

Fig. 9. Screenshots from the free QWICS online demonstration and testing environment: Admin Dashboard showing a running QWICS VM instance (above) and the online source code editor showing a COBOL example (below).

6 Conclusion

In conclusion, in this paper the previously proposed Quick Web-Based Interactive COBOL Service (QWICS) [6] was presented and its extension to support asynchronous, transactional, message-oriented communication in COBOL via the Java Message Service (JMS) API was introduced. The latter is in particular important for building large-scale enterprise applications.

Therefore, QWICS now allows to run the most relevant types of transactional COBOL applications (synchronous and asynchronous) within the context of a Java EE application server using a pure OSS stack. Therefore, it allows to mix COBOL and Java code for extending and converting "legacy" applications into web services. Its pure OSS stack runs on most Un*x and Linux platforms, allowing in particular to modernize "legacy" applications on the mainframe itself using Linux.

The feasibility of the approach has already been evaluated by porting a semi-realistic third-party COBOL application to QWICS [6], as well as by making it available for public use via an online demonstration and testing environment at https://qwics.org. Also, meanwhile the community feedback received via the GitHub source code repository of QWICS (See footnote 12) has been taken into account as far as possible.

However, since QWICS is still a proof-of-concept implementation and thus not feature-complete, or its production readiness further extensions and adaptations may be necessary, which should be detected and addressed during migration of a real-world application. Therefore, further research is needed to explore the approach in a real industry case study.

References

1. Abbany, Z.: Fail by design: banking's legacy of dark code. https://m.dw.com/en/fail-by-design-bankings-legacy-of-dark-code/a-43645522 (2018). Accessed 05 Jan 2019
2. Apte, A., et al.: Method and apparatus for migration of application source code, US Patent App. 15/397,473 (2017)
3. Bainbridge, A., Colgrave, J., Colyer, A., Normington, G.: CICS and Enterprise JavaBeans. IBM Syst. J. **40**(1), 46–67 (2001)
4. Banavar, G., Chandra, T., Strom, R., Sturman, D.: A case for message oriented middleware. In: Jayanti, P. (ed.) DISC 1999. LNCS, vol. 1693, pp. 1–17. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48169-9_1
5. Bodhuin, T., Guardabascio, E., Tortorella, M.: Migrating COBOL systems to the WEB by using the MVC design pattern. In: Proceedings of the Ninth Working Conference on Reverse Engineering, 2002, pp. 329–338. IEEE (2002)
6. Brune, P.: A hybrid approach to re-host and mix transactional cobol and java code in java ee web applications using open source software. In: Proceedings of the 14th International Conference on Web Information Systems and Technologies - Volume 1: WEBIST, pp. 239–246. INSTICC, SciTePress (2018)
7. Calladine, J.: Giving legs to the legacy–web services integration within the enterprise. BT Technol. J. **22**(1), 87–98 (2004)
8. El Beggar, O., Bousetta, B., Gadi, T.: Getting objects methods and interactions by extracting business rules from legacy systems. J. Syst. Integr. **5**(3), 32 (2014)
9. Farmer, E.: The reality of rehosting: understanding the value of your mainframe (2013)

10. Ferguson, D.F., Stockton, M.L.: Service-oriented architecture: programming model and product architecture. IBM Syst. J. **44**(4), 753–780 (2005)
11. FinTech Futures: How open will your bank become?. https://www.bankingtech.com/2018/11/how-open-will-your-bank-become/ (2018). Accessed 05 Jan 2019
12. Hashem, I.A.T., Yaqoob, I., Anuar, N.B., Mokhtar, S., Gani, A., Khan, S.U.: The rise of "big data" on cloud computing: review and open research issues. Inf. Syst. **47**, 98–115 (2015)
13. Huang, H., Tsai, W.T., Bhattacharya, S., Chen, X., Wang, Y., Sun, J.: Business rule extraction techniques for COBOL programs. J. Softw.: Evol. Process **10**(1), 3–35 (1998)
14. Kanter, H.A., Muscarello, T.J.: Reuse versus rewrite: an empirical study of alternative software development methods for web-enabling mission-critical COBOL/CICS legacy applications. Fujitsu Software, CICS Legacy Applications (2005)
15. Karremans, J.: Postgres in the enterprise: real world reasons for adoption. https://www.enterprisedb.com/blog/postgres-enterprise-real-world-reasons-adoption (2018). Accessed 05 Jan 2019
16. Khadka, R., Batlajery, B.V., Saeidi, A.M., Jansen, S., Hage, J.: How do professionals perceive legacy systems and software modernization? In: Proceedings of the 36th International Conference on Software Engineering. pp. 36–47. ACM (2014)
17. Khadka, R., et al.: Does software modernization deliver what it aimed for? a post modernization analysis of five software modernization case studies. In: 2015 IEEE International Conference on Software Maintenance and Evolution (ICSME), pp. 477–486. IEEE (2015)
18. Kiefer, C.: COBOL as a modern language. https://digitalcommons.northgeorgia.edu/honors_theses/17/ (2017). Accessed 27 July 2018
19. Knoche, H., Hasselbring, W.: Using microservices for legacy software modernization. IEEE Softw. **35**(3), 44–49 (2018)
20. Lämmel, R., De Schutter, K.: What does aspect-oriented programming mean to COBOL? In: Proceedings of the 4th International Conference on Aspect-Oriented Software Development, pp. 99–110. ACM (2005)
21. Lancia, M., Puccinelli, R., Lombardi, F.: Feasibility and benefits of migrating towards JEE: a real life case. In: Proceedings of the 5th International Symposium on Principles and Practice Of Programming in Java, pp. 13–20. ACM (2007)
22. Lee, M.S., Shin, S.G., Yang, Y.J.: The design and implementation of Enterprise JavaBean (EJB) wrapper for legacy system. In: 2001 IEEE International Conference on Systems, Man, and Cybernetics, vol. 3, pp. 1988–1992. IEEE (2001)
23. Lymer, S.F., Starkey, M., Stephenson, J.W.: System for automated interface generation for computer programs operating in different environments, US Patent 6,230,117, 8 May 2001
24. Mainetti, L., Paiano, R., Pandurino, A.: MIGROS: a model-driven transformation approach of the user experience of legacy applications. In: Brambilla, M., Tokuda, T., Tolksdorf, R. (eds.) ICWE 2012. LNCS, vol. 7387, pp. 490–493. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31753-8_51
25. Malaika, S., Park, H.: A tale of a transaction monitor. IEEE Data Eng. Bull. **17**(1), 3–9 (1994)
26. Mateos, C., Zunino, A., Misra, S., Anabalon, D., Flores, A.: Migration from COBOL to SOA: measuring the impact on web services interfaces complexity. In: Damaševičius, R., Mikašytė, V. (eds.) ICIST 2017. CCIS, vol. 756, pp. 266–279. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-67642-5_22

27. Moore, G.: Systems of engagement and the future of enterprise IT: a sea change in enterprise IT. AIIM Whitepaper (2011)
28. Nelson, J.: Why banks didn't 'rip and replace' their mainframes. https://www.networkworld.com/article/3305745/hardware/why-banks-didnt-rip-and-replace-their-mainframes.html (2018). Accessed 05 Jan 2019
29. Rodriguez, J.M., Crasso, M., Mateos, C., Zunino, A., Campo, M.: Bottom-up and top-down cobol system migration to web services. IEEE Internet Comput. **17**(2), 44–51 (2013)
30. Sagers, G., Ball, K., Hosack, B., Twitchell, D., Wallace, D.: The mainframe is dead. Long live the mainframe!. AIS Trans. Enterp. Syst. **4**, 4–10 (2013)
31. Sellink, A., Sneed, H., Verhoef, C.: Restructuring of COBOL/CICS legacy systems. In: Proceedings of the Third European Conference on Software Maintenance and Reengineering, 1999, pp. 72–82. IEEE (1999)
32. Sellink, A., Sneed, H., Verhoef, C.: Restructuring of COBOL/CICS legacy systems. Sci. Comput. Program. **45**(2–3), 193–243 (2002)
33. SimoTime Technologies and Services: The CICS connection, sample programs for CICS. http://www.simotime.com/indexcic.htm. Accessed 21 Feb 2018
34. Sneed, H.M.: Migration of procedurally oriented COBOL programs in an object-oriented architecture. In: Proceerdings of the Conference on Software Maintenance, 1992, pp. 105–116. IEEE (1992)
35. Sneed, H.M.: Wrapping legacy COBOL programs behind an XML-interface. In: Proceedings of the Eighth Working Conference on Reverse Engineering, 2001, pp. 189–197. IEEE (2001)
36. Suganuma, T., Yasue, T., Onodera, T., Nakatani, T.: 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, pp. 685–696. ACM (2008)
37. Talati, K., Lackie, C.W.: Virtual software machine for enabling CICS application software to run on UNIX based computer systems, uS Patent 6,006,277, 21 December 1999
38. The Financial Brand: The four pillars of digital transformation in banking. https://thefinancialbrand.com/71733/four-pillars-of-digital-transformation-banking-strategy/ (2018). Accessed 05 Jan 2019
39. Tommy, R., Ravi, U., Mohan, D., Luke, J., Krishna, A.S., Subramaniam, G.: Internet of Things (IoT) expanding the horizons of mainframes. In: 2015 5th International Conference on IT Convergence and Security (ICITCS), pp. 1–4. IEEE (2015)
40. Vinaja, R.: 50th aniversary of the mainframe computer: a reflective analysis. J. Comput. Sci. Coll. **30**(2), 116–124 (2014)
41. White, J.W.: Portable and dynamic distributed transaction management method, US Patent 6,115,710, 5 September 2000
42. Wilkes, A.: The mainframe evolution: banking still needs workhorse tech. https://www.finextra.com/blogposting/16067/the-mainframe-evolution-banking-still-needs-workhorse-tech (2018). Accessed 05 Jan 2019
43. Zhou, N., Zhang, L.J., Chee, Y.M., Chen, L.: Legacy asset analysis and integration in model-driven SOA solution. In: 2010 IEEE International Conference on Services Computing (SCC), pp. 554–561. IEEE (2010)