Converting Robot Offline Programs to Native Code Using the AdaptPack Studio Translators

João Pedro Souza, André Castro, Luís Rocha, Pedro Relvas
INESC TEC
Porto, Portugal
{joao.p.souza, andre.l.castro, luis.f.rocha, pedro.m.relvas}@inesctec.pt

Manuel F. Silva
INESC TEC and ISEP-IPP
Porto, Portugal
mss@isep.ipp.pt

***Abstract*—The increase in productivity is a demand for modern industries that need to be competitive in the actual business scenario. To face these challenges, companies are increasingly using robotic systems for end-of-line production tasks, such as wrapping and palletizing, as a mean to enhance the production line efficiency and products traceability, allowing human operators to be moved to more added value operations. Despite this increasing use of robotic systems, these equipments still present some inconveniences regarding the programming procedure, as the time required for its execution does not meet the current industrial needs. To face this drawback, offline robot programming methods are gaining great visibility, as their flexibility and programming speed allows companies to face the need of successive changes in the production line set-up. However, even with a great number of robots and simulators that are available in market, the efforts to support several robot brands in one software did not reach the needs of engineers. Therefore, this paper proposes a translation library named AdaptPack Studio Translator, which is capable to export proprietary codes for the ABB, Fanuc, Kuka, and Yaskawa robot brands, after their offline programming has been performed in the Visual Components software. The results presented in this paper are evaluated in simulated and real scenarios.**

***Index Terms*—Industrial Application, Robot Language Translator, Simulation, Offline Programming**

I. Introduction

The adoption of robots in the production scale of Fast Moving Consumer Goods (FMCG) industry has been increasing according to the rise of global trade competitiveness. The goals are the high performance, efficiency, and flexibility focusing on the productivity while helping industries to handle dangerous and repetitive tasks.

Regarding the programming and the setting up of a robotic cell, two approaches can be highlighted: the online and the offline programming. The first one, the most commonly used by industries, consists of manually teaching points to a robot supervised by a human operator in a real scenario. More specifically, the operator moves the robot end-effector with a teaching pendant to determined positions and creates the program based on them. It is also called the lead-through method. This programming method depends on the operator skill, and several authors propose assisted online programming to help in this procedure [1]–[4]. Some drawbacks of this method are the test procedures which are time-consuming and lack of program flexibility. Thus, the process becomes expensive, tedious and repetitive. The offline method is commonly based on a previous programming of the robot in a simulated environment. Therefore, it is possible to create complex and flexible systems, besides test it in different situations [5]. As reported by the authors in [5], this procedure consists of a chain of steps, including 3D modeling, trajectory and process planning, simulation, calibration and post-processing.

After completing the development of a robotic cell using the offline programming approach, the simulated program needs to be translated to the real robot controller. Concerning this method, robot programmers face the following issue: there exists a great number of simulators and robot brands available in the market, and the attempt to standardize both their interfaces and the robot programming languages has been quite modest. One example is the development of the Industrial Robot Language (IRL) [6], that despite being a standard, is not truly used (considering the authors knowledge) by any robot manufacturer. Thus, it is necessary to program different routines according to the robot brand. This is a problem for industries that, aiming to be competitive, need fast and flexible solutions to handle different requirements. In relation to this, the authors in [7] and [8] propose a framework for offline robot programming that generates codes to several robots; however, only some basic commands are translated. The paper [9] presents a framework to allow interpretation of different native robot languages assisting the simulation phase, *i.e.*, a native language translator to a simulation language. Nonetheless, this approach requires the knowledge of the proprietary used languages.

This paper addresses the translation of programs developed offline to the robot native language with the creation of the AdaptPack Studio Translator Library, integrated in the Visual Components (VC) simulation software [10]. This library is an extension of the AdaptPack project [11], [12] which proposes to research and develop a framework focused on the design and development of modular robotic packaging and palletizing sys-

This work is co-financed by the ERDF European Regional Development Fund through the Operational Programme for Competitiveness and Internationalisation - COMPETE 2020 under the PORTUGAL 2020 Partnership Agreement, and through the Portuguese National Innovation Agency (ANI) as a part of project AdaptPack: POCI-01-0247-FEDER-017887

978-1-7281-3558-8/19/$31.00 ©2019 IEEE

tems. The proposed approach includes a number of innovative components that support, in an efficient and integrated way, the conception, design and evaluation of packaging/palletizing solutions and, at a later stage the assembly of the solution on the shop floor.

The main contributions of the present paper are the following:

- It is proposed a library to automatic translate offline robot programs from the VC software to native robot languages of the following manufacturers: ABB, Fanuc, Kuka, and Yaskawa. This approach allows engineers to create robotic cells solutions in a simulated environment, test it and simply export the code to the robot controller.
- It is performed a comparative evaluation of the following proprietary languages: Rapid (ABB), Karel (Fanuc), KRL (Kuka) and Inform (Yaskawa).

Bearing these ideas in mind, the remainder of the paper is organized as follows: after this introductory section, Section II presents the comparative evaluation between the proprietary languages used in this work. Section III describes the AdaptPack Translator Library and its structure. Section IV presents the experimental results obtained using the proposed library. Finally, Section V presents the main conclusions of the developed work and some ideas for possible future developments.

II. Comparative Evaluation of Some of the Most Used Robot Proprietary Languages

As referred in the previous section, this paper addresses the problem of off-line programming of robots from different manufacturers (with their corresponding proprietary programming languages), namely: ABB (RAPID), FANUC (Karel), KUKA (KRL) and Yaskawa (Inform). A description of each of these programming languages is performed in the following subsections, being also presented their comparative evaluation.

A. RAPID

RAPID is the high-level programming language developed by ABB [13]. The instructions and variables used in the VC software that have a direct correspondence in this language are shown in Tables I and II. In RAPID, the registers are modified if any change occurs in the controller memory while the program is running. Thus, the user needs to be aware of this detail if a re-execution of the code is needed. The operations with coordinate references are performed using already available RAPID language instructions. For example, any frame can be translated and rotated by the instructions presented in Code Script 1, and a transformation in relation to a tool or base frame is defined by a set of parameters, as shown in Code Script 2.

```
frame.uframe.trans := [Δx, Δy, Δz];
frame.uframe.rot := OrientZYX(Δω_x, Δω_y, Δω_z);
```

Code Script 1. Rapid references relationship methods

```
MoveJ GoalPose, v200,fine,toolFrame\WObj:=pickPlace;
MoveL GoalPose, v200,fine,toolFrame\WObj:=pickPlace;
```

Code Script 2. Rapid base and tool setup of a movement

B. Karel

Fanuc has two different programming approaches: using TP or using Karel. The TP language is the language used in the teach pendant and it is FANUC's lower-level programming language. Karel, on the other hand, is FANUC's high-level programming language [14]. Different from the other languages discussed in this section, the Karel code needs to be compiled before being uploaded to the robot controller.

As in the RAPID case, this language allows modular programming. The instructions and variables used in the VC software that have their correspondence in Karel are presented in Tables I and II, respectively. The operations with coordinate frames are performed by independently setting each component of the frame, as presented in Code Script 3, and its reference is set in the motion instructions parameters, as depicted in the Code Script 4.

```
frame.x := Δx; frame.y = Δy, frame.z = Δz;
frame.w := Δω_x; frame.p = Δω_y, frame.r = Δω_z;
```

Code Script 3. Rapid references relationship methods

```
WITH $TERMTYPE = FINE, $MOTYPE = JOINT, $SPEED = 1000,
     $UFRAME = RobotBase, $UTOOL = ToolFrame MOVE TO
     GoalPose
WITH $TERMTYPE = FINE, $MOTYPE = LINEAR, $SPEED = 1000,
     $UFRAME = RobotBase, $UTOOL = ToolFrame MOVE TO
     GoalPose
```

Code Script 4. Rapid references relationship methods

Some requirements of offline programming are the capability to adjust positions and frames according to the calibration procedure. Therefore, during the program execution, some high-level declarations and definitions are not applicable and it is necessary to perform this procedure with Karel registers functions and instructions. These instructions can be seen in the part of the Karel code presented in Code Script 5.

```
-- Defining Targets
CNV_STR_CONF('N U T, 0, 0, 0',config_var, 0)
GenPos[1] = POS(799.13,0.00,747.89, 180.00,
     48.86, 180.00, config_var) -- P3
SET_PREG_CMT(1, 'P3', SetPosCmt)-- Set comment on
      Position Register
CNV_STR_CONF('N U T, 0, 0, 0',config_var, 0)
GenPos[2] = POS(670.00,0.00,715.00, -180.00,
     35.18, -180.00, config_var) -- P4
SET_PREG_CMT(2, 'P4', SetPosCmt)-- Set comment on
      Position Register
FOR i = 1 TO 4 DO
    -- Read existent Position Registers from TP
    PosReg[i] = GET_POS_REG(i,GetPosStatus)
    -- Check for uninitialized Position Registers
    IF UNINIT(PosReg[i]) = TRUE THEN
    --Create Position Register if Uninitialized
        SET_POS_REG(i, (GenPos[i]),SetPosStatus)
        PosReg[i] = GET_POS_REG(i,GetPosStatus)
    ENDIF
    --Compare Points to check for changes
    IF GenPos[i] >=< PosReg[i] THEN
        -- If point not changed on TP, write
             generated point to Position Register
    ELSE
        -- If point  changed on TP, write changed
              point to Position Register
        GenPos[i] = PosReg[i]
    ENDIF
    SET_POS_REG(i, (GenPos[i]), SetPosStatus)
ENDFOR
```

Code Script 5. Karel register methods

TABLE I
INSTRUCTIONS MAPPED FROM VISUAL COMPONENTS TO THE LANGUAGES USED IN THE ADAPTPACK STUDIO TRANSLATOR LIBRARY

Visual Components	RAPID	Karel	KRL	Inform
Linear Motion	MoveL	WITH $MOTYPE = LINEAR MOVE TO ...	LIN	MOVL
Joint Motion	MoveJ	WITH $JOINT = LINEAR MOVE TO ...	PTP	MOVJ
Define Tool	”Tool Name” := ”tooldata”	”Tool Name”.x = ”value” ”Tool Name”.y = ”value” ”Tool Name”.z = ”value” ”Tool Name”.w = ”value” ”Tool Name”.p = ”value” ”Tool Name”.r = ”value”	TOOL_DATA[”base id”] = ”tool data”	SETTOOL TL#(”tool id”) ”pose”
Define Base	”Base Name” := ”wobjdata”	”Base Name”.x = ”value” ”Base Name”.y = ”value” ”Base Name”.z = ”value” ”Base Name”.w = ”value” ”Base Name”.p = ”value” ”Base Name”.r = ”value”	BASE_DATA[”base id”] = ”base data”	MFRAME UF#(”base id”) ”frame data”
Call Routine	”Routine Name”	”Routine Name”	”Routine Name”()	CALL JOB:”Routine Name”
Assign	”variable” := ”expression”	”variable” = ”expression”	”variable” = ”expression”	SET ”variable code” ”expression”

While Control Loop	WHILE ”condition” DO
	...
	ENDWHILE
		WHILE (”condition”)
		...
		ENDWHILE
			WHILE ”condition” DO
			...
			ENDWHILE
				*”while label”
				...
				JUMP *(while label) IF ”condition”

If/Else Control Loop	IF ”condition” THEN
	...
	ELSE
	...
	ENDIF
		IF (”condition”) THEN
		...
		ELSE
		...
		ENDIF
			IF ”condition” THEN
			...
			ELSE
			...
			ENDIF
				JUMP *(label if) IF ”condition”
				JUMP *(label else)
				*(label if)
				...
				JUMP *(label endif)
				
				*(label else)
				...
				JUMP *(label endif)
				
				*(label endif)

Delay Command	WaitTime ”seconds”	DELAY ”miliseconds”	WAIT SEC ”seconds”	TIMER T=”seconds”
Halt Command	Stop	PAUSE	HALT	-
Comment Statement	! ”comment”	– ”comment”	; ”comment”	’ ”comment”
Print Statement	TPWrite ”text”	WRITE(”text”)	CWRITE(.....,”text”)	MSG ”text”
Wait for Binary Input	WaitDI ”port”, ”state”	WAIT FOR DIN[”port id”] = ”state”	WAIT FOR IN[”port id”]	WAIT IN#(”port id”) = ”state”
Set Binary Output	SetDO ”port”, ”state”	DOUT[”port id”] = ”state”	$OUT[”port id”]	DOUT OT#(”port id”) = ”state”

TABLE II
VARIABLES MAPPED FROM VISUAL COMPONENTS TO THE LANGUAGES USED IN THE ADAPTPACK STUDIO TRANSLATOR LIBRARY

Visual Components	RAPID	Karel	KRL	INFORM*
Integer	VAR num	: INTEGER	DCL INT	IXXX
Double	VAR num	: REAL	DCL REAL	DXXX
Boolean	VAR bool	: BOOLEAN	DCL BOOL	BXXX
String	VAR string	: STRING[”size”]	DCL CHAR[”size”]	SXXX

*The “X” represents an arbitrary ID value.

C. KRL

KRL is the language supported by Kuka [15]. A KRL robot program consists of data and source files. The first one is the file on which are performed all data declarations, such as variables, frames and tools. The second one is where the program routines are defined. The KRL language includes instructions that simplify the code, *e.g* homogeneous transformation operators (represented by the “:” operator). All instructions and variables that have their correspondence in the VC software are presented in Tables I and II, respectively. Different from the other considered languages, KRL presents the Inline Form Programming. This technique, illustrated in Code Script 6, allows the offline programmer to wrap a code to be recognized by the controller teach pendant.

```
;FOLD LIN P4 CONT Vel = 0.35m/s CPDATP4 Tool[1] Base
     [2];%{PE}%R8.3.31,%MKUKATPBASIS,%CMOVE,%VLIN,%P 1:
     LIN, 2:P4, 3:C_DIS, 5:0.35, 7:CPDATP4
....
;ENDFOLD
```

Code Script 6. KRL inline form programming for a motion statement

D. Inform

Unlike the other languages discussed in this section, Inform is a low-level language developed by Yaskawa [16]. For this reason, some statements of the VC software are not in conformity with this language, such as the if-else and while control loops. The mapping of these expressions is achieved through the use of pointers instructions (see Table I). The Inform language already provides some instructions to execute operations among coordinate references; however, it is necessary to set some security parameters on the robot controller for these functions to be effective. The translation and rotation of a base and tool frame are done with the “MFRAME” and the “SETTOOL” instructions, respectively (as can be seen in Table I). Regarding the use of registers, since Inform is a low-level language, the translator needs to map the variable names to register codes, or ID registers. An example of an ID register declaration is shown in Code Script 7. It should be noted that the indexes B, I, D and S are used for Boolean, Integer, Double and String variables, respectively.

```
SET B0000 0
SET B0001 1
SET I0000 0
SET D0001 12.5
SET S0001 "string"
```

Code Script 7. Definition of variables in Inform using ID registers

The base and tool frames are separately defined, each one in a different ".CND" file, as depicted in the Code Scripts 8 and 9, respectively.

```
//UFRAME 1
///NAME UFRAME0
///TOOL 0
///GROUP 1,0,0,0,0,0,0,0
///PULSE
////RORG C000=0,0,0,0,0,0
////RXX C001=0,0,0,0,0,0
////RXY C002=0,0,0,0,0,0
////BUSER 0.00,0.00,0.00,0.00,0.00,0.00
```

Code Script 8. Inform base frame definition file

```
//TOOL 0
///NAME mountplate
0.00,0.00,0.00,0.00,0.00,0.00
0.000,0.000,0.000
0.000
0.000,0.000,0.000
0,0
//TOOL 1
///NAME TOOL0
0.00,0.00,100.00,0.00,0.00,0.00
0.000,0.000,0.000
0.000
0.000,0.000,0.000
0,0
```

Code Script 9. Inform tool frame definition file

E. Discussion

All languages discussed in this section provided support to generate programs based on the interpretation of a high level programming language, such as the programs generated by the VC software. As presented in Tables I and II, all variables and instructions statements are mapped between the VC language and the native robot programming languages. What differs in the translation procedure for each of the considered languages, is the ease of development and the drawbacks in the program post-processing and correspondent download to the robot controller.

Regarding the high-level languages (RAPID, Karel and KRL), these have a good support library and are capable to deal with all programming requirements. One of the main issues that was identified on the Karel language is that the user is required to compile the program before downloading it to the controller, which is a shortcoming to a fast post-process. Concerning the KRL, in the authors opinion, the Inline Form Programming [15] suffers from lack of documentation, making the programming procedure even more time consuming.

For its turn, the Inform, as a low-level programming language, requires more expertise of the programmer. The use of pointers instructions is an alternative to the lack of control loops and decision instructions. The use of ID registers (as presented in Code Script 7) could increase the effort for the development of an automatic translator (that needs to map the variables and its position in the program). Another issue of this language are the drawbacks in the post-process: the support library depends on each particular controller configuration, which delays the procedure of downloading a program and running it in the real robot.

III. AdaptPack Translator Library Structure

The AdaptPack Studio Translator is a dynamic library that incorporates in the Visual Components (VC) software the capability to translate a program developed in this offline programming tool to a specific proprietary code, from the following robot manufacturers: ABB, Fanuc, Kuka, and Yaskawa. This library is part of the AdaptPack Studio. In the AdaptPack Studio solution the user does not need to have deep knowledge on the particular language of the used robot brand, since the programming is developed in the VC language and, mainly, using graphical instructions, as described in [12].

A. Translation Procedure

The schematic presented on Figure 1 illustrates the translation procedure. After the offline programming phase in the VC software, the developer can translate the program by clicking one of the buttons inside the Translators section, as depicted in Figure 2. With the native codes generated, the user can downloaded it to the robot, using the File Transfer Protocol (FTP) protocol or flash disk.

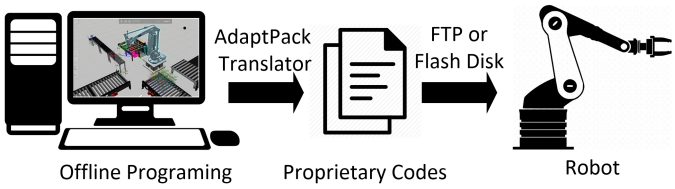

Fig. 1. Translation procedure.

B. Translation Architecture

The translation library is composed of three processing entities: the parser, the translator, and the central code. The parser is responsible for identifying and extracting information, parameters, and instructions from the VC program, and storing it in a data structure. After this step, the translator performs the conversion of this data, and structures the program created according to the language chosen by the user. Therefore, there is a processing step for data acquisition and a translation step for each chosen language. The management and execution of the translation library is performed by the central code. Each one of these entities will be detailed in the sequel.

1) The central code: The central code is executed when the user presses one of the translation buttons, depicted in Figure 2. This code checks if the robot has been selected by the user, identifies the language to be translated, activates the parser, and executes the translation code. The central code also displays an advise window, according to the language chosen, with complementary information for the user concerning the translation procedure. Finally, the generated code is exported and saved in a specific path chosen by the user.

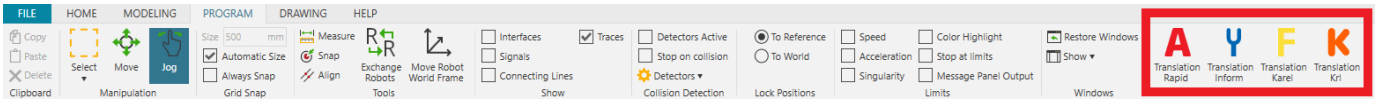

Fig. 2. AdaptPack translator section in VC's program tab.

2) The parser: The implemented parser performs the identification of VC's data and instructions, using the .NET API that implements the Robot Sequence Language (RSL) [17]. Through this library, it is possible to obtain each instruction, subroutine, and variable programmed on the VC program, and analyze it individually. Based on each identification, data structures are created with the parameters that compose it. This data is later used in the translators. The parser uses three main classes: the program class, the variables class, and the instruction class. The program class is relative to program data, such as the main structure and subroutines, path and file name to be created. The variables class identifies and defines the variables used in the program and stores them in a data structure, in the memory. The variables supported by the parser are of the types integer, double, string and boolean. The properties of each variable stored in the parser structure are name and value. The parser also assigns a unique ID to each variable to be used in languages that use the register declarations methods. The instruction class registers the necessary parameters that define a programmed operation instruction. Figure 3 presents a sample code generated in VC. In this case, it is possible to observe examples of instructions that must be translated, such as: waiting for a digital signal, flow control statements, motion order, subroutines call, delay command, assignment of values and signals. Table III presents the instructions supported by the parser, and the properties that compose each instruction. These instructions were chosen according to the necessities of the AdaptPack project [11], [12]. Table IV shows the data types extracted from the instructions that use the target, base and tool frames. The parser also assigns IDs to the targets, base and tool frames, based on the first incidence in the program, in order to be used by languages that use the register declaration approach.

3) The RAPID Translator: The RAPID translator is responsible for examining the sequence of variables and statements stored in the database created by the parser. After, it executes the writing process in a file, with extension ".mod", in the folder specified by the user. During this process, the headers are first written according to the RAPID syntax. Then, the base frames, the tool frames and the target positions (identified by the motion commands) are declared and defined. Base frames and tool frames backups are also created, since the RAPID language changes the records of these frames definitively if any instruction in the code modifies them. In this way, the frames are restored to their original values at the end of the execution process. Subsequently, the translator writes the main routine, with its scope, and the subroutines in the ".mod" file specified by the user. The subroutines are defined in the same file of the main routine. The variables are declared and defined

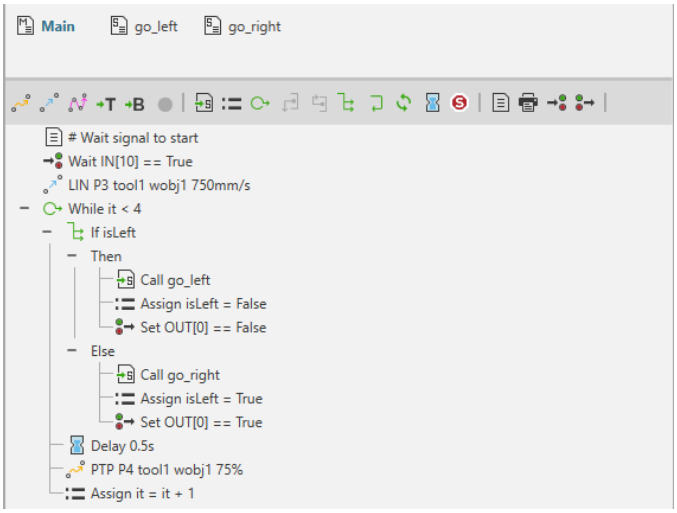

Fig. 3. VC example code.

TABLE III
PARSER SUPPORTED INSTRUCTIONS

Instructions	Properties
Comment	Text
Wait BIN	Port and digital binary value
Delay	Time
Linear Motion	Target*, speed, base and tool frames*
Point to Point Motion	Target*, speed, base and tool frames*
Set BIN	Port and digital value
While	Condition and scope
If/Else	Condition and scope
Call Routine	Routine's name
Assign	Operators
SetBase	Base's name, ID and pose
SetTool	Tool's name, ID and pose
Print	Text
Break	-
Continue	-

*See Table IV.

TABLE IV
TARGET, BASE AND TOOL FRAMES SUPPORTED BY THE PARSER

Data	Properties
Target	Target's name, pose, joints and configuration
Base Frame	Base frame's name, ID and pose
Tool Frame	Tool frame's name, ID and pose

within each scope in which they are used.

4) The Karel Translator: As in the RAPID case, the Karel translator is responsible for reading the sequence of variables and statements stored in the parser database. Afterwards, it executes the writing process in a file with extension ".kl", in the folder specified by the user. Headers are initially written according to the Karel syntax. The base frames, tool frames and target positions (identified by the motion commands) are declared and defined, and the variables are also declared. All subroutines are defined, and have their scope transcribed so that later the main scope is created. In this scope are inserted the definitions of the necessary frames and variables, besides the instructions that constitute the program. It should be noted that the Karel translator performs the declaration and definition of target positions using registers (Code 5). This allows users to perform modifications and adjustments, using the robot teach pendant, without the need to translate a new code. These records have an index according to the first incidence of the data in the code created in VC.

5) The KRL Translator: The statement and variable sequences, wrapped by the parser, are analyzed by the KRL translator, as the user chooses this language. The writing process is done in two distinct files: a file with extension ".dat" and another with the extension ".src". The ".dat" file is where program data is declared and defined, while the ".src" file is where the program is defined.

The KRL translator initially creates the ".dat" file by setting its header. It then defines and declares the base and tool frames used, in addition to the targets, identified by the motion commands. It also includes the declaration and definition of the properties that constitute these motions. The KRL translator defines the base and tool frames using registers; this way, the translations are done based on the first incidence of this data in the VC's program. The KRL translator creates a file with the extension ".src", starting with the header, according to the KRL syntax. The main scope of the code is defined and transcribed, and then each subroutine is defined and written to the same file. The Inline Form Documentation method is also created (Code 6), thus the KRL code instructions are exposed in the robot's teach pendant.

6) The Inform Translator: As in the previous translators, the one for Inform identifies and writes all the instructions and variables parsed. In this case, at least three files are generated and saved in a specified folder defined by the user: one ".JBI" file for the main code, one ".JBI" file for each subroutine (if it exists) and two other files for base and tool frames declaration and definition, both with the ".CND" extension.

At the beginning of the translation the file that declares and defines the base frames is generated, followed by the tool frames file. The frame registers, in both files, are set with the ID of each one, based on the first incidence in the VC's program. The main code and the subroutines are created after. Each code has its own header, following the Inform syntax, that first defines all the target registers used in the scope. After this, the instructions are written. As stated in Section II-D and Table I, all "while" and "if" statements are identified and translated by pointers instructions.

IV. Experimental Results

Test codes generated by the AdaptPack Studio Translator were submitted, to both, simulated and real scenarios. A robot program was developed offline, in the VC software, and was performed its translation to robot native code. Regarding the simulation phase, the simulators used were: ABB RobotStudio (Figure 4), Fanuc Roboguide (Figure 5) and Yaskawa MotosimEG.The robots used in the tests were the ABB IRB 2600-20/1.6, Fanuc LR Mate 200iD 7L, and Motoman YR-HP6-B10/NX100. Concerning the KRL language, the tests in the KUKA.Sim simulator and the real robot will be done as future work.

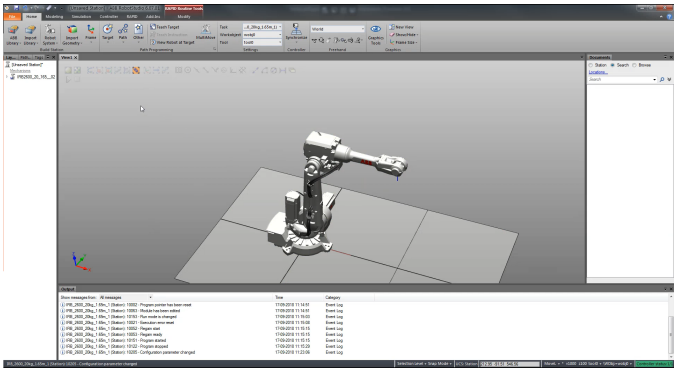

Fig. 4. The ABB RobotStudio simulator.

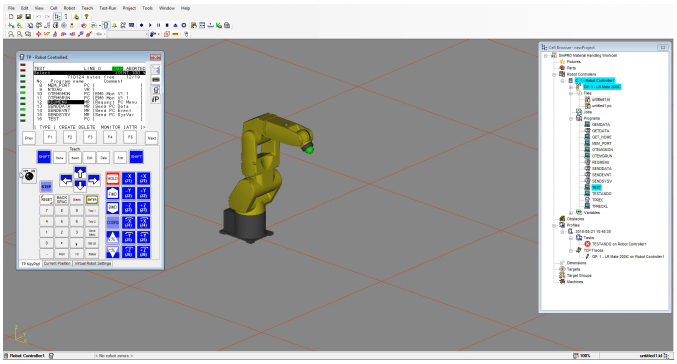

Fig. 5. The Fanuc Roboguide simulator.

Since, the test program codes are extensive to be included in the paper, parts of it based on the VC program depicted on Figure 3 are showed in Code Scripts 10,11 and 12.

```
25        ! Wait signal to start
26        WaitDI di10,high;
27        MoveL P3,v600,fine,tool1\WObj:=wobj1;
28        WHILE it < 4 DO
29          IF isLeft THEN
30            go_left;
31            isLeft:=False;
32            SetDO do0,low;
33          ELSE
34            go_right;
35            isLeft:=True;
36            SetDO do0,high;
```

```
37          ENDIF
38          WaitTime 0.5; ! seconds
39          MoveJ P4, vmax \V := vmax.v_tcp*0.75,fine,tool1
                \WObj:=wobj1;
40          it:=it + 1;
41        ENDWHILE
```

Code Script 10. Rapid sample code

```
24' Wait signal to start
25WAIT IN#(101)=ON
26MOVL P0000 V=750 PL=0  //P3
27*0
28' BEGIN WHILE
29JUMP *1 IF B0000
30'GO TO IF
31JUMP *2
32' GO TO ELSE
33*1
34' BEGIN IFTHEN
35CALL JOB:go_left
36SET B0000 0
37DOUT OT#(01)=OFF
38JUMP *3
39' END IFTHEN
40*2
41' BEGIN ELSETHEN
42CALL JOB:go_right
43SET B0000 1
44DOUT OT#(01)=ON
45JUMP *3
46' END ELSETHEN
47*3
48' END IFELSE STRUCT
49'Timer in seconds
50TIMER T=0.5
51MOVJ P0001 VJ=75 PL=0  //P4
52SET I0000 I0000 + 1
53JUMP *0 IF I0000 < 4
54' END WHILE
```

Code Script 11. Inform sample code

```
70        --Wait signal to start
71        WAIT FOR DIN[11] = ON
72        WITH $TERMTYPE = FINE, $MOTYPE = LINEAR,  $SPEED
              = 750, $UFRAME = Uframe1, $UTOOL = Utool1
              MOVE TO GenPos[1]
73        WHILE (it < 4) DO
74          IF (isLeft) THEN
75            go_left
76            isLeft = False
77            DOUT[1] = OFF
78          ELSE
79            go_right
80            isLeft = True
81            DOUT[1] = ON
82          ENDIF
83          DELAY 500 -- ms
84           WITH $TERMTYPE = FINE, $MOTYPE = JOINT,
                $SPEED = 0.75*$PARAM_GROUP[1].
                $SPEEDLIMJNT, $UFRAME = Uframe1, $UTOOL =
                 Utool1 MOVE TO GenPos[2]
85          it = it + 1
86        ENDWHILE
```

Code Script 12. Karel sample code

After the translation process, all simulations were performed as expected. All the programming done in VC was reproduced in the native simulators. No difference between the simulations, performed on the robot brand simulators, and the real scenarios were found. This way, the practical tests were performed according to the VC's software programming.

All programs generated by the AdaptPack Studio Translator were able to accept modifications through the teach pendant. Therefore, the process of calibration can be performed, if there are differences in the environment between the simulated and the real-world scenario.

V. Conclusions

This paper presented the AdaptPack Studio Translator, a library to translate programs developed offline in Visual Components to native robot languages. The brands considered in this project were ABB, FANUC, Kuka and Yaskawa. It was also presented a comparative evaluation between each language, regarding the development of the translator.

The performed tests, either in the simulated and real scenarios, allowed to verify the capability and the practicality of the proposed work. This improves the post-process in offline programming and helps engineers in the project development. Furthermore, this methodology is able to handle the industries demands of fast and flexible solutions.

As future work extensive tests will be performed using different robots from distinct brands.

References

[1] M. H. Choi and W. W. Lee, "A force/moment sensor for intuitive robot teaching application," in *IEEE International Conference on Robotics and Automation (ICRA).*, vol. 4. IEEE, 2001, pp. 4011–4016.

[2] S. Sugita, T. Itaya, and Y. Takeuchi, "Development of robot teaching support devices to automate deburring and finishing works in casting," *The International Journal of Advanced Manufacturing Technology*, vol. 23, no. 3-4, pp. 183–189, 2004.

[3] R. D. Schraft and C. Meyer, "The need for an intuitive teaching method for small and medium enterprises," *VDI BERICHTE*, vol. 1956, p. 95, 2006.

[4] M. Ferreira, P. Costa, L. Rocha, and A. Paulo Moreira, "Stereo-based real-time 6-DoF work tool tracking for robot programing by demonstration," *International Journal of advanced manufacturing technology*, vol. 85, no. 1-4, pp. 57–69, 2016, citations: crossref, scopus, wos.

[5] Z. Pan, J. Polden, N. Larkin, S. Van Duin, and J. Norrish, "Recent progress on programming methods for industrial robots," in *41st International Symposium on Robotics (ISR) and 6th German Conference on Robotics (ROBOTIK)*. VDE, 2010, pp. 1–8.

[6] *Industrial Robot Language (IRL) - DIN Standard 66312*, Deutsche Norm Std., 1996.

[7] V. S. Bottazzi and J. F. C. Fonseca, "Off-line robot programming framework," in *Joint International Conference on Autonomic and Autonomous Systems and International Conference on Networking and Services - (icas-isns'05)*, Oct 2005, pp. 71–71.

[8] M. Bruccoleri, C. D'onofrio, and U. La Commare, "Off-line programming and simulation for automatic robot control software generation," in *5th IEEE Int. Conf. on Industrial Informatics*, vol. 1, 2007, pp. 491–496.

[9] E. Freund, B. Ludemann-Ravit, O. Stern, and T. Koch, "Creating the architecture of a translator framework for robot programming languages," in *IEEE Int. Conf. on Rob. and Aut.*, vol. 1. IEEE, 2001, pp. 187–192.

[10] V. Components, "Visual components," WebSite, 2018. [Online]. Available: https://www.visualcomponents.com/

[11] R. Silva, L. F. Rocha, P. Relvas, P. Costa, and M. F. Silva, "Offline programming of collision free trajectories for palletizing robots," in *Iberian Robotics conference*. Springer, 2017, pp. 680–691.

[12] A. Castro, J. P. Souza, L. Rocha, and M. F. Silva, "Adaptpack studio: Automatic offline robot programming framework for factory environments," in *19th IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC 2019)*, vol. 1. IEEE, 2019.

[13] ABB, *Technical reference manual RAPID Instructions, Functions and Data types*.

[14] I. FANUC Robotics America, *FANUC Robotics SYSTEM R-J3iB Controller KAREL Reference Manual*, 2003.

[15] K. Roboter, *Quickguide KRL-Syntax*, 2012.

[16] Y. Motoman, *DX100 Options Instructions for Inform Language*.

[17] V. Components, *3D Simulation Software - Quick Start Guide 3.1*, 3rd ed., Visual Components, December 2004.