

LEAVING PL/I

CONSIDERATIONS FOR MOVING PL/I CODE TO OTHER LANGUAGES

At a Glance

PL/I is a very powerful though outdated programming language. “Powerful” in terms of the language features, its concise way of coding and the functions which go beyond those of nowadays’ compilers; “outdated” because of functions such as direct memory addressing, jumps, a built-in macro processor and many more which are purposefully avoided by modern languages.

There is no one-and-only way to modernize PL/I applications! Based on the possibilities of its RULAMAN® product range and modern programming language paradigms, this paper aids in choosing the best modernisation path which depends – not surprisingly – on several technical and strategic factors.

LANGUAGE BACKGROUND

PL/I was created in the 1960s at the same time as COBOL. With PL/I, IBM intended to surpass COBOL which is standardized by the ANSI committee ever since.

COBOL was designed as a language for commercial purposes like data presentation on paper, handling of files, sorting and simple calculations. It was created at a time when computer science did not exist and the rapidly growing demand for programmers needed to be satisfied with untrained non-computer people.

PL/I on the other hand tried to combine some advantages of the mainframe assembler and mathematical thinking. Its nature is procedural like COBOL, but many features have been incorporated from ALGOL and other languages of that time. PL/I requires programmers with a much higher level of abstraction.

From a functional perspective, PL/I is a superset of COBOL. In real life, however, programmers often use only a small set of the available functions. Thus many applications might as well have been written in COBOL.

AUTOMATED LANGUAGE TRANSFORMATION

IT Modernisation offers language transformation products which allow for fully automated conversion virtually from any language to any other.

APPLICATION DEVELOPMENT

If you need to determine the best target language when it comes to converting PL/I applications you should answer the following questions:

- Is this application static or do you expect lots of maintenance and extension?
- Will you need the macro possibilities in future?
 - What is your future programming language?
 - On which operating system will the code be executed?
 - Which range of PL/I functions are actually used? (*see next page*)
 - Will there be COBOL skill around – for the time this application lives?

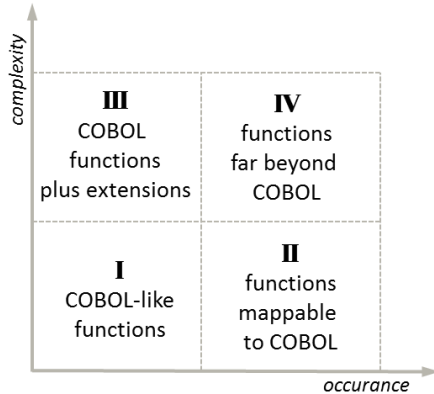
IS RE-WRITING AN OPTION?

Definitely not!

Re-writing is 5 to 10 times more expensive than transforming it using language transformation products like RULAMAN®. Furthermore you must consider that you won’t have the program specifications at hand and – any new code will inevitably have new errors.

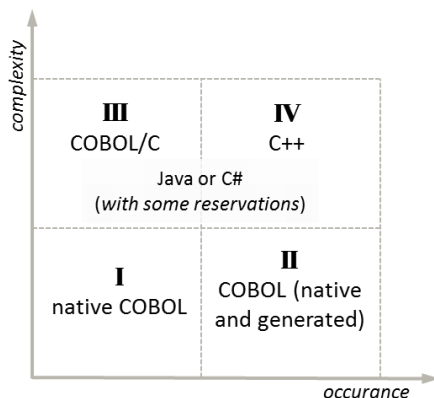
THE PL/I QUADRANTS

By analysing your PL/I code you will be able to determine which of the four quadrants in the graph below describes your application best.



Many PL/I programs will fit into quadrants I and II to a large extent. If COBOL is set as one of your strategic languages, modernize towards COBOL!

Here are possible the target languages:



- II COBOL subroutines to emulate PL/I functions will be generated
- III COBOL Code must be heavily extended by C or C++ routines
- IV C++ will be able to map the required functions. Its class libraries save a lot of effort

If functions like memory addressing, jumps, pre-compiler features etc. were not used, it is possible to convert the PL/I code to an object-oriented language like Java or C#.

DETERMINING THE PL/I QUADRANT

The table lists functions which point to quadrants II to IV. Functions not listed are all part of quadrant I.

II	III	IV
Data Types		
float, single bits, wide chars (WX), DBCS, array, concatenate WX	Varying(z), format, CEIL	Complex, concatenate mixed (DBCS, SBSCS, WX, GX)
Load & invoke		
INONLY, OUTONLY, FROMALIEN, DLLINTERNAL	FETCH, RELEASE, SET, LIST, GENERIC, BUILTIN	
Type Definitions		
Typed structures ("."-reference)	DEFINE: ALIAS, ORDINAL, STRUCTURE; HANDLE, BIND, CAST, FIRST, LAST, NEW, RESPEC, SIZE	
Data Structuring		
DEFAULTVALUE, UNION	DEFAULT RANGE, BASED, LIKE, ALIGNED	
Storage Control		
REFER	ALLOCATE, AREA, BIG-/LITTLEENDIAN, OFFSET, NONCONNECTED	NON-ASSIGNABLE
File I/O		
STREAM files	STRING option	edit directed
Exception handling		
RECORD, SUB-SCRIPTRANGE	condition prefix	ATTENTION, CONVERSION, AREA, NAME, INVALIDOP, SIZE, STORAGE, STRINGRANGE, STRINGSIZE
Other		
	preprocessor usage, use of built-in functions (ABS ... Y4YEAR)	multithreading

ANALYSE THOROUGHLY!

Before deciding on your modernisation road you should take the time to analyse your application from strategic, technical and economic aspects. The right decision can save you a lot of time and money.

IT Modernisation has the proper tools and ample experience to assist you with that analysis.