15

Choosing Appropriate Programming Language to Implement Software for Real-Time Resource-Constrained Embedded Systems

Mouaaz Nahas[1] and Adi Maaita[2]
[1]*Department of Electrical Engineering, College of Engineering and Islamic Architecture, Umm Al-Qura University, Makkah,*
[2]*Software Engineering Department, Faculty of Information Technology, Isra University, Amman,*
[1]*Saudi Arabia*
[2]*Jordan*

1. Introduction

In embedded systems development, engineers are concerned with both software and hardware aspects of the system. Once the design specifications of a system are clearly defined and converted into appropriate design elements, the system implementation process can take place by translating those designs into software and hardware components. People working on the development of embedded systems are often concerned with the software implementation of the system in which the system specifications are converted into an executable system (Sommerville, 2007; Koch, 1999). For example, Koch interpreted the implementation of a system as the way in which the software program is arranged to meet the system specifications.

Having decided on the software architecture of the embedded design, the first key decision to be made in the implementation stage is the choice of programming language to implement the embedded software (including the scheduler code, for example). The choice of programming language is an important design consideration as it plays a significant role in reducing the total development time (Grogono, 1999) (as well as the complexity and thus maintainability and expandability of the software).

This chapter is intended to be a useful reference on "computer programming languages" in general and on "embedded programming languages" in particular. The chapter provides a review of (almost) all common programming languages used in computer science and real-time embedded systems. The chapter then discusses the key challenges faced by an embedded systems developer to select a suitable programming language for their design and provides a detailed comparison between the available languages. A detailed literature review of the work done in this area is also provided. The chapter also provides real data which shows that – among the wide range of available choices – "C" remains the most popular language for use in the programming of real-time, resource-constrained embedded systems. The key features of "C" which made it so popular are provided in a great detail.

The chapter is organized as follows. Section 2 provides various definitions of the term "programming language" from a wide range of well-known references. Section 3 and Section 4 provide classification and history of programming languages (respectively). Section 5 provides a review of programming languages used in the fields of real-time embedded systems. Section 6 discusses the choice of programming languages for embedded designs. Section 7 and Section 8 provide the main advantages of "C" which made it the most popular language to use in real-time, resource-constrained embedded systems and a detailed comparison with alternative languages (respectively). Real data which shows the prevalence of "C" against other available languages is also provided in Section 8. Section 9 presents a brief literature review of using "C" to implement software for real-time embedded systems. The overall chapter conclusions are drawn in Section 10.

2. What is a programming language?

Simply, programming as a problem has only arisen since computer machines were first created. The magnitude of the problem is however relative to the size (and complexity) of the computer machine used (Cook, 1999). To program a computer system, a programming language is required. The latter is seen as the major way of communication (interface) between a person who has a problem and the computer system used to solve the problem.

Programming language has been defined in several ways. For example, American Standard Vocabulary for Information Processing (ANSVIP, 1970) defined a programming language as "A language used to prepare computer programs". The IFIP-ICC Vocabulary of Information Processing (IFIP-ICC, 1966) defined it as "A general term for a defined set of symbolic and rules or conventions governing the manner and sequence in which the symbols may be combined into a meaningful communication". The IFIP-ICC glossary also noted that "An unambiguous language, intended for expressing programs, is called a PROGRAMMING LANGUAGE". Other definitions for a programming language include:

- "A computer tool that allows a programmer to write commands in a format that is more easily understood or remembered by a person, and in such a way that they can be translated into codes that the computer can understand and execute." (Budlong, 1999).
- "An artificial language for expressing programs." (ISO, 2001).
- "A self-consistent notation for the precise description of computer programs" (Wizitt, 2001).
- "A standard which specifies how (sort of) human readable text is run on a computer." (Sanders, 2007).
- "A precise artificial language for writing programs which can be automatically translated into machine language." (Holyer, 2008).

However, it was noted elsewhere (e.g. Sammet, 1969) that standard definitions are usually too general as they do not reflect the language usage. A more specific definition for a programming language was given by Sammet as a set of characters and rules (used to combine the characters) that have the following characteristics:

- A programming language requires no knowledge of the machine code by the programmer, thus the programmer can write a program without much knowledge about the physical characteristics of the machine on which the program is to be run.

- A programming language should be machine independent.
- When a program written in a programming language is translated into the machine code, each statement should explode to generate a large set of machine instructions.
- A programming language must have problem-oriented notations which are closer to the specific problem intended to be solved.

It is worth mentioning that a vast number of different programming languages have already been created, and new languages are still being created.

3. Classification of programming languages

This section provides a classification of programming languages. Sources for this section include (Sammet, 1969; Booch, 1991; Grogono, 1999; Lambert & Osborne, 2000; Mitchell, 2003; Calgary, 2005; Davidgould, 2008; Network Dictionary, 2008).

In general, programming languages can be divided into programming paradigms and classified by their intended domain of use. Paradigms include procedural programming, object-oriented (O-O) programming, functional programming, and logic programming. Note that some languages combine multiple paradigms. Each of these paradigms is briefly introduced here.

Procedural programming (or imperative programming) is based on the concept of decomposing the program into a set of procedures (i.e. series of computational steps). Examples of procedural languages are: FORTRAN (**FOR**mula **TRAN**slator), Algol (**ALGO**rithmic **L**anguage), COBOL (**CO**mmon **B**usiness **O**riented **L**anguage), PL/I (**P**rogramming **L**anguage **I**), Pascal, BASIC (**B**eginner's **A**ll-purpose **S**ymbolic **I**nstruction **C**ode), Modula-2, "C" and Ada. Object-Oriented (O-O) programming is a method where the program is organized as cooperative collections of "objects". This style of programming was not commonly used in software application development until the early 1990s, but nowadays most of the modern programming languages support this type of programming paradigm. Examples of object-oriented languages are: Simula, Smaltalk, C++, Eiffel and Java. Functional programming treats computation as the evaluation of mathematical functions. In functional programming, a high order function can take another function as a parameter or returns a function. An example of functional languages is LISP (**LIS**t **P**rocessor). Finally, logic programming uses mathematical logic in which the program enables the computer to reason logically. An example of logic languages is Prolog (**PRO**gramming in **LOG**ic). It is often argued that languages with support for an O-O programming style have advantages over those from earlier generations (Pont, 2003). For example, Jalote (1997) noted that using O-O helps to represent the problem domain, which makes it easier to produce and understand designs.

In addition to programming paradigm, the purpose of use is an important characteristic of a language: it is unlikely to see one language fitting all needs for all purposes (Sammet, 1969). Programming languages can be divided, according to their purpose, into general-purpose languages, system programming languages, scripting languages, domain-specific languages, and concurrent / distributed languages (or a combination of these). A general-purpose language is a type of programming language that is capable of creating various types of programs for various applications, e.g. "C" language. There has been an argument that some of the general-purpose languages were designed mainly for educational purposes

(Wirth, 1993). A system programming language is a language used to produce software which services the computer hardware rather than the user, e.g. Assembly and Embedded C. Scripting language is a language in which programs are a series of commands that are interpreted and then executed sequentially at run-time without compilation, e.g. JavaScript (used for web page design). Domain-specific programming languages are, in contrast to general-purpose languages, designed for a specific kind of tasks, e.g. Csound (used to create audio files), and GraphViz (used to create visual representations of directed graphs). Concurrent languages are programming languages that have abstractions for writing concurrent programs. A concurrent program is the program that can execute multiple tasks simultaneously, where these tasks can be in the form of separate programs or a set of processes or threads created by a single program. Concurrent programming can support distributed computing, message passing or shared resources. Examples of concurrent programming languages include Java, Eiffel and Ada.

In his famous book (i.e. "Programming Languages: History and Fundamentals", 1969), Jean E. Sammet used the following set of defining categories as a way of classifying programming languages: 1) procedural and non-procedural languages; 2) problem-oriented, application-oriented and special purpose languages; 3) problem-defining, problem describing and problem solving languages; 4) hardware, publication and reference languages. Sammet however underlined that any programming language can fall into more than one of these categories simultaneously: for further details see Sammet (1969).

4. History of programming languages

It has been argued that studying the history of programming languages is essential as it helps developers avoid previously-committed mistakes in the development of new languages (Wilson & Clark, 2000). It was also pointed out that an unfortunate trend in Computer Science is creating new language features without carefully studying previous work in this field (Grogono, 1999). Most books and articles on the history of programming languages tend to discuss languages in terms of generations where languages are classified by age (Cook, 1999). Many articles and books have discussed the generations of programming languages (e.g. Wexelblat, 1981; Martin & Leben, 1986; Watson, 1989; Zuse, 1995; Flynn, 2001). Pont (2003) provides a list of widely-used programming languages classified according to their generations (see Table 1).

Language generation	Example languages
-	Machine code
First generation language (1GL)	Assembly
Second generation languages (2GL)	COBOL, FORTRAN
Third generation languages (3GL) "process-oriented'	C, Pascal, Ada 83
Fourth generation languages (4GL) 'object-oriented'	C++, Java, Ada 95

Table 1. Classification of programming languages by generations (Pont, 2003).

A brief history of the most popular programming languages (including the ones presented in Table 1) is provided in this section. Sources for the following material mainly include (Wexelblat, 1981; Martin & Leben, 1986; Watson, 1989; Halang & Stoyenko, 1990; Grogono, 1999; Flynn, 2001).

In the 1940s, the first electrically powered digital computers were created. The computers of the early 1950s used machine language which was quickly superseded by a second generation of programming languages known as Assembly languages. The limitations in resources (e.g. computer speed and memory space) enforced programmers to write their hand-tuned assembly programs. However, it was shortly realized that programming in assembly required a great deal of intellectual effort and was prone to error. It is important to note that although many people consider Assembly as a standard programming language, some others believe it is too low-level to bring satisfactory of communication for user, hence was excluded from the programming languages list (Sammet, 1969).

1950s saw the development of a range of high-level programming languages (some of which are still in widespread use), e.g. FORTRAN, LISP, and COBOL, and other languages such as Algol 60 that had a substantial influence on most of the lately developed programming languages. In 1960s, languages such as APL (**A** **P**rogramming **L**anguage), Simula, BASIC and PL/I were developed. PL/I incorporated the best ideas from FORTRAN and COBOL. Simula is considered to be the first language designed to support O-O programming.

The period between late 1960s and late 1970s brought a great prosperity to programming languages most of which are used nowadays. In the mid-1970s, Smalltalk was introduced with a complete design of an O-O language. The programming language "C" was developed between 1969 and 1973 as a systems programming language, and remained popular. In 1972, Prolog was designed as the first logic programming language. In 1978, ML (**M**eta-**L**anguage) was developed to found statically-typed functional programming languages in which type checking is performed during compile-time allowing more efficient program execution. It is important to highlight that each of these languages originated an entire family of descendants. Some other key languages which were developed in this period include: Pascal, Forth and SQL (**S**tructured **Q**uery **L**anguage).

In 1980s, C++ was developed as a combined O-O and systems programming language. Around the same time, Ada was developed and standardized by the United States government as a systems programming language intended for use in defense systems. One noticeable tendency of language design during the 1980s was the increased focus on programming large-scale systems through the use of modules, or large-scale organizational units of code. Therefore, languages such as Modula-2, Ada, and ML were all extended to support such modular programming in 1980s. Some other languages that were developed in this period include: Eiffel, PEARL (**P**ractical **E**xtraction and **R**eport **L**anguage) and FL (**F**unction **L**evel).

In mid-1990s, the rapid growth of the Internet created opportunities for new languages to emerge. For example, PEARL (which is originally a Unix scripting tool first released in 1987) became widely adopted in dynamic web sites design. Another example is Java which was commonly used in server-side programming. These language developments provided no fundamental novelty: instead, they were modified versions of existing languages and paradigms and largely based on the "C" family of programming languages.

It is difficult to determine which programming languages are most widely used, as there have been various ways to measure language popularity (see O'Reilly, 2006; Bieman & Murdock, 2001). Mostly, languages tend to be popular in particular types of applications. For example, COBOL is a leading language in business applications (Carr & Kizior, 2000),

FORTRAN is widely used in engineering and science applications (Chapman, 2004), and "C" is a genuine language for programming embedded applications and operating systems (Barr, 1999; Pont, 2002; Liberty & Jones, 2004).

5. Programming languages for real-time embedded systems

To develop a real-time embedded system, a number of tools and techniques would be required: the key one is the programming language used to develop the application code (Burns, 2006). Assembly was the first programming language used to implement the software for embedded applications. However, it was argued that the development environments that used the first generation languages such as Assembly lacked the basic support for debugging and testing (Halang & Stoyenko, 1990). Therefore, in 1960s, the need for high-level programming languages to program real-time systems, instead of continuing to use Assembly language, was agreed among many real-time system designers; due to advantages such as ease of learning, programming, understanding, debugging, maintaining and documenting and also code portability (see Boulton & Reid, 1969; Sammet, 1969).

The work in this area began by identifying the essential requirements for a high-level language to fulfill the objectives of real-time applications (Opler, 1966). Such requirements were summarized by Boulton & Reid (1969) as methods of handling real-time signals and interrupts, and methods of scheduling real-time tasks. Opler (1966) argued that to achieve such requirements, one can make extensions / modifications to an existing programming language, where an alternative solution is to develop new languages dedicated specifically for real-time software. Some success, in extending existing languages to real-time computing, was achieved using languages such as FORTRAN (e.g. Jarvis, 1968; Roberts, 1968; Hohmeyer, 1968; Mensh & Diehl, 1968; Kircher & Turner, 1968) and PL/I (e.g. Boulton & Reid, 1969). Some other studies, however, attempted to develop new real-time languages but with some similarity to existing languages, e.g. PROSPRO (Bates, 1968), SPL (Oerter, 1968) and RTL (Schoeffler & Temple, 1970).

In 1970s, a major concern of many researchers became the programming of real-time applications which involve concurrent processing. Useful work in this area demonstrated that, same as before, concurrent programming can be achieved by either extending available general-purpose languages (e.g. Hansen, 1975; Wirth, 1977) or developing entirely new concurrent-processing languages (e.g. Schutz, 1979). However, it was noticed that extended general-purpose languages still lacked genuine concurrency and real-time concepts (Steusloff, 1984). This led to the development of more efficient concurrent real-time languages such as PEARL (DIN, 1979), ILIAD (Schutz, 1979) and Ada (Ada, 1980).

Ada is a well-designed and widely used language for implementing real-time systems (Burns, 2006). Therefore, it is worth discussing it in greater detail. As previously noted, Ada is an object-oriented, high-level programming language which was first developed and adopted by the U.S. Department of Defense (DoD) to implement various defense mission-critical software applications (Ada, 1980; Baker & Shaw, 1989). Ada appeared as a standard language in 1983 – when Ada83 was released – and was later reviewed and improved in 1995 by producing Ada95. Since developed, Ada has gained a great deal of interest by many real-time and embedded systems developers (e.g. see Real-Time Systems (RTS) Group webpage, The University of York, UK). It was declared that Ada embodies features which

facilitate the achievement of safety, reliability and predictability in the system behavior (Halang & Stoyenko, 1990). Halang & Stoyenko (1990) carried out a detailed survey on a number of representative real-time programming languages including Ada, FORTRAN, HALL/S, LTR, PEARL, PL/I and Euclid, and concluded that Ada and PEARL were the most widely available and used languages among the others which had been surveyed.

In addition to the previous sets of modified and specialized real-time languages, it was accepted that universal, procedural programming languages (such as C) can also be used for real-time programming although they contain just rudimentary real-time features: this is mainly because such languages are more popular and widely available than genuine real-time languages (Halang & Stoyenko, 1990). Later generations of O-O languages such as C++ and Java also have popularity in embedded programming (Fisher et al., 2004). Embedded versions of famous ".Net" languages are gaining more popularity in the field of embedded systems development. However, they are not a favorite choice when it comes to resource constrained embedded systems as they are O-O languages, hence, they require a lot of resources as compared to the requirements of "C".

6. Choosing a suitable programming language for embedded design

In real-time embedded systems development, the choice of programming language is an important design consideration since it plays a significant role in reducing the total development time (Grogono, 1999).

Overall, it has been widely accepted that the low-level Assembly language suffers high development costs and lack of code portability, and only very few highly-skilled Assembly programmers can be found today (see Barr, 1999; Walls, 2005). If the decision is therefore made not to use the Assembly language due to its inevitable drawbacks, there is no scientific way to select the most optimal high-level programming language for a particular application (Sammet, 1969; Pont, 2002). Instead, researchers tend to discuss the important factors which should be considered in the choice of a language. For example, Sammet (1969) indicated that a major factor in selecting a language is the language suitability to solve the particular classes of problems for which it is intended, and the type of the actual user (i.e. user level of professionalism). It has also been noted by Sammet that factors such as availability on the desired computer hardware, history and previous evaluation, implementation consequences of the language are also key factors to take into account during the language selection process. However, Sammet stressed that a successful choice can only be made if the language includes the required technical features.

Specifically, when choosing a language for embedded systems development, the following factors must be considered (Pont, 2003):

- Embedded processors normally have limited speed and memory, therefore the language used must be efficient to meet the system resource constraints.
- Programming embedded systems requires a low-level access to the hardware. For example, there might be a need to read from / write to particular memory locations. Such actions require appropriate accessing mechanisms, e.g. pointers.
- The language must support the creation of flexible libraries, making it easy to re-use code components in various projects. It is also important that the developed software

should be easily ported and adapted to work on different processors with minimal changes.
- The language must be widely used in order to ensure that the developer can continue to recruit experienced professional programmers, and to guarantee that the existing programmers can have access to information sources (such as books, manuals, websites) for examples of good design and programming practices.

Of course, there is no perfect choice of programming language. However, the chosen language is required to be well-defined, efficient, supports low-level access to hardware, and available for the platform on which it is intended to be used. Against all of these factors, "C" language scores well, hence it turns out to be the most appropriate language to implement software for low-cost resource-constrained embedded systems. Pont (2003) stated that *"C's strengths for embedded system greatly outweigh its weaknesses. It may not be an ideal language for developing embedded systems, but it is unlikely that a 'perfect' language will be created"*.

7. The "C" programming language

In his famous book "Programming Embedded Systems in "C" and C++", Michael Barr (1999) emphasized that "C" language has been a constant factor across all embedded software development due to the following advantages:

- It is small and easy to learn.
- Its compilers are available for almost every processor in use today.
- There are so many experienced "C" programmers around the world.
- It is a hardware-independent programming language, a feature which allows the programmer to concentrate only on the algorithm rather than on the architecture of the processor on which the program will be running.

Despite this, Barr highlighted that the key advantage of "C" which made it the favorite choice for many embedded programmers is its low-level nature that provides the programmer with the ability to interact easily with the underlying hardware without sacrificing the benefits of using high-level programming.

In (Grogono, 1999), it was declared that "C" is based on a small number of primitive concepts, therefore it is an easy language to learn and program by both skilled and unskilled programmers. Moreover, Grogono stated that "C" can be easily compiled to produce efficient object code.

In a more recent publication, Pont (2002) stated that *"C's strengths for embedded system greatly outweigh its weaknesses. It may not be an ideal language for developing embedded systems, but it is unlikely that a 'perfect' language will be created"*. According to (Pont, 2002, 2003), the key features of the "C" language can be summarized as follows.

- It is a mid-level language with both high-level features (such as support for functions and modules) and low-level features (such as access to hardware via pointers).
- It is very efficient, popular and well understood even by desktop developers who programmed on C++ or Java.
- It has well-proven compilers available nowadays for every embedded processor (e.g. 8-, 16-, 32-bit or more).

- Books, training courses, code examples and websites that discuss the use of the language are all widely available.

In (Jones, 2002), it was noted that features such as easy access to hardware, low memory requirements, and efficient run-time performance make the "C" language popular and foremost among other languages. In (Brosgol, 2003), it was made clear that "C" is the typical choice for programming embedded applications as it is processor-independent, has low-level features, can be implemented on any architecture, has reasonable run-time performance, is an international standard, and is familiar to almost all embedded systems programmers. Fisher et al. (2004) emphasized that, in addition to portability and low-level features of the language, C structured programming drives embedded programmers to choose "C" language for their designs. Moreover, it has been clearly noted that "C" cannot be competed in producing a compact, efficient code for almost all processors used today (Ciocarlie & Simon, 2007).

Furthermore, since "C" was recognized as the de facto language for coding embedded systems including those which are safety-related (Jones, 2002; Pont, 2002; Walls, 2005), there have been attempts to make "C" a standard language for such applications by improving its safety characteristics rather than promoting the use of safer languages that are less popular (such as Ada). For example, The UK-based Motor Industry Software Reliability Association (MISRA) has produced a set of guidelines (and rules) for the use of "C" language in safety-critical software: such guidelines are well known as "MISRA C". For more details, see (Jones, 2002).

8. Why does "C" outperform other languages?

When comparing "C" to other alternative languages such as C++ or Ada, the following observations have been made. C++ is a good alternative to "C" as it provides better data abstraction and offers a better O-O programming style, but some of its features may cause degradation in program efficiency (Barr, 1999). Also, such a new generation O-O language is not readily available for the small embedded systems, primarily because of the overheads inherent in the O-O approach, e.g. CPU-time overhead (Pont, 2003).

Despite that Ada was a leading language that provided full support for concurrent and real-time programming, it has not gained much popularity (Brosgol, 2003) and has rarely been used outside the areas related to defense and aerospace applications (Barr, 1999; Ciocarlie & Simon, 2007). Unlike C, not many programmers nowadays are experienced in Ada, therefore only a small number of embedded systems are currently developed using this language (Ciocarlie & Simon, 2007). In addition, despite their approved efficiency, Ada compilers are not widely available for small embedded microcontrollers and usually need hard work to accept the program; especially by new programmers (Dewar, 2006). Indeed, both Ada and C++ have too large demand on low-cost embedded systems resources (e.g. memory requirements) and therefore cannot be suitable languages for such applications[1] (Walls, 2005).

[1] However, despite the indicated limitations of Ada, there has been a great deal of work on assessing a new version of Ada language (i.e. Ada-2005) to widen its application domain (see Burns, 2006; Taft *et al.*, 2007). It has been noted that Ada-2005 can have the potential to overwhelm the use of "C" and its descendants in embedded systems programming (Brosgol and Ruiz, 2007).

In a survey carried out by Embedded Systems Design (ESD) in 2006, it was shown that the majority of existing and future embedded projects to which the survey applied were programmed (and likely to be programmed) in C. In particular, the results show that for 2006 projects, 51% were programmed in C, 30% in C++, and less than 5% were programmed in Ada. The survey shows that 47% of the embedded programmers were likely to continue to use "C" in their next projects. See Fig. 1 for further details.

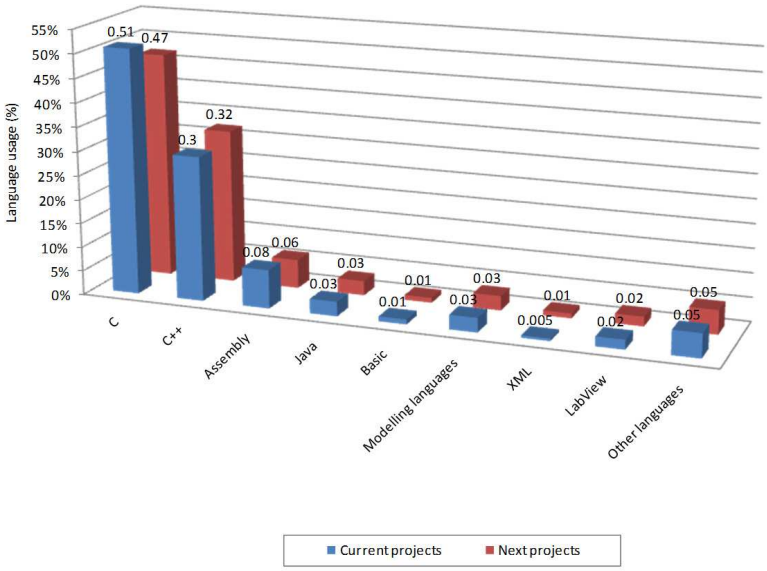


Fig. 1. Programming languages used in embedded system projects surveyed by ESD in 2006. The figure is derived from the data provided in (Nahas, 2008).

9. Using "C" to implement software for real-time embedded systems

Since "C" remains the most popular means for developing software in real-time embedded systems, it has been extensively used in the implementation of real-time schedulers and operating systems for embedded applications. In general, "C" was adopted in the software development of almost all operating systems (including RTOSs) in which schedulers are the core components (Laplante, 2004). In Michael Barr's book on embedded systems programming (i.e. Barr, 1999), it was noted that "C" is the main focus of any book about embedded programming. Therefore, most of the sample codes presented in Barr's book – for both schedulers and operating systems – were written in "C" and the key focus of the discussion was on how to use "C" language for 'in-house' embedded software development. However, some of the example code presented later in the book was written in C++ while Assembly language was avoided as much as possible. In (Barr & Massa, 2006), possible ways for implementing the eCos and the Embedded Linux, as a small and a large open-source operating systems (respectively), in "C" language were discussed. Other books which discuss the use of "C" language in the software implementation of real-time embedded systems include (Ganssle, 1992; Brown, 1994; Sickle, 1997; Zurell, 2000; Labrosse , 2000; Samek, 2002; Barnett et al., 2003; Laplante, 2004).

More specifically, using "C" language to implement the software code for particular scheduling algorithms is quite common. For example, Mooney et al. (1997) described a strategy for implementing a dynamic run-time scheduler using both hardware and software components: the software part was implemented using "C" language. Kravetz & Franke (2001) described an alternative implementation of the Linux operating system scheduler using "C" programming. It was emphasized that the new implementation can maintain the existing scheduler behavior / semantics with very little changes in the existing code.

Rao et al. (2008) discussed the implementation of a new pre-emptive scheduler framework using "C" language. The study basically reviewed and extracted the positive characteristics of existing pre-emptive algorithms (e.g. rate monotonic, EDF and LLF) to implement a new robust, fully pre-emptive real-time scheduler aimed at providing better performance in terms of timing and resource utilization.

Researchers of the Embedded Systems Laboratory (ESL), University of Leicester, UK have been greatly concerned with developing techniques and tools to support the design and implementation of reliable embedded systems, mainly using "C" programming language. An early work in this area was carried out by Pont (2001) which described techniques for implementing Time-Triggered Co-operative (TTC) architectures using a comprehensive set of "software design patterns" written in "C" language. The resulting "pattern language" was referred to as "PTTES[2] Collection" which contained more than seventy different patterns. As experience in this area has grown, this pattern collection has expanded and subsequently been revised in a series of ESL publications (e.g. Pont & Ong, 2003; Pont & Mwelwa, 2003; Mwelwa et al., 2003; Mwelwa & Pont, 2003; Pont et al., 2003; Pont & Banner, 2004; Mwelwa et al., 2004; Kurian & Pont, 2005; Kurian & Pont, 2006b; Pont et al., 2006; Wang et al., 2007, Kurian & Pont, 2007).

In (Nahas et al., 2004), a low-jitter TTC scheduler framework was described using "C" language. Phatrapornnant and Pont (2004a, 2004b) looked at ways for implementing low-power TTC schedulers by applying "dynamic voltage scaling" (DVS) algorithm programmed in "C" language. Moreover, Hughes & Pont (2008) described an implementation of TTC schedulers – in "C" language – with a wide range of "task guardian" mechanisms that aimed to reduce the impact of a task-overrun problem on the real-time performance of a TTC system. On the other hand, various ways in which Time-Triggered Hybrid (TTH) scheduler can be implemented in practice using "C" have been described in (Pont, 2001; Maaita & Pont, 2005; Hughes & Pont, 2008; Phatrapornnant, 2007). The ESL group has also been involved in creating software platforms for distributed embedded systems in which Shared-Clock (S-C) scheduling protocols are employed to achieve time-triggered operation over standard network protocols. All different S-C schedulers were implemented using "C" (for further details, see Pont, 2001; Ayavoo et al., 2007).

10. Conclusions

Selecting a suitable programming language is a key aspect in the success of the software development process. It has been shown that there is no specific method for selecting an appropriate programming language for the development of a specific project. However, the

[2] PTTES stands for Patterns for Time-Triggered Embedded Systems.

accumulation of experience along with subjective judgment enables software developers to make intelligent choices of programming languages for different application types.

Embedded software developers utilize different programming languages such as: Assembly, Ada, C, and C++. We have demonstrated that C is the most dominant programming language for embedded systems development. Although other languages may be winning ground when it comes to usage, C remains the de facto language for developing resource-constrained embedded systems which comprise a large portion of today's embedded applications.

11. Acknowledgement

The research summarized in this paper was partially carried out in the Embedded Systems Laboratory (ESL) at University of Leicester, UK, under the supervision of Professor Michael Pont, to whom the authors are thankful.

12. References

Ada (1980) "Reference Manual for the Ada Programming Language", proposed standard document, U.S. Department of Defense.

ANSVIP (1970) "American National Standard Vocabulary for Information Processing", American National Standards Institute, Inc., 1430 Broadway, New York, N.Y.

Ayavoo, D., Pont, M.J., Short, M. and Parker, S. (2007) "Two novel shared-clock scheduling algorithms for use with CAN-based distributed systems", Microprocessors and Microsystems, Vol. 31(5), pp. 326-334.

Baker, T.P. and Shaw, A. (1989) "The cyclic executive model and Ada. Real-Time Systems", Vol. 1 (1), pp. 7-25.

Barnett, R.H., O'Cull, L. and Cox, S. (2003) "Embedded C Programming and the Atmel Avr", Thomson Delmar Learning.

Barr, M. (1999) "Programming Embedded Systems in C and C++", O'Reilly Media.

Bates, D.G. (1968) "PROSPRO/1800", IEEE Transactions on Industrial Electronics and Control Instrumentation, Vol. 15, pp. 70-75.

Bieman, J.M., and Murdock, V. (2001) "Finding code on the World Wide Web: a preliminary investigation", Proceedings First IEEE International Workshop on Source Code Analysis and Manipulation, pp. 73-78.

Booch, G. (1991) "Object Oriented Design with Applications", Benjamin / Cummings.

Boulton, P.I.P. and Reid, P.A. (1969) "A Process-Control Language", IEEE Transactions on Computers, Vol. 18 (11), pp. 1049-1053.

Brosgol, B. and Ruiz, J. (2007) "Ada enhances embedded-systems development", Embedded.com, WWW website (Last accessed: November 2010) http://www.embedded.com/columns/technicalinsights/196800175?_requestid=167577

Broster, I. (2003) "Flexibility in dependable real-time communication", PhD thesis, University of York, York, U.K.

Brown, J.F. (1994) "Embedded Systems Programming in C and Assembly", Kluwer Academic Publishers.

Budlong, M. (1999) "Teach Yourself COBOL in 21 days", Sams.

Burns, A. (2006) "Real-Time Languages", Network of Excellence on Embedded Systems Design, WWW website (Last accessed: November 2010) http://www.artist-embedded.org/artist/Real-Time-Languages.html

Calgary (2005) "Calgary Ecommerce Services – Glossary", WWW website (Last accessed: November 2010) http://www.calgary-ecommerce-services.com/glossary.html

Carr, D. and Kizior, R.J. (2000) "The case for continued Cobol education", IEEE Software, Vol. 17 (2), pp. 33-36.

Chapman, S.J (2004) "Fortran 90/95 for Scientists and Engineers", McGraw-Hill Science Engineering.

Ciocarlie, H. and Simon, L. (2007) "Definition of a High Level Language for Real-Time Distributed Systems Programming", EUROCON 2007 The International Conference on "Computer as a Tool", Warsaw, September 9-12.

Cook, D. (1999) "Evolution of Programming Languages and Why a Language is Not Enough to Solve Our Problems", Software Technology Support Center, available online (Last accessed: November 2010)
http://www.stsc.hill.af.mil/crosstalk/1999/12/cook.asp

Davidgould (2008) "Davidgould – Glossary", WWW website (Last accessed: November 2010) http://www.davidgould.com/Glossary/Glossary.htm

Dewar, R.B.K. (2006) "Safety-critical design for secure systems: The languages, tools and methods needed to build error-free-software", WWW website (Last accessed: November 2010)
http://www.embedded.com/columns/technicalinsights/190400498?_requestid=177701

DIN (1979) "Programming language PEARL", Part 1. Basic PEARL, Part 2: Full PEARL, Deutsches Institut für Normung (DIN) German Standards Institute, Berlin, DIN 66253, 1979 (in English).

Fisher, J.A., Faraboschi, P. and Young, C. (2004) "Embedded Computing: A VLIW Approach to Architecture, Compilers and Tools", Morgan Kaufmann.

Flynn, I.M. (2001) "Generations, Languages", Macmillan Science Library: Computer Sciences, WWW website (Last accessed: November 2010)
http://www.bookrags.com/research/generations-languages-csci-01/

Ganssle, J. (1992) "The art of programming embedded systems", Academic Press, San Diego, USA.

Grogono, P. (1999) "The Evolution of Programming Languages", Course Notes, Department of Computer Science, Concordia University, Montreal, Quebec, Canada.

Halang, W.A. and Stoyenko, A.D. (1990) "Comparative evaluation of high-level real-time programming languages", Real-Time Systems, Vol. 2 (4), pp. 365-382.

Hansen, P.B. (1975) "The programming language Concurrent Pascal", IEEE Transactions on Software Engineering, Vol. 1 (2), pp. 199-207.

Hohmeyer, R.E. (1968) "CDC 1700 FORTRAN for process control", IEEE Transactions on Industrial Electronics and Control Instrumentation, Vol. 15, pp. 67-70.

Holyer, I (2008) "Dictionary of Computer Science", Department of Computer Science, University of Bristol, UK, WWW website (Last accessed: November 2010) http://www.cs.bris.ac.uk/Teaching/Resources/COMS11200/jargon.html

Hughes, Z.M. and Pont, M.J. (2008) "Reducing the impact of task overruns in resource-constrained embedded systems in which a time-triggered software architecture is employed", Trans Institute of Measurement and Control.

IFIP-ICC (1966) "The IFIP-ICC Vocabulary of Information Processing", North-Holland Pub. Co., Amsterdam.

ISO (2001) "ISO 5127 Information and documentation –Vocabulary", International Organisation for Standardisation (ISO).

Jalote, P. (1997) "An integrated approach to software engineering", Springer-Verlag.

Jarvis, P.H. (1968) "Some experiences with process control languages," IEEE Transactions on Industrial Electronics and Control Instrumentation, Vol. 15, pp. 54-56.

Jones, N. (2002) "Introduction to MISRA C", Embedded.com, WWW website (Last accessed: November 2010) http://www.embedded.com/columns/beginerscorner/9900659

Kircher, O. and Turner, E.B. (1968) "On-line MISSIL", IEEE Transactions on Industrial Electronics and Control Instrumentation, Vol. 15, pp. 80-84.

Koch, B. (1999) "The Theory of Task Scheduling in Real-Time Systems: Compilation and Systematization of the Main Results", Studies thesis, University of Hamburg.

Kravetz, M. and Franke, H. (2001) "Implementation of a Multi-Queue Scheduler for Linux", IBM Linux Technology Center, Version 0.2, April 2001.

Kurian, S. and Pont, M.J. (2005) "Building reliable embedded systems using Abstract Patterns, Patterns, and Pattern Implementation Examples", In: Koelmans, A., Bystrov, A., Pont, M.J., Ong, R. and Brown, A. (Eds.), Proceedings of the Second UK Embedded Forum (Birmingham, UK, October 2005), pp. 36-59. Published by University of Newcastle upon Tyne.

Kurian, S. and Pont, M.J. (2006) "Restructuring a pattern language which supports time-triggered co-operative software architectures in resource-constrained embedded systems", Paper presented at the 11th European Conference on Pattern Languages of Programs (EuroPLoP 2006), Germany, July 2006.

Kurian, S. and Pont, M.J. (2007) "Maintenance and evolution of resource-constrained embedded systems created using design patterns", Journal of Systems and Software, Vol. 80 (1), pp. 32-41.

Labrosse, J.J. (2000) "Embedded Systems Building Blocks: Complete and Ready-to-use Modules in C", Focal Press.

Lambert, K.A. and Osborne, M. (2000) "Java: A Framework for Program Design and Data Structures", Brooks / Cole.

Laplante, P.A. (2004) "Real-time Systems Design and Analysis", Wiley-IEEE.

Liberty, J. and Jones, B. (2004) "Teach Yourself C++ in 21 Days", Sams.

Maaita, A. and Pont, M.J. (2005) "Using 'planned pre-emption' to reduce levels of task jitter in a time-triggered hybrid scheduler". In: Koelmans, A., Bystrov, A., Pont, M.J., Ong, R. and Brown, A. (Eds.), Proceedings of the Second UK Embedded Forum (Birmingham, UK, October 2005), pp. 18-35. Published by University of Newcastle upon Tyne

Martin, J. and Leben, J. (1986) "Fourth Generation Languages Volume 1: Principles", Prentice Hall.

Mensh, M. and Diehl, W. (1968) "Extended FORTRAN for process control", IEEE Transactions on Industrial Electronics and Control Instrumentation, Vol. 15, pp. 75-79.

Mitchell, J.C. (2003) "Concepts in Programming Languages", Cambridge University Press.

Mwelwa C., Pont M.J. and Ward D. (2003) "Towards a CASE Tool to Support the Development of Reliable Embedded Systems Using Design Patterns", In: Bruel, J-M [Ed.] Proceedings of the 1st International Workshop on Quality of Service in Component-Based Software Engineering, June 20th 2003, Toulouse, France, Published by Cepadues-Editions, Toulouse.

Mwelwa, C. and Pont, M.J. (2003) "Two new patterns to support the development of reliable embedded systems", Paper presented at VikingPLoP 2003 (Bergen, Norway, September 2003).

Mwelwa, C., Pont, M.J. and Ward, D. (2004) "Code generation supported by a pattern-based design methodology", In: Koelmans, A., Bystrov, A. and Pont, M.J. (Eds.)

Proceedings of the UK Embedded Forum 2004 (Birmingham, UK, October 2004), pp. 36-55. Published by University of Newcastle upon Tyne

Nahas, M. (2008) "Bridging the gap between scheduling algorithms and scheduler implementations in time-triggered embedded systems", PhD thesis, Department of Engineering, University of Leicester, UK.

Nahas, M., Pont, M.J. and Jain, A. (2004) "Reducing task jitter in shared-clock embedded systems using CAN", In: Koelmans, A., Bystrov, A. and Pont, M.J. (Eds.) Proceedings of the UK Embedded Forum 2004 (Birmingham, UK, October 2004), pp. 184-194. Published by University of Newcastle upon Tyne.

Network Dictionary (2008) "Concurrent programming", WWW website (Last accessed: November 2010) http://wiki.networkdictionary.com/index.php/Concurrent_programming

Oerter, G.W. (1968) "A new implementation of decision tables for a process control language", IEEE Transactions on Industrial Electronics and Control Instrumentation, Vol. 15, pp. 57-61.

Opler, A. (1966) "Requirements for real-time languages", Communications of the ACM, Vol. 9 (3), pp. 196-199.

O'Reilly, T. (2006) "Programming Language Trends", WWW website (Last accessed: November 2010) http://radar.oreilly.com/archives/2006/08/programming-language-trends.html

Phatrapornnant, T. (2007) "Reducing Jitter in Embedded Systems Employing a Time-Triggered Software Architecture and Dynamic Voltage Scaling", PhD thesis, Department of Engineering, University of Leicester, UK.

Phatrapornnant, T. and Pont, M.J. (2004a) "The application of dynamic voltage scaling in embedded systems employing a TTCS software architecture: A case study", Proceedings of the IEE / ACM Postgraduate Seminar on "System-On-Chip Design, Test and Technology", Loughborough, UK, 15 September 2004. Published by IEE. ISBN: 0 86341 460 5 (ISSN: 0537-9989), pp. 3-8.

Phatrapornnant, T. and Pont, M.J. (2004b) "The application of dynamic voltage scaling in embedded systems employing a TTCS software architecture: A case study", Proceedings of the IEE / ACM Postgraduate Seminar on "System-On-Chip Design, Test and Technology", Loughborough, UK, 15 September 2004. Published by IEE. ISBN: 0 86341 460 5 (ISSN: 0537-9989), pp. 3-8.

Pont, M.J. (2001) "Patterns for time-triggered embedded systems: Building reliable applications with the 8051 family of microcontrollers", ACM Press / Addison-Wesley.

Pont, M.J. (2003) "An object-oriented approach to software development for embedded systems implemented using C", Transactions of the Institute of Measurement and Control, Vol. 25 (3), pp. 217-238.

Pont, M.J. and Banner, M.P. (2004) "Designing embedded systems using patterns: A case study", Journal of Systems and Software, Vol. 71 (3), pp. 201-213.

Pont, M.J. and Mwelwa, C. (2003) "Developing reliable embedded systems using 8051 and ARM processors: Towards a new pattern language", Paper presented at Viking PLoP 2003 (Bergen, Norway, September 2003).

Pont, M.J. and Ong, H.L.R. (2003) "Using watchdog timers to improve the reliability of TTCS embedded systems", in Hruby, P. and Soressen, K. E. [Eds.]Proceedings of the First Nordic Conference on Pattern Languages of Programs, September, 2002, pp.159-200. Published by Micrsoft Business Solutions.

Pont, M.J., Kurian, S. and Bautista-Quintero, R. (2006) "Meeting real-time constraints using 'Sandwich Delays'", In: Zdun, U. and Hvatum, L. (Eds) Proceedings of the Eleventh European conference on Pattern Languages of Programs (EuroPLoP '06), Germany, July 2006: pp. 67-77. Published by Universitätsverlag Konstanz.

Pont, M.J., Norman, A.J., Mwelwa, C. and Edwards, T. (2003) "Prototyping time-triggered embedded systems using PC hardware". Paper presented at EuroPLoP 2003 (Germany, June 2003).

Rao, M.V.P, Shet, K.C, Balakrishna, R. and Roopa, K. (2008) "Development of Scheduler for Real Time and Embedded System Domain", 22nd International Conference on Advanced Information Networking and Applications - Workshops, 25-28 March 2008, AINAW, pp. 1-6.

Roberts, B.C (1968) "FORTRAN IV in a process control environment", IEEE Transactions on Industrial Electronics and Control Instrumentation, Vol. 15, pp. 61-63.

Samek, M. (2002) "Practical Statecharts in C/C++: Quantum Programming for Embedded Systems", CMP Books.

Sammet, J.E. (1969) "Programming languages: history and fundamentals", Prentice-Hall.

Sanders, J. (2007) "Simple Glossary", WWW website (Last accessed: October 2007) http://www-xray.ast.cam.ac.uk/~jss/lecture/computing/notes/out/glossary/

Schoeffler, J.D. and Temple, R.H. (1970) "A real-time language for industrial process control", Proceedings of the IEEE, Vol. 58 (1), pp. 98-111.

Schutz, H.A. (1979) "On the Design of a Language for Programming Real-Time Concurrent Processes", IEEE Transactions on Software Engineering, Vol. 5 (3), pp. 248-255.

Sickle, T.V. (1997) "Reusable Software Components: Object-Oriented Embedded Systems Programming in C", Prentice Hall.

Sommerville, I. (2007) "Software engineering", 8th edition, Harlow: Addison-Wesley.

Steusloff, H.U. (1984) "Advanced real time languages for distributed industrial process control", IEEE Computer, pp. 37-46.

Taft, S.T., Duff, R.A., Brukardt, R.L., Ploedereder, E. and Leroy, P. (2007) "Ada 2005 Reference Manual: Language and Standard Libraries", Springer.

Walls, C. (2005) "Embedded Software: The Works", Newnes.

Wang, H., Pont, M.J. and Kurian, S. (2007) "Patterns which help to avoid conflicts over shared resources in time-triggered embedded systems which employ a pre-emptive scheduler", Paper presented at the 12th European Conference on Pattern Languages of Programs (EuroPLoP 2007).

Watson, D. (1989) "High Level Languages and Their Compilers", Addison-Wesley.

Wexelblat, L. (1981) "History of Programming Languages", Academic Press.

Wilson, L.B. and Clark, R.G. (2000) "Comparative Programming Languages", Addison-Wesley.

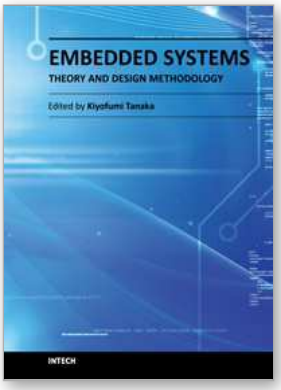
Wirth, N (1993) "Recollections about the development of Pascal", Proceedings of the 2nd ACM SIGPLAN conference on history of programming languages, pp. 333-342.

Wirth, N. (1977) "Modula - A programming language for modular multiprogramming", Software - Practice and Experience, Vol. 7, pp. 3-35.

Wizitt (2001) "T223 – A Glossary of Terms (Block 2)", Wizard Information Technology Training (Wizitt), WWW website (Last accessed: November 2010) http://wizitt.com/t223/glossary/glossary2.htm

Zurell, K. (2000) "C programming for embedded systems", CMP Books.

Zuse, K (1995) "A Brief History of Programming Languages", Byte.com, WWW website (Last accessed: November 2010) http://www.byte.com/art/9509/sec7/art19.htm



Embedded Systems - Theory and Design Methodology
Edited by Dr. Kiyofumi Tanaka

ISBN 978-953-51-0167-3
Hard cover, 430 pages
Publisher InTech
Published online 02, March, 2012
Published in print edition March, 2012

Nowadays, embedded systems - the computer systems that are embedded in various kinds of devices and play an important role of specific control functions, have permitted various aspects of industry. Therefore, we can hardly discuss our life and society from now onwards without referring to embedded systems. For wide-ranging embedded systems to continue their growth, a number of high-quality fundamental and applied researches are indispensable. This book contains 19 excellent chapters and addresses a wide spectrum of research topics on embedded systems, including basic researches, theoretical studies, and practical work. Embedded systems can be made only after fusing miscellaneous technologies together. Various technologies condensed in this book will be helpful to researchers and engineers around the world.

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821