Object Orientation in Embedded and Real-Time Systems
- Useful or Not

Jens Vesti
School of Information Technology
Deakin University
vesti@deakin.edu.au

Sunday, June 1, 2003
Contents

1 Introduction 3

2 Understanding Embedded and Real Time Systems 3
   2.1 Timeliness 4
   2.2 Resources 4
   2.3 Reliability 4
   2.4 Compilers and Tools Available 4

3 Trends in Embedded and Real-Time Development 5

4 Methods and Methodologies 6
   4.1 Advantages 6
   4.2 Disadvantages 6

5 Programming 7
   5.1 Advantages 7
   5.2 Disadvantages 7
   5.3 Summary 8

6 Conclusion 9

References 9
Abstract

Professionals tend not to use OO when developing embedded and real-time systems. It is the intention in this report to present and evaluate whether it is as a result of simple stubbornness and conservatism or there really is a reason not to use OO. We present some of the issues that professionals and academics have with OO in embedded and real-time systems. The characteristics of embedded and real-time systems are being presented. Next these are being compared with the issues in developing such systems. This is done from the method and methodology point of view and the programming. In the end we conclude that there is a downside to using OO, but it can still be beneficial to use OO in larger and less critical embedded and real-time systems.

1 Introduction

Embedded and real-time systems have traditionally been developed using non-object oriented methods and methodologies and imperative programming languages in support. As requirements for more stability in the systems and quicker time-to-marked arise, it is fair to wonder whether different methods, methodologies and languages could lead the development in the right direction.

In this report we will take a look whether object orientation (OO) can be applied to these kinds of applications. We will do this from two angles, the method and methodology point of view and the programming. The reason being that the problems applying OO to embedded and real-time systems are quite different.

2 Understanding Embedded and Real Time Systems

Embedded and real-time systems differ from other systems in various ways. In order to get a clear picture of how OO fit into the picture of embedded and real-time systems we need to take a look at the characteristics in development of such systems that may affect the way we think about introducing OO in the development. With characteristics we mean those that differ in an embedded and real-time system development project from other types of systems such as administrative systems.

\[^{1}\text{Without going into further details why, we should clearly distinguish between methods and methodologies as they are not the same [1]}\]
2.1 Timeliness

In real-time systems it is not, always, about making the system fast, but more to make the system respond within the time restrictions given. It is said that real time systems are constrained in time [2]. There are 2 commonly accepted different categories of real-time systems.

*Hard real-time* systems are systems if whose process’ deadline is missed the system has failed, which leads to extraordinary costs or even loss of lives. Examples of such systems could be a car’s ABS where if a deadline is not kept the breaking system does not work properly.

*Soft real-time* systems accept delays in the response time from the operating system, as they do not lead to significant financial loss as a consequence of a missed deadline. Such systems may include vending machines or toys.

The categories presented here do not depend on the system but on the application of the system and the tasks it is set to serve [8]. Often the categories are mixed so that the system contains both hard and soft real-time parts.

2.2 Resources

Resources must be managed with care. Business systems running on new workstations, embedded systems often run on smaller platforms such as Intels 80186 processor with few kilobytes of memory available. As the hardware comes with the system it is up to the developers to make the software fit into the hardware, and the hardware usually gets cheaper the “slower” and older the technology is so the developer has to deal with the resources available.

2.3 Reliability

Embedded and real-time systems requires more reliability than other types of systems as users in general are more depended on these systems. Ensuring reliability requires a more ”rigorous development process” [2].

2.4 Compilers and Tools Available

Embedded systems often requires specialised tools to develop software to. In the case of personal computers a lot of tools are being mass produced with a lower price to follow [2]. This is not always the case for embedded systems, and the tools (such as compilers, real-time operating systems etc.) come with a higher price and/or lower quality. The restrictions given may
therefore complicate the development process. This could be that the OO compiler introduces a bigger overhead than other compilers for other platforms, which makes the usefulness of OO smaller.

3 Trends in Embedded and Real-Time Development

In order to identify the problems in applying OO in embedded and Real-Time systems an understanding of current trends in the business and opinions on the use of OO in development of these types of systems must be addressed.

The reasons for using OO are many and more or less obvious, but the ultimate goal is to accommodate the increased complexity and decreased time to marked [3]. In order to do that we need methods to modulate our software as well as make it reusable.

But OO programming is not a magic paradigm that when applied all problems in software development will disappear. It is merely a subset of tools in our arsenal [6] which we can use. This is a good point as not all systems may be suitable for OO. Some computer scientists may argue that OO applies for everything, which is true to some extend, but not in all cases though. In some systems using OO may not give you the control of what is required or may give an overhead that, compared to the advantages one get from using OO, cannot be neglected.

When that is said there is still the trend that for some reason real-time system developers tend to be the last to adopt new technology [2] such as OO when it resolves in increased resource use and application size. The reasons are somewhat unclear but factors such as education and tradition could be the biggest contributors as OO is in some way newer than traditional methods, methodologies and empirical languages. [3] argues that the main reason is due to OO programming performance overhead, but they do, however, not mention that OO programming, methods and methodologies are conceptually useless in embedded and real-time systems. This is further backed up in [6], where Ganssle praises the use of encapsulation in order to partition the code (divide-and-conquer), but he states that this can be done in low-level languages as well as in OO languages, and using OO methods and methodologies therefore do not necessary imply using OO programming with overhead to follow.

Most embedded systems have been soft real time systems. That is the safety mechanisms provided have been of non-software type (eg. mechanical, hydraulic etc.) [8]. This is changing as the mechanisms may just as well reside in software as in hardware. It does however set new requirements to the software provided in terms of reliability. This is one of the main arguments from Kopetz [8] to refine the software engineering techniques. This could for example be to introduce OO in the systems development.
The trends presented here are also some of the main issues in using OO in embedded and real-time systems development. The trends are many and it would not be realistic to address all issues here but only the major ones.

4 Methods and Methodologies

There is no doubt that the use of OO methodologies can be of significant benefit in the development of embedded systems. As embedded systems get more complicated and time to marked decreases, it is essential to provide methodologies that support such requirements. But introducing OO into embedded and real time development is not a silver bullet. There are disadvantages as well as advantages. We will in the following take a look at a few.

4.1 Advantages

The advantages of OO methods and methodologies are:

- Greater abstraction which makes the development of larger systems easier to manage and comprehend as it is more intuitive to use objects, compared to structured development methods and methodologies.
- Easier reuse in form of analysis or design patterns. Software patterns have evolved and become more popular. A major part of software development is used on creating and refining a model of the system, but many models are actually quite similar. So bringing in standard or reused software patterns can reduce the development time spent. Examples of analysis patterns can be found in [4] and in [5] for design patterns.

4.2 Disadvantages

On the other side we have some disadvantages that need to be solved in order for OO to be successfully adopted in embedded development.

- Most OO methods are focused on the value domain and takes the temporal domain for “an implementation issue” [8]. OO will only be successful if the temporal domain properties are dealt with by the object technology [7].
- Some of the more recent software and design specification techniques such as real-time UML and real-time CORBA does not have a solid foundation in the understanding of time and focus on temporal properties as second to the value domain [8].
- OO methods and methodologies are non-formal whereas in embedded systems we need the formalism in the specification [11] in order to describe eg. the time constrains in the
system. However there has been attempts on providing such formalism to OO [9].

5 Programming

Making the design is one thing, realising it is another, and in the case of real time and embedded systems more critical due to the previous mentioned constrains and issues that need to be dealt with. We will in the following describe the advantages and disadvantages in using OO programming in real time systems development as this seems to be one of the most critical criteria for using OO in real time and embedded systems.

5.1 Advantages

There are many advantages, but we will only describe those relevant to what we are trying to accomplish in this report.

- One of the main advantages in OOP is the information encapsulation. By cleanly encapsulate implementation issues into simpler objects the programmer can easier create an overview and therefore easier handle larger programs. This is a way of modulate the program so it is possible to "divide and conquer" the program. By modulating the program it is easier, if done correctly, to develop and maintain the system.
- Another advantage is easier reuse of code. Providing reusable code the development time shortens dramatically and the quality may be improved as modules used may be tested more or the errors are being detected as some of the code is already in production.

5.2 Disadvantages

The disadvantages depend on the compiler and language used. But can be generalised into following:

- Polymorphism introduces a small runtime overhead in terms of additional pointer dereference and small memory extra used for virtual functions, but if applied with care the overhead can be partially removed if not totally.
- Using copy constructor may require temporary objects to be created which introduces a slight overhead. This can, as well as the overhead in using polymorphism, be avoided if the programmer is careful when writing the code.
- Exceptions are not OO but are nonetheless being supported in most OO languages. It was introduced in ADA. Studies have shown that the overhead of exception, if they are not used, are up to 3% [2].
- Memory fragmentation can be a major problem in embedded and real-time systems, especially if the system is to run for longer times. Even though the developer is skilled and avoids memory leaks, it can be difficult to spot memory fragmentation. Memory fragmentation happens when frequent allocation and freeing of memory creates smaller holes of different size that, because of the sizes that are too small cannot directly be used again.

There are several solution how to avoid this problem [2].

One is to overload the `new()` and `delete()` operator so that it will allocate equally large memory areas every time regardless of the use.

Another is to allocate all the memory from the start and therefore eliminate the dynamic allocation that creates the problem.

A third and more problematic method is to introduce a garbage collector as in eg. Java. The problem is that the processes will have to be halted in order to reorganise the memory. A solution that is less attractive in a system that is constrained in time. There are however approaches combining garbage collection with the first approach in combination with non-contiguous memory use [10].

- Execution time and power consumption should be considered when using OOP programming. Especially power consumption may be of major concern in some embedded systems. If the system is restricted in the power it may receive this should be considered carefully. With OO languages there is, if not applied properly, some overhead which may affect not only the performance but also the power consumption [3].

- Lack of control over resources and time consumption as abstraction level increases. With assembler the time consumed per line or statement is well-known, this is less the case with imperative languages and certainly with OO languages. Using a real-time operating system increases this problem as control over the operating system most likely does not exist.

5.3 Summary

In more general terms, it is a fact that OO programs are, depending on compilers and languages, a bit more slow and inefficient than empirical languages. Comparing programs written in the OO language C++\(^2\) with corresponding programs written in C, they are no more than 10% slower [2].

\(^2\)According to Bjarne Stroustrup who initially developed the language, C++ is not fully OO but merely OO friendly
6 Conclusion

We have described in this article different views on OO in embedded and real-time systems, just as we have presented in more details the various issues in using OO. There are many advantages in using OO, but it cannot be neglected that there are just as many, if not more, disadvantages. However, as with most other types of systems (such as administrative systems) it all depends on the type of system to be developed and the environment the system is meant for. There is a break-even on when it is beneficial to use OO. There are different factors that decide this break-even such as size of project, time-to-marked and requirements to reliability, resource use etc. It can therefore be concluded that OO to some extend do belong in embedded and real-time systems but is a case-to-case decision whether it should be used.

References


