White Paper

The future of real time embedded systems

in high volume printing

C. Delnooz, L.A.J. Dohmen, J. van de Hee, R.P.M. Jacobs, D.P.C. Janssen, A.B. van der Wal
Océ R&D, E-Mail: chris.delnooz@oce.com, lou.dohmen@oce.com, johan.vandehee@oce.com, ruud.jacobs@oce.com, ton.janssen@oce.com, alex.vanderwal@oce.com.

Embedded system developers face many and sometimes conflicting challenges. To increase efficiency and drive down hardware cost, they must aim for ever higher levels of system integration. But to keep applications manageable and scaleable, they need solutions that are as general and modular as possible. At the same time, embedded system complexity rises, quality standards go up and development time needs to decrease.

With such opposing forces, how can the development of embedded systems still remain successful, maintainable and reliable?

This white paper gives an overview of how embedded system engineers can cope with these issues, now and in the future. It does so by showing what successful strategies we have developed in the past few years. Taking the development of the fastest duplex cut sheet printer in the world as a case example, we outline four elements of successful embedded software design:

- Model driven development as the leading method for designing and building real-time systems.
- A standard real-time embedded software architecture.
- Reuse of software as a "company philosophy".
- Systematic approach to quality control.

The lessons learned from our experiences with embedded design are so general that they can be applied to a wide range of real-time system designs.

Content

Introduction

Modern high-volume cut sheet printers are advanced process controlling systems. Designing, building, and testing such systems is no simple task; they must monitor and control an increasing amount of mechanical, electronic, and chemical processes within the printing equipment. Ever more time-critical production schemes, rising customer expectations, cost reduction, and stricter environmental and safety regulations push technology to the physical limits of what printing technologies are capable of.

This white paper presents strategies for designing, building and testing embedded systems in the industrial printing environment. It looks at what developers need in order to deal with the vastly rising complexity of printing equipment. Embedded system developers need to cope with different hardware components with different software running on each component, changing hardware platforms and tool sets used for generating code, rising quality standards and expanding functionality. At the same time, new environmental and safety regulations must be met.

Some of the problems of designing embedded systems are presented here, as well as a case study in which the design of the fastest duplex printer in the world is described. We will present the most relevant lessons learned from our new systematic design approach. Since our work started about a decade ago, our experiences may serve as a basis for an outlook to future trends and developments.

Challenges

High-tech demands for embedded systems

Any real-time embedded system must address the following issues:

- Event handling: real-world systems involve complex sequences of events – internal events caused by components of the system or external events caused by an operator or the environment.
- Timing behavior: many events require a real-time response from the embedded system. There are two different types of real-time requirements: hard real-time requirements, which cause system failure if not met, and soft real-time requirements, which are performance requirements that mostly cause a loss of productivity if not met.
- Concurrency: events may occur simultaneously. Since embedded systems can do only a finite number of things at a time, they must implement a scheduling policy that controls when tasks execute.
- Synchronization and communication: concurrent systems must agree on the use of hardware resources such as memory, processing capacity, bandwidth or power.
- Exception handling: in the physical world mechanical systems may wear down. Users may push machines beyond their physical limits or make mistakes. So, embedded systems need to be extremely fault tolerant.
- Robustness: embedded systems must monitor and control a wide variety of processes for days or even years on end in local environments that may be extremely cold, hot, dusty, or humid.

Actual systems have to exhibit numerous other properties and obey constraints such as size, extensibility, maintainability, power consumption, weight, user-friendliness, or meeting environmental regulations.

From such a list of characteristics one can deduct that one has to live with compromises. There is not one "silver bullet" solution for embedded systems design. Strategies have to be multi- and cross-disciplinary and should reflect that technology is just one part of the solution. Innovative management and organizational structures are equally important.

The scope of the conflicting challenges faced in embedded system design will be illustrated by a short historical overview of rising system complexity.

Historical overview

The race for complexity

One major challenge of embedded systems is that hardware performance and the quality of software tools are out of sync. Whereas the performance of micro controllers doubled about every three years in the past two decades, the complexity of designing, programming, and testing embedded systems with controller boards has grown at a much faster rate.

The net result of this development is that embedded developers have to face a "tools gap" – the overall complexity of a system grows at a faster rate than the capacity of available tools to predict or analyze the real-time behavior of such systems. Even though common engineering practices have evolved, from "no method" (using assembly language) as a programming environment on 8 bit micro controllers) via "structured method" (using C) to "object orientation" (commonly using C++), designing real-time embedded software in large teams is an extremely difficult task.[1]

Today's developers have to deal with problems that require multitasking, object-oriented programming and, real-time execution all at once. Within the printing domain, embedded software engineers have to cope with the following additional factors (see also figure 1):

- Cheaper hardware components need to be kept within fault tolerances using intelligent controlling software.
- Software needs to compensate for fewer hardware components that deliver more and better functionality.
- Rising equipment complexity creates more interaction with sensors and actuators.
- Increasing printing speed.
- Higher levels of embedded system integration.
- Better user interfaces, and in general, more complex interfacing with an increasing number of standards and protocols.
- Increased need for modularity.
- Decreasing time to market.
- Lower total cost of ownership.

All this has presented embedded systems developers with immense challenges. At the end of the 1990s, surveys showed that future projects would fail to meet requirements or even fail completely, because of the overall poor state of programming tools in complex real-time environments.[2]

Historical overview

The race for complexity

Figure 1: Embedded systems complexity in printing equipment

Product	Year	Method	OS	Memory (ROM)	Processor	Language	Images/ minute	Print resolution
1900	1979	No	No	64 K	8085	Assembler	45	Analog
2500	1984	Yourdon	Own	394 K	8088 & 68000	C/Pascal	100	Analog
3165	1995	Ward & Mellor	VxWorks	1 M	8088 & 68000	C	62	600 DPI
Varioprint 2090	2002	UML / Rose RT	VxWorks	5*4 M	5*AMD SC 520	C++	85	600 DPI
Varioprint 6250	2006	UML / Rose RT	VxWorks & Own	1*64 M & 16*256 K	Celeron & 16*Infineon XC167	C++ & C	250	1200 DPI

While engineering practices evolved from "no method" to "UML", on-board memory capacity increased by a factor 1000 in the past two decades. Overall system complexity has risen even faster.

Varioprint 6250

- World fastest duplex printer (250 images/minute).
- 100% accurate registration.
- Automatic and complete job recovery.
- Printing all paper formats between 178 x 203 and 320 x 488 mm.
- Productive mixed media printing from 12 different trays.

Case example

Building the fastest duplex printer in the world

In 2001, Océ Technologies set out designing the fastest cut sheet duplex printer in the world. The initial specifications of the "Varioprint 6250" were a real challenge to embedded systems. Printing speed should be a mind-blowing 250 images per minute – 80% faster than anything ever built. Printing should be duplex in near-offset quality. At the time, it was unknown whether the physical constraints of the printing process and the required synchronization of paper flow and high-bandwidth digital image streams would allow for printing both sides of a sheet of paper at the same time at the required printing speed. The machine should be capable of handling any paper format and size, including rare oversizes, without slowing down. Up to twelve paper trays – each tray possibly loaded with pre-printed or prepunched media – should be addressed independently.

Manual reloading of the paper trays should not interrupt the printing process. The system should be capable of detecting paper size and orientation fully automatically. A micro-positioning system correctly adjusts each sheet's time of arrival, rotation and aligning, all adjusting done in less then half a second.

One of the most challenging tasks was designing a system that would keep track of dozens of individual sheets of paper at the same time, while they were routed through the paper path. The paper path could be of variable length, due to different system configurations, and had a maximum length of more than 15 meters. Planning, routing, positioning, synchronizing and printing 6 million sheets of paper per month through this path, while handling more than a dozen types of sheets simultaneously in the path and coping with possible media jams safely and correctly, was at the limit of what embedded systems would be capable of.

It would have been impossible to use one or even a few central processing units to control this machine. Such a centralized system would simply fail to meet critical timing demands.

Also, due to physical distances between the embedded hardware and the sensors and actuators mounted throughout the machine, large amounts of cabling would have been required. We decided to use the most advanced hardware architectures and software tools available. In the new, distributed architecture, we introduced intelligent subnodes.[3]

The heart of an intelligent subnode is formed by a universal embedded control platform. The platform consists of a high performance 16-bit CPU, running at 40 MHz. To guarantee hard real-time responsiveness, the nodes run a fixed time scheduler, with a round robin time of 2 msec. Time slots are 100 µsec with the maximum delay between signals caused by interrupt handling of just 30 µsec.

Case example

Building the fastest duplex printer in the world

A total of 17 such control cards are placed throughout the hardware, each on top of a customized I/O board. Some of the nodes meet hard real-time requirements, others are used for soft real-time actions. The subnodes are connected through two Controller Area Network (CAN) buses to a main control unit, running a fast, 2Ghz Intel Pentium Celeron. The CAN buses have been specifically designed to be robust in electromagnetically noisy environments such as automobiles. Since the physical distances in a car are comparable to those in our printers, the CAN bus is ideally suited for use in printing equipment. The two CAN buses use different dedicated protocols: the Inter Node Protocol (INP) and the Océ Finisher Interface (OFI) protocol. A USB 2.0 connection is used for transmitting compressed bitmaps (600x1200 dpi) from the controller to the image datapath subnode. Another optional USB network connects all subnodes. This enables direct high-speed logging and debugging raw sensor data in R&D.

All actions must be synchronized over the non-real-time CAN bus. In order to accomplish this, actions must be timestamped and then sent to subnodes in advance. It also implies that all subnodes and mainnode have the same notion of time (see figure 2).

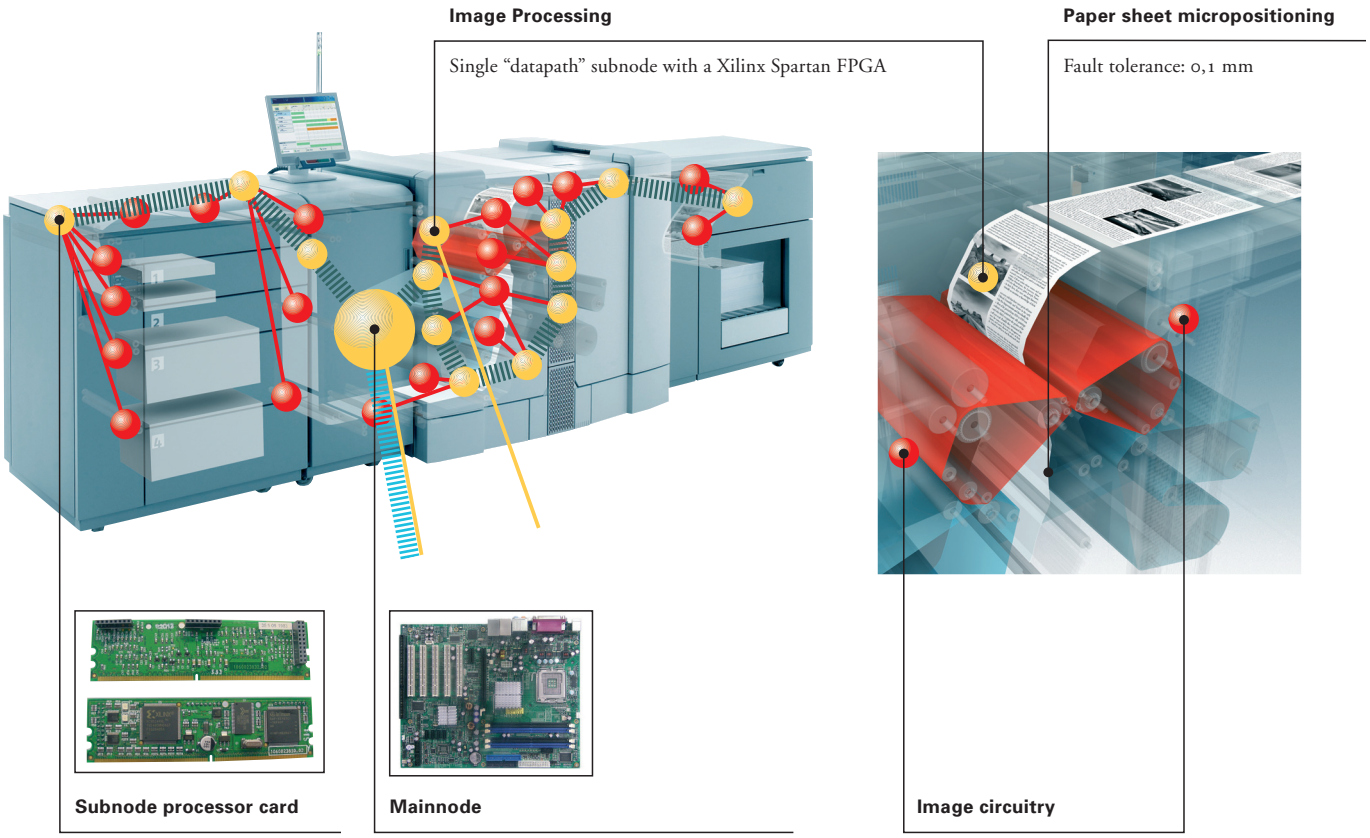
Thus, interconnection and interoperation were the keywords in this new machine. For the embedded system developer team, this translated into convoluted architectures and critical timing and integration issues.

We realized that in order to avoid project delays, missed deadlines, and exploding costs, we would have to do more than just design software and hardware. We needed a fundamental look at our existing software tools (Step 1), system architectures (Step 2), and design strategies (Step 3). It would also be necessary to make testing and quality control procedures more systematic and efficient (Step 4).

Case example

Building the fastest duplex printer in the world

Figure 2: An overview of embedded systems within the Varioprint 6250



Mainnode

Subnode with synchronised clock
Maximum time deviation: 10 µsec

Interaction point
Sensor-triggered exchange moment at wich sheets or images are passed to the next transport function.

Ethernet connection
For logging and connection with controller

2 CAN databuses (INP, OFI):
- 250 kbit/sec bi-directional datatransfer
- Facilitates communication between mainnode and subnodes

USB 2.0 connection
Used for high speed logging of raw sensor data. Also used for the transport of bitmap data from controller to subnode

Sensor /actuator cable
Used for high speed logging of raw sensor data

Case example

Building the fastest duplex printer in the world

Step 1: Existing software tools

Facing the increasing difficulties with existing software tools, Océ decided in 1998 to experiment with Model Driven Development (MDD). This was a remarkable step, since MDD was known primarily in the telecommunications domain. In other real-time software-design domains it was hardly ever used in the actual process of coding software. At the time, automatically generated code from models was regarded as complex and resource-hungry. Most real-time developers held (and some even do so today) that the syntax of MDD tools would be too general and imprecise. So how could MDD work for real-time applications?

From our extensive experiments and real-world experiences within running projects we have learned that modern tools for MDD not only work for complex systems using C++ under a real-time, multi-threaded operating system, but that they also work surprisingly well with severely constrained systems that utilize an execution framework instead of a multi-threaded OS.[4,5]

We used an industry-leading MDD development environment, which is based upon the ROOM Language (ROOM/UML, see inset). The tool is fully model driven and object oriented. It generates C++ or C, which runs on numerous operating systems. Using this tool, we were able to decompose complex systems into subsystems (see figure 3). This allowed us to model a vast number of concurrent finite state machines. This solution generates very efficient code on targets varying from 8-bit micro controllers to 32-bit general purpose processors.

Unified Modeling Language

The Unified Modeling Language (UML) was developed in 1997[6]. UML is a visual language and has both syntax and semantics. Its graphical elements of objects are used as representations of a concept, but also tell you something about their specific content. It has an extensive vocabulary for capturing behavior and process flow. And it can even be used to make a blueprint of software. These blueprints, much like architectural drawings, use icons to represent structure (classes and objects, attributes, operations, and relationships), function (the functionality of the system from the user's point of view) and behavior (sequences, activities, and states).

UML was developed from earlier modeling languages, such as the Real-time Object Oriented Modeling language (ROOM). ROOM was introduced by Bran Selic, Garth Gullekso and Paul T. Ward in 1994. The essential concepts of ROOM have been incorporated into UML 2.0. Among these are composite structures, ports and message passing. It is these concepts that make UML suitable for the construction, design and implementation of real-time systems.

The first tool which took advantage of ROOM was ObjecTime. This tool has evolved into "Rational Rose Real-Time" (RRT). RRT is considered to be a leading tool for UML generated code in real-time embedded systems (http://www-306.ibm.com/software/rational/). Interestingly, RRT is also used by NASA for the James Webb Space Telescope – a large, infrared-optimized space telescope, scheduled for launch in 2013 (see: www.jwst.nasa.gov).

Case example

Building the fastest duplex printer in the world

Equipped with the MDD techniques, we were able to design, build and debug the hugely complex system of the "Varioprint 6250" (see figure 2). Design guidelines were used to instruct the engineers how to create simple and efficient models, even when using a lot of concurrency. Such models are based on multiple-processor, multiple-threaded hardware and software architectures. Because of the run-to-completion semantics of active objects, the usual difficulties, in which the output of the process is unexpectedly and critically dependent on the sequence of other events ("race conditions") were virtually absent. The common and very difficult error in real-time design, in which competing actions are waiting for each other to finish ("deadlocks"), was no problem any more to our team.

With MDD, engineers were able to concentrate on functionality and quality. Moreover, all of our engineers could understand and talk to each other in the same language. And since explaining a diagram is far more simple than explaining a piece of code, MDD stimulated dialogue within the organization as a whole.

Overall, the decision in favor of MDD greatly enhanced the application development cycle, while improving product maintenance and quality control.

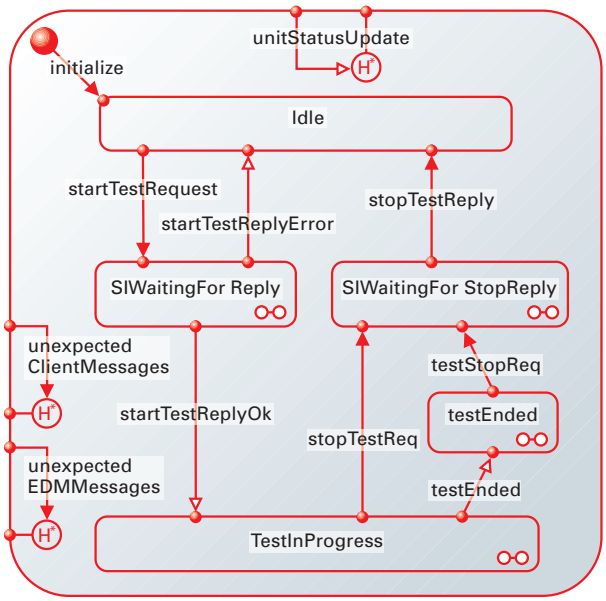


Figure 3: State diagram

Part of a hierarchical finite state machine. The state transitions inside an active object contain hand-written code. Concurrency is guaranteed since the system ensures that state transitions execute atomically (run to completion). A code generator generates a complete source file for the finite state machine, including the hand-written code. Interestingly, the generated code is never touched by the engineers due to the abstraction that the model offers. One exception to this is when code is debugged at source level in order to look at what happens inside state transitions. But it is also possible to debug finite state machines visually.

Case example

Building the fastest duplex printer in the world

Step 2: Standardised system architectures

Much of the quality of a real-time system is defined by its system architecture. Such an architecture gives a picture of the hardware and software components in use, their relationships to each other and to the environment, and the principles governing the design and evolution of the system.

So, before designing the "Varioprint 6250", the Océ-developer team introduced the "Embedded Software Reference Architecture" (ESRA). ESRA defines the generic context for the MDD software tools and hardware and software components in use. It does so by defining universal mechanisms and procedures for handling the basic functionality of printing engines. This means that ESRA defines status behavior, controlling paper sheets and images, and error handling. For distributed deployment, ESRA defines persistent storage mechanisms and communication protocols. ESRA also offers designers standardized methods for developing and analyzing systems based on its general architecture.

Designing, implementing, and maintaining a reference architecture over a long period of time has proven to be a delicate matter. The key issue is that the architecture itself is not static: projects with new functionality come in and old projects are discontinued. This makes ESRA a managerial as well as a technological challenge.

From the technological point of view, one of the main advantages of ESRA is that software deployment is no longer tied to specific hardware deployments. This means that ESRA standard architecture components – which have been designed for large, industrial printers – may also fit into smaller office printers.

As for the managerial aspects, the success of a standard system architecture is critically dependent on a flexible, non-hierarchical organizational structure. Essential elements are:

- Collective "ownership": the reference architecture is shared by all architects in "using" projects.
- No obligations: every architect has the freedom of using solutions not covered in ESRA – provided that sound arguments support such a decision.
- Team dialogue: architects and engineers are regularly talking and working together and are all trained in how to use the architecture (regular three-day architecture courses teach the fundamentals of ESRA).
- Interdisciplinary dialogue: Having a reference architecture as a starting point enables the embedded architect in an early phase to have impact on both hardware and system architecture.

Overall, ESRA has established itself as a robust design environment. Its main advantages are:

- New projects can be "jump-started" with total time needed for making executable software for a working product prototype reduced to just a few months.
- With ESRA taking care of much of the standard functionality's of a new project, engineers can focus on innovation.
- Research and development creates a progressive set of knowledge, technologies, and experts.
- ESRA is a technology catalyst, providing dynamic feedback and control to both software reuse and model driven development (see figure 4).

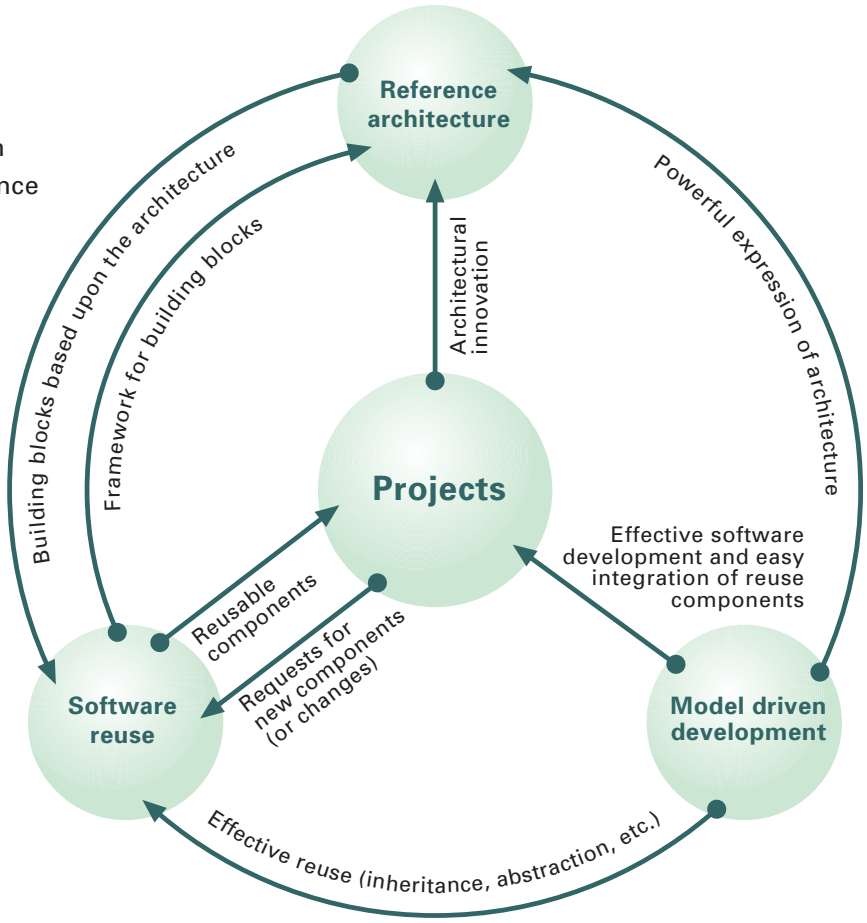
ESRA's high level of system integration and modularity allows for implementation on diverse printer models. One of the most fascinating examples of hardware independent deployment is that we are currently able to use the same architecture on completely different platforms: it runs on a high end distributed platform with more than 18 subnodes, as well as on a very limited platform with only a single processor.

Case example

Building the fastest duplex printer in the world

Figure 4: ESRA as a technology catalyst

Note the dynamic interaction between projects, the reference architecture, software reuse, and MDD.



Step 3: Reusing software components

With the right software tools and standard system architecture available, the Océ developer team encountered another major issue: software reuse. Reusing software is essential for efficiency and long-term product development. It means using standardized software modules for meeting current and future technological demands.

So, well before the "Varioprint 6250" was designed, a small group of software engineers started delivering components with standard behavior. The components were described in UML models and ESRA data structures. From the start, these models were tested and used in multiple real-world projects. The projects were all in different phases of the life cycle, having different functional requirements and technological innovations. Today, the number of software components has grown to a set with which the complete standard control software of any kind of printing equipment can be built. (see figure 5,6)

This is a remarkable achievement, since software reuse is a complex, and a highly knowledge-intensive subject.[7] Software reuse must focus not only on the problems related to developing reusable components, but also faces many challenges related to keeping previously defined reusable components up-to-date, slim, manageable and useful. Finding the right balance between long-term interests – universal applicable models – short-term needs – suitability for particular projects, time to market, ad hoc flexibility – may fail easily.

Case example

Building the fastest duplex printer in the world

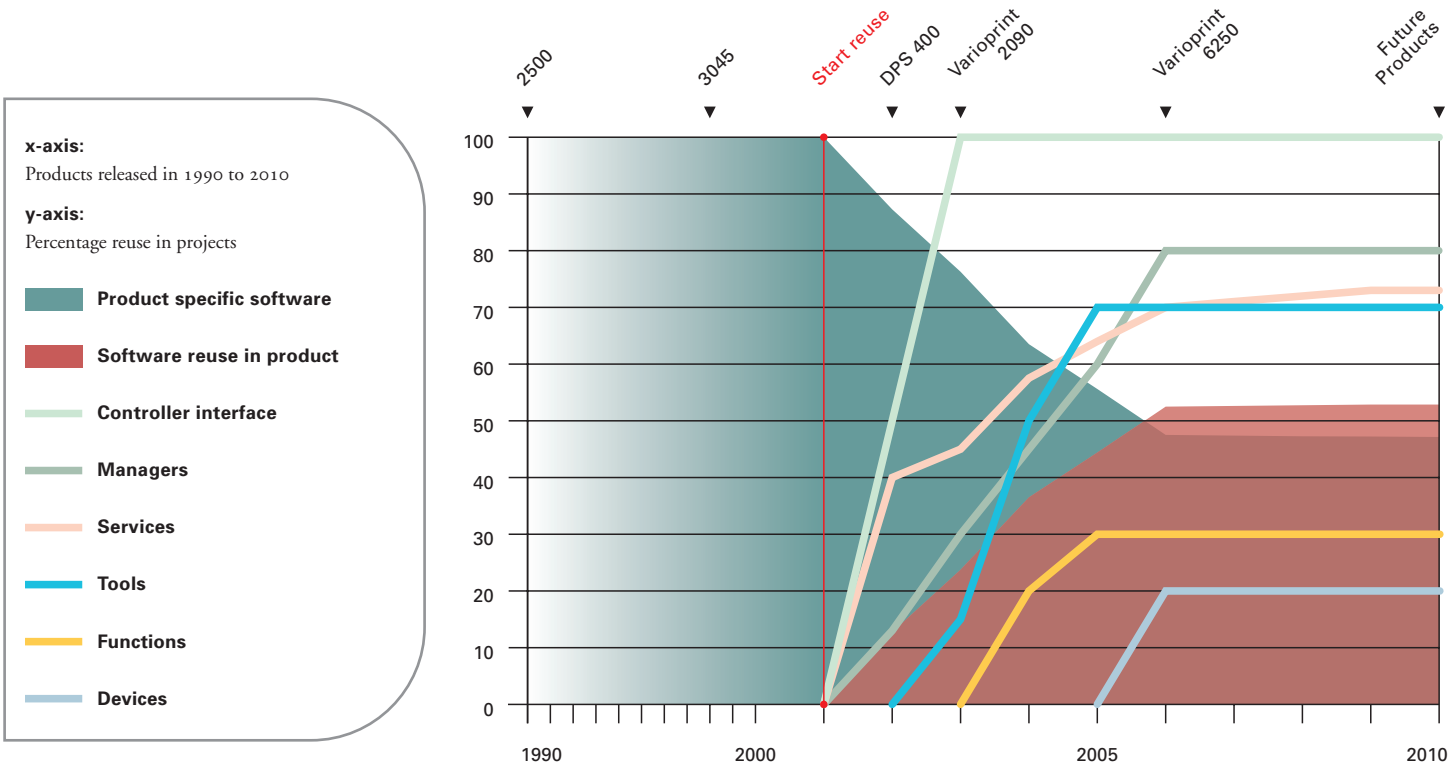
The most prominent causes of failure in reuse projects are:

- Inferior quality. Risk: buggy reuse components that do not meet the customers quality expectations
- "Top-down" approach. Risk: reusable components become too large and generic. They are poorly optimized for running projects. Priorities for bug fixing and new functionality are getting out of sync with the needs of engineering teams.
- Insufficient organizational support. Risk: the immanent conflicting forces between short term project needs and long term reuse portfolio strategies may endanger the cooperation between engineers and line management. Ultimately, engineers may not want to participate in the "reuse-project" at all.
- Dogmatism. Risk: throttling innovation. When reuse is applied dogmatically, engineers lack the freedom to look for new solutions.

Figure 5: Estimated and forecast net effect of reuse efforts within Océ from 2000-2010.

Not all software is suitable for reuse. Reuse can be much more effective in such areas as "services" and "controller interface" than in "functions" or "devices" (product-specific coding).

- Controller interface: components for communication with the controller PC.
- Managers: standard ESRA managers, e.g. status manager, engine information manager.
- Services: diverse service functions.
- Tools: development tools, e.g. for data logging
- Functions: machine-specific functionality, e.g. paper handling.
- Devices: drivers for sensors and actuators.



Case example

Building the fastest duplex printer in the world

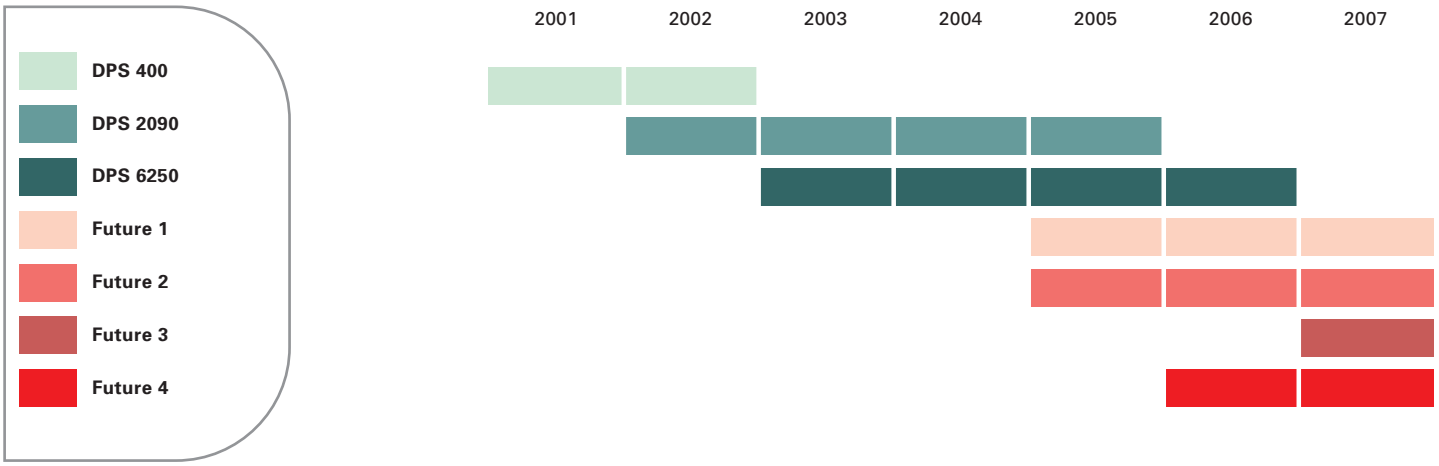
We prevent the first two causes of failure, as described on the previous page, by designing reuse software components not as abstract concepts, but as building blocks which can be directly used in currently running projects. We also found social software techniques helpful, such as giving everyone involved access to an intranet website and a wiki. Both are used to list all available reuse components, documentation, release notes, and bug reports. The last two causes are prevented by having full management commitment and focus for innovation. This is something that starts with the architecture and ends with evolved software components. It is about having the right staffing and competence available at every stage of the process.

Moreover, a "change control board" is in charge of new functionality and bug fixing. The board also sets maintenance priorities and discusses new developments and releases. Engineers from running projects participate in the board. Overall, an informal, flat organizational structure has proven essential for software reuse.

Although reuse does not need MDD, we found that it is greatly aided by the object orientation approach of MDD. Interfaces are narrow and uniform. Inheritance allows for adaptable behavior. Overall, reuse significantly shortened our time-to-market, increased efficiency, and improved the overall quality of the product.

Figure 6: Timeline

One of the challenges of software reuse is that projects are in different phases of their life cycle. This makes finding the right balance between short-term goals (tight integration) and long-term goals (modularity) a delicate matter.



Case example

Building the fastest duplex printer in the world

Step 4: quality control

Debugging and testing the new "Varioprint 6250" turned out to be a real challenge to our developers. For it is still not feasible to prove from theory that a modeled deterministic finite state machine will work as specified in all possible states.[8] This is because there are no mature simulation tools available that would allow formal analyses of problems such as deadlocks or critical timings for every possible execution path of the system. Testing and prototyping thus has to run through various scopes and states of complexity:

- Simulation: parts of the new design are simulated on a desktop computer. A complete engine node can be tested on a Windows PC, since MDD-models contain hardly any platform dependent code. Porting to the real target platform is easy.
- Table-top testing: a fragment of the target hardware is built with a few sensors and actuators. This allows for early-phase testing of critical real-time control loops. Peer reviews check the validity of the design. Automatic regression tests check whether program changes had any unintended effects on overall software functionality.
- Embedded testing: a functional representation of the real machine is built, with all required sensors and actuators available. The path of the paper through the machine is simulated. This enables testing of the controlling architecture, drivers, interfaces and inter-processor communication. This approach results in a high level of software quality before integration with the actual printer, which in turn results in very short integration lead times.

An important consideration is the difficulty of debugging software once it has been downloaded to the target hardware. Tools such as debuggers and in-circuit emulators are only of use when a problem can be reproduced. And this is not always the case. Reproducing errors may also be too time-consuming because of the non-deterministic character of the embedded software.

The Océ-team developed a method in which nearly all bugs can be traced without the need for reproducing failure scenarios. This has been achieved by:

- Data logging. All internal events, states and other relevant information for every state of every processor and subnode is logged, resulting in 12.5 megabytes of logging per minute. Subnodes are equipped with extra logging facilities, including a high-bandwidth USB 2.0 interface. Even in production, extensive logging is being carried out.
- Data probing. The transmission on the CAN databuses can be probed directly and in real-time. The high-speed datatransmission between engine and controller, via a 100 Mbit ethernet connection, can also be probed.
- Data injection. The same channels available for probing can be used for real-time injection of raw data.

We developed sophisticated "stubbing" methods with which various parts of the physical hardware can be replaced by a simulated part on different levels of the system hierarchy. "Stubbed" components can vary from individual sensors and actuators to complete subnodes. With stubbing, we are able to simulate all real-time sensor data that a physical sheet of paper produces when it moves through the paper path of the target machine. This enables us to test large portions of the paper handling machinery, without wasting even one single physical sheet of paper.

Outlook

Meeting future demands of the printing industry

Today, Océ's state-of-the-art real-time embedded system development environment performs extremely well. And it still carries many promises for the future. How do we translate those promises into real innovations?

Several times a year, we let a group of architects and engineers identify future technologies that the company may require. A "technology road map" is then created and if necessary, action is taken.

On a broad level, our current technology roadmap shows three future developments within embedded systems.

First, new software tools will solve numerous technical design and debugging issues. A general trend is the integration of stand-alone design- and debugging tools into one intelligent "knowledge system". Interestingly, powerful open source tools have entered the embedded domain in recent years. Open source solutions such as "Eclipse" offer new perspectives on building and maintaining high-quality extensible architectures, runtimes, and application frameworks (see www.eclipse.org).

Second, further strategic application of MDD, software reuse, and standard system architecture design will bring the dream of embedded development – to use context-dependent code as little as possible – further within reach. This will mark a distinct shift in the role of developers – from designers of customized code to integrators of standard modules. One interesting instance of such a development is the future role of testing. With an ideal set of reusable software and hardware modules available, developers would have to spend less time testing. Instead, they would be able to concentrate on innovations and new functionality. A cross-disciplinary approach may be able to generate testing, monitoring and design tools for other disciplines in the printing industry, such as mechanics or chemistry.

Third, the role of simulations will increase substantially. Future software tools will enable embedded system developers to simulate a wide range of critical hardware characteristics of the printing equipment. Such simulation tools will not just enable detailed insight in the printer's paper path, but will also be capable of predicting power consumption, noise levels and other environmental characteristics. New, cross-domain simulation tools might be able to predict things such as mechanical wear and tear of components over time. With such tools, it would be possible to analyse all possible hardware failures in the equipment life-cycle in great detail even well before the actual machine has been built.

In a more distant future, it may be possible that three-dimensional simulation tools will revolutionize maintenance and the development process of printers.

However, the real challenge of embedded system engineering is not just to find ever more state-of-the art technologies, but to explore new strategies for dealing with the relentless rise in system complexity. At Océ, we take pride being at the forefront of this endeavor.

Profile

Océ is one of the world's leading suppliers of professional printing and document management systems. Océ employs 23.000 people world-wide. About 3.500 work in Venlo, where headquarters, manufacturing, logistics and R&D are located.

Océ was founded in 1877 in Venlo. The company entered the copying business in 1920. In 1986, the first fully self-developed printers were introduced. In 1994 the first digital copier/printers entered the market. With 90 percent of Océ's sales relating to digital products, Océ is now ranked among the world's top 100 IT businesses.

Total R&D manpower in Venlo is 1.000, making Océ one of the top ten R&D companies in the Netherlands.[9] About 8 percent of the total revenue of Océ is spent on R&D. Most of the research is multi-disciplinary. Océ's total R&D budget expanded from about 60 million Euro in 1986 to 220 million Euro in 2006.

Acknowledgements

Text: Dr. Sybe Rispens
www.rispens.de

Design: Maximilian Werner
www.mw-kommunikationsdesign.de

References

1. Tanenbaum, Andrew. *Computer Networks,* Upper Saddle River, NJ: Prentice Hall, 1996. Tanenbaum, Andrew S. *Modern Operating Systems,* Prentice Hall International, 2001.

2. Hyysalo, J., Parviainen, P. and Tihinen, M. "Collaborative embedded systems development: survey of state of the practice," *Engineering of Computer Based Systems,* March, 2006, p. 9-30.

3. Kopetz, Hermann. *Real-Time Systems: Design Principles for Distributed Embedded Applications,* Berlin: Springer, 1997, pp. 29-45. Rao, Navneet. "Next-Generation 65nm FPGA", *FPGA and structured ASIC Journal,* Volume X, Number 3, 2007.

4. Dohmen, Lou A. J. and Somers, L. J. "Experiences and Lessons Learned Using UML-RT to Develop Embedded Printer Software." *Product Focused Software Process Improvement.* 4th International Conference, PROFES 2002, p. 475-484.

5. Janssen, Ton and Graham, William. "Modeldriven Development of Resource-constrained Embedded Applications," *IBM technical report,* June 2004. (www-128.ibm.com/developers/rational/library/04/r-3151/index.html)

6. Selic, Bran, Gullekson, Farth and Paul T. Ward. *Real-time object-oriented modelling,* John Wiley & Sons Inc, 1994. Booch, Grady, Rumbaugh, James and Jacobson, Ivar. *The Unified Modelling Language User Guide,* Addison-Wesley Professional, 1998. Douglass, Bruce Powel. *Real-Time UML: Developing Efficient Objects for Embedded Systems,* Pearson Education, 1999. Pilone, Dan. *UML 2.0 in a Nutshell. A Desktop Quick Reference,* O'Reilly Media, 2005.

7. Gamma, Erich, Helm, Richard, Johnson, Ralph and Vlissides, John. *Design Patterns – Elements of Reusable Object-Oriented Software,* Reading, Massachusetts: Addison Wesley, 1995.

8. Buttazzo, Giorgio C. *Soft Real-Time Systems: Predictability Vs. Efficiency,* Berlin: Springer, 2005, p. 195-206. Douglas, Bruce Powell. *Real-Time Design Patterns: Robust Scalable Architecture for Real-Time Systems,* Reading, Massachusetts: Addison-Wesley, 2002.

9. "Special R&D in cijfers", *Technisch Weekblad,* 31. March 2007, p. 21.

Printing for
Professionals

Océ R&D
St. Urbanusweg 43, Venlo, the Netherlands
P.O. Box 101, 5900 MA Venlo, the Netherlands
Telephone (+31) 77 359 22 22
Telefax (+31) 77 354 47 00
E-mail: info@oce.com
For general information about Océ: telephone (+31) 77 359 20 00
For information and services, visit us at www.oce.com