# Development Experiences of a Component-based System

Magnus Larsson[1], Ivica Crnkovic[2]

[1]ABB Automation Products AB, LAB, 721 59 Västerås, Sweden
Magnus.Larsson@mdh.se

[2]Mälardalen University, Department of Computer Engineering, 721 23 Västerås, Sweden,
Ivica.Crnkovic@mdh.se

## Abstract

*Building software systems with reusable components brings many advantages. If the reuse concept is utilized on several levels of a system development, the development becomes more efficient, the reliability of the products is enhanced, and the maintenance requirement is significantly reduced. The levels of reuse are spread out from the reuse of source code and common libraries, through the reuse of large business components, up to the reuse of the standard products in the configuration of large systems. Designing, developing and maintaining components for reuse is, however, a very complex process which places high requirements not only for the component functionality and flexibility, but also for the development organization. In this paper, we discuss the different levels of component reuse, and certain aspects of component development. As an illustration of reuse issues, we present a successful implementation of a component-based system, which is widely used for industrial process control.*

## 1 Introduction

Reuse and an open component-based architecture are the keys to the success of systems with a long lifecycles. Designing a system that supports this approach, requires more effort in the design phase and the time to market might be longer, but in the long run, the reusable architecture will prove profitable. The reuse concept can be used on different levels: On a low level it is a reuse of source-code, and small-size components. More reuse is obtained with larger components encapsulating business functions. Finally, the integration of complete products in complex systems can be seen as the highest level of reuse. On each level of reuse there are specific demands on the reusable components, on the component management and on the integration process.

This paper describes important issues related to the development and maintenance of reusable components and as an example uses the ABB Advant industrial process control system. In chapter 2 we give an overview of the Advant system design and in chapter 3 the main characteristics of Advant reusable components. Chapter 4 outlines all the development and maintenance aspects of a component based system, which must comply with customer requirements. During evolution of the system new technologies were developed which resulted in the appearance on the market of many components with the same functionality as the proprietary ones. The fact that new components must be incorporated into the existing systems introduces new demands on the system development process. These new issues are discussed in chapter 5.

## 2 ABB Advant Open Control System

ABB is a global electrical engineering and technology company, serving customers in power generation, transmission and distribution, in industrial automation products, etc. The ABB group is divided into companies, one of which, ABB Automation Products AB, is responsible for development of industrial automation products. The automation products encompass several families of industrial process-control systems including both software and hardware.

The main characteristics of these products are reliability, high quality and compatibility. These features are results of responses to the main customers requirements: The customers require stable products, running around the clock, year after year, which can be easily upgraded without impact on the existing process. To achieve this, ABB uses a component-based system approach designing the extendable and flexible systems.

The Advant Open Control System (OCS) [1] is component-based to suit different industrial applications. The range includes systems for Power Utilities, Power Plants and Infrastructure, Pulp and Paper, Metals and Minerals, Petroleum, Chemical and Consumer Industries, Transportation systems, etc. An overview of the Advant system is shown in Figure 1.

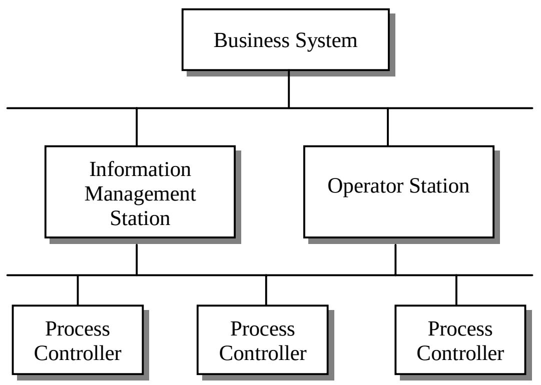

**Figure 1.** An overview of the conceptual architecture of the Advant open control system.

Advant OCS performs process control and provides business information by assembling a system of different families of Advant products. Process information is managed at the level of process controllers. The process controllers are based on a real-time operating system and execute the control loops. The Operator Station (OS) and Information Management Station (IMS) gather and supervise product information, while the business system analysis information for optimization of the entire processes. Advant products use standard and proprietary communication protocols to satisfy real-time requirements.

## 2.1 Designing for Reuse

The Advant system architecture is designed for reuse. Different products such as Operator and Information Management Stations are used as system components in assembling complete systems. Examples of the products are. The two operator station versions, Master OS and MOD OS are used in building different types of operator applications.

Having up-to-date information at the right time and in the right place is critical to the success of any industrial operation and Advant OCS therefore includes information management functions with real-time insight into all aspects of the process controlled. Advant Information Management has an SQL-based relational database accessible to resident software and all surrounding computers. Historical data acquisition reports, versatile calculation packages and a application programming interface (API) for proprietary and third party applications are examples of the functionality provided. Advant components have access to process, production and quality data from any Process Control unit in a plant or in an Intranet domain.

**2.1.1 Scalability.** Advant OCS can be configured in a multitude of ways, depending on the size and complexity of the process. The initial investment can consist of stand-alone process controllers and, optionally, local operator stations for control and supervision of separate machines and process sections. Subsequently, several process controllers can be interconnected and, together with central operator and information management stations build up a control network. Several control networks can be interconnected to give a complete plant network which can share centrally located operator, information and engineering workplaces.

**2.1.2 Openness.** The system is further strengthened by the flexibility to add special hardware and software for specific applications such as weighing, fixed- and variable-speed motor drives, safety systems and product quality measurements and control in for example the paper industry. Second- and third party administrative, information, and control can also be easily incorporated

**2.1.3 Cost-effectiveness.** The step-by-step expansion capability of Advant OCS allows users to add new functionality without making existing equipment obsolete. The system’s self-configuration capability eliminates the need for engineers to enter or edit topology descriptions when new stations are physically installed. New units can be added while the system is in full operation. With Advant OCS, system expansion is therefore easy and cost-effective.

## 2.2 Designing with Reuse

Designing with reuse of existing components has many advantages [2]. The software development time can be reduced and the reliability of the products increased. These were important prerequisites for the Advant OCS development.

Advant OCS products can be assembled in many different configurations for use in various branches of industry. Specific systems are designed with the reuse of Advant OCS products and other external products. This means which customers get a tailor-made system that meets their needs. External products and components can be used together with the Advant OCS due to the openness of the system. For example a satellite communication component, which is used to transmit data from the offshore station to the supervision system inland, can be integrated with the Advant OCS (Figure 2).

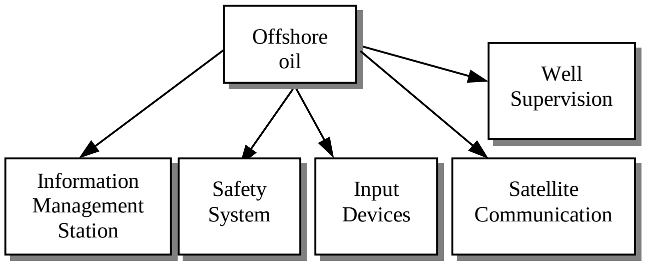

**Figure 2.** Solution for an offshore oil production platform.

The offshore system in Figure 2 uses the Information Management Station to gather all relevant data from the oil producing process and this is then transmitted to the headquarters on shore via the external satellite component. A safety component is used to provide a more secure system. Another component is the well supervision unit which monitors the oil wells.

Component-based systems for different types of applications can be easily designed and produced because of the open and scalable architecture of Advant OCS.

### 2.3 Experiences

The Advant system is a successful system that can be used to build different types of process automation systems. It has effective build and integration procedures. The main reason for the success is - component-based architecture and the component features (flexibility, robustness, stability and compatibility).

However, the cost to achieve these features has been high. To be able to suit the requirements of an open system, new ABB products had always to be backward compatible. It would have been easier to develop a new system that did not have to be compatible with the previous systems. To guaranty that the system is backward compatible works as a warranty that the current system will integrate with new products and this makes the system trustworthy. The system is carefully designed and a lot of effort has been put in test and maintenance.

Development with big components that are easy to reuse increase the efficiency significantly compared to reusing a smaller component that could have been developed in-house to the same cost as buying it. The Advant OCS products are examples of big components that have been used to assembly process automation systems.

## 3 Reusable Components

### 3.1 Components

The Advant OCS products are component based to minimize the maintenance and development cost. Figure 3 shows the component architecture of the operator station.

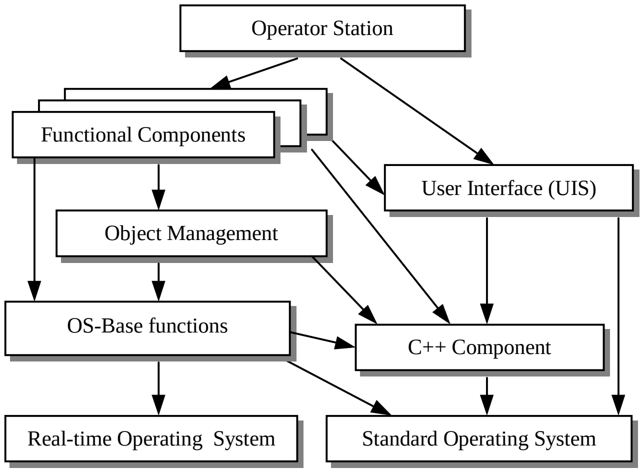

**Figure 3.** The operator workstation is assembled from components.

The operator station consists of a specific number of functional components and of a set of standard Advant components. These components use the User Interface System (UIS) component. Object Management Facility (OMF) is a component which handles the infrastructure and data management. OMF is similar to CORBA [3] in that it provides a distributed object model with data, operation and event services. The UxBase component provides drivers and other specific operating system functions. Helper classes for strings, lists, pointers, maps and other general-purpose classes are available in the C++_complib component. The components are built upon operating systems, one, a standard system(such as Unix or Windows), and the other a proprietary real-time system.

To illustrate different aspects of component-based development and maintenance, we shall further look at two components:

- Object Management Facility (OMF), a business type of component with a high-level of functionality and a complex internal structure;
- C++_complib is a basic and a very general component.

### 3.2 Object Management Facility (OMF)

OMF is object-oriented middle-ware for industrial process automation. It encapsulates real-time process control entities of almost every conceivable description into objects that can be accessed from applications running on different platforms, for example Unix and Windows NT. Programming interfaces are available for many languages such as C, C++, Visual Basic, Java, Smalltalk and SQL while interfaces to the IEC 1131-3 [4] process control languages are under development. OMF is also adapted to Microsoft Component Object Model (COM) via adapters and another component called OMF COM aware. The adapters for OPC (OLE for Process Control) [6] and OLE Automation are also implemented. Thanks to all these software interfaces, OMF makes process and production data available to the majority of computer programmers and users i.e. even to those not necessarily involved in the industrial control field. For instance, it is easy to develop applications in Microsoft Word, Excel and Access to access process information. OMF has been developed for demanding real-time applications, and incorporates features, such as real-time response, asynchronous communications, standing queries and priority scheduling of data transfers. On one side OMF provides industry-standard interfaces to software applications, and on the other, it offers interfaces to many important communication protocols in the field (see Figure 4), including MasterNet, MOD DCN, TCP/IP and Fieldbus Foundation. These adapters make it possible to build homogeneous control systems out of heterogeneous field equipment and disparate system nodes.

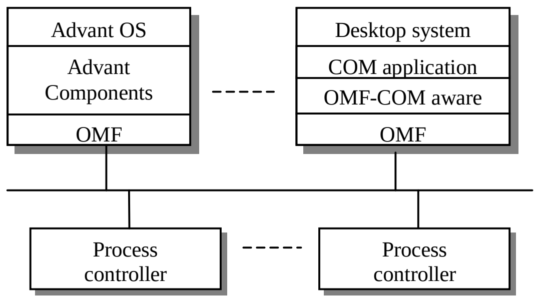

**Figure 4.** Many different components and products use the OMF component.

OMF reduces the time and cost of software development by providing frameworks and tools for a wide range of platforms and environments. These utilities are well integrated into their respective surroundings, allowing developers to retain the tools and utilities they prefer to work with.

### 3.3 C++_complib

C++_complib is a class library that contains general-purpose classes, such as containers, string management classes, file management classes, etc. The C++_complib library was developed when no standard libraries, such as STL [5], were available on the market. The main purpose of having this library was to improve the efficiency and the quality, and to get the uniform usage of the basic functions. The c++_complib library is reused whenever possible even in cases where similar services are supported by a specific platform or development package. The library was ported to several platforms, and in some cases the implementation part had small variations. The declaration part is the same on all platforms.

## 4 Different Reuse Aspects

### 4.1 Component generality and efficiency

Reuse principles place high demands on the reusable components. The components must be sufficiently general to cover the different aspects of their use. At the same time they must be concrete and simple enough to serve a particular requirement in an efficient way. Developing a reusable component requires three to four times more resources than developing a component, which serves a particular case [7]. In the case of C++_complib, the situation was simpler, because the requirements from the theoretical point of view were clear. It was relatively easy to define the interface, which was used by different components in the same way. The situation was more complicated with complex components, such as OMF. Although the basic concept of component functionality was clear, the demands on the component interface and behavior were different in different components and products. Some components required a high level of abstraction, others required the interface to be on a more detailed level. These different types of requirements have led to the creation of two levels of components: OMF base, including all low-level functions, and OMF framework, containing only a higher level of functions and with more pre-defined behavior and less flexibility.

### 4.2 Evolution of Functional Requirements

The development of reusable components would be easier if functional requirements did not evolve during the time of development. As a result of new requirements for the products, new requirements for the components will be defined. The more reusable a component is, the more demands are placed on it from products using that component. A number of the requirements coming from different products, may be the same or very similar, but this is not necessarily the case for all requirements passed to the components. In addition to the requirements stated from the products, a component also collects demands on the internal behavior (for example code improvement, improvement of the maintainability, etc.). This means that the number of requirements of reusable components grow faster than of particular products or of a non-reusable piece of software. To satisfy these requirements the components must be updated more rapidly and the new versions must be released more frequently than the products using them.

The evolution process is illustrated in Figure 5. The first graph shows the growing number of requirements for certain products. The number of requirements of a common component grows faster. Some of the product requirements are satisfied with each new release of a product, which are shown as steps on the second graph. The component satisfying the requirements by its releases, which normally precede the releases of each product.

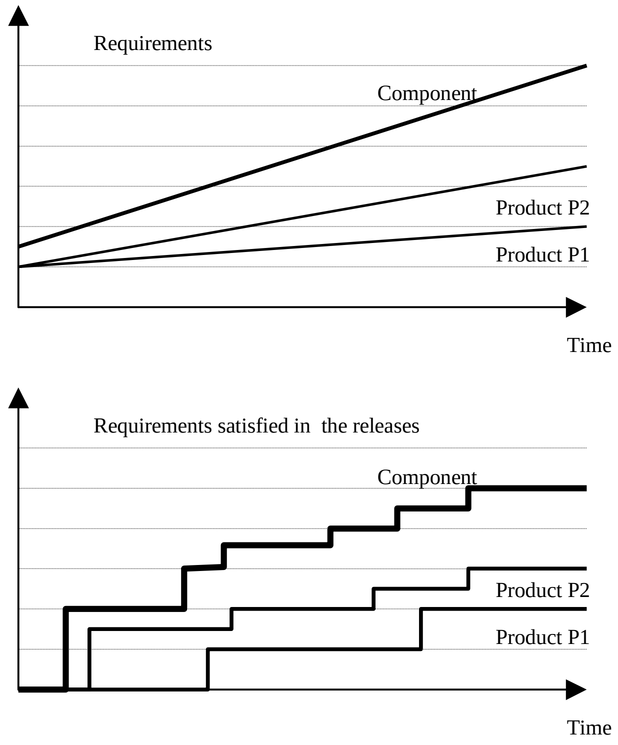

**Figure 5.** To satisfy the requirements the reusable component must be modified more often.

Indeed this was the case with both components we are analyzing here: New functions and classes were required from C++_complib, and new adapters and protocol support were required from OMF. The development time for these components was significantly shorter than for products: While new versions of a product are typically released each six months, new versions of components are released as least twice as often.

## 4.3 Migration Between Different Platforms

During their several years of development, Advant products have been ported on different platforms. The reasons for this were the customer requirement, that the products should run on specific platforms, and general trends in the growing popularity of certain operating systems. Of course, at the same time, new versions and variants of the platform already used appeared, supporting new, better and cheaper hardware. Figure 6 shows the migration path of Advant products on different platforms.

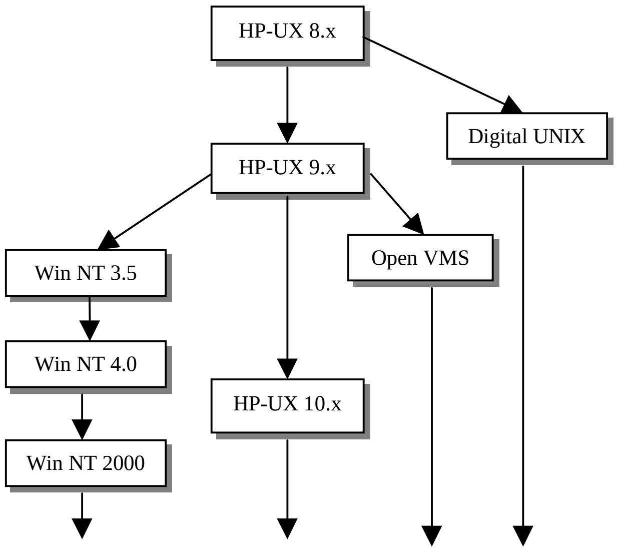

**Figure 6.** Different platforms supported by OMF.

As an important part of the reuse concept was to keep the high-level components unchanged as far as possible, it was decided to encapsulate the differences between operating systems in low-level components. This concept works, however, only to some extent. The minimal activity required for each platform is to rebuild the system for that platform. To make it possible to rebuild the software on every platform, standard-programming languages C and C++ have been used. Unfortunately, different implementations of the C++ standard in different compilers, caused problems in the code interpretation and required the rewriting of certain parts of the code. To ensure that standard system services are available on all platforms, the POSIX standard has been used. POSIX worked quite well on different Unix platforms, but much less so on Windows NT. The second level of compatibility problem was Graphical User Interface (GUI). The main dilemma was whether to use exactly the same GUI on every platform, or to use the standard "look and feel" GUI for each platform. This question applied particularly on NT in relation to Unix platforms. Experience has shown that it is not possible to give a definitive answer. In some cases it was possible to use the same GUI and the same graphical packages, but in general, different GUIs were implemented.

The main work regarding to the reuse of code on different platforms was performed on low-level components, such as UxBase and OMF. While UxBase provides different low-level packages for every platform (for example different drivers), OMF capsulated the differences directly in the code using conditional compilation. OMF itself is designed in such a way that it was possible to divide the code into two layers. One layer is specific for each operating system, and the other layer, with the business logic, is implemented for all of the supported platforms. Reuse issues on different platforms for C++_complib were easier, strictly the package contains general algorithms, which are not hard connected to a specific operating system. Some problems appeared however, related to different characteristics of compilers on different platforms.

## 4.4 Compatibility

One of the most important factors for successful reusability is the compatibility between different versions of the components. A component can be replaced easily or added in new parts of a system if it is compatible with its previous version. The compatibility requirements are essential for Advant products, since smooth upgrading of systems, running for many years, is required. Compatibility issues are relative simple when changes introduced in the products are of maintenance and improvement nature only. Using appropriate test plans, including regression tests, functional compatibility can be tested to a reasonable extent. More complicated problems occur when new changes introduced in a reusable component eliminate the compatibility. In such a case, additional software, which can manage both versions, must be written.

A typical example of such an incompatible change, is a change in the communication protocol between OMF clients and servers. All different versions of OMF must be able to talk to each other to make the system flexible and open (Figure 7 It is possible to have different combinations of operating systems and versions of OMF and it still works. This has been solved with an algorithm that ensures the transmission of correct data format. If two OMF nodes have the same version, they talk in their native protocol.

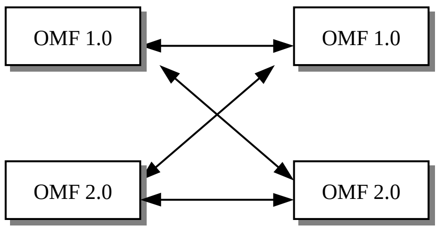

**Figure 7.** Different versions of OMF must be compatible with all older versions.

If an old OMF node talks with a new, the new OMF is responsible for converting the data to the new format, this being designated RMIR ("receiver makes it right"). If a new OMF sends data to an older, the older OMF can not convert the data since it is unaware of the new protocol. In this case the newer OMF must send in the old protocol format, SMIR ("sender makes it right"). This algorithm builds on that fact all machines know about each other and that they also know what protocol they talk. However, if an OMF-based node does not know of the other node then it can always send in a predefined protocol referred to as “well known format”. All nodes do recognize this protocol and can translate from it. This algorithm minimizes the number of data conversions between the nodes.

In the case of C++_complib the problems with compatibility were somewhat different. New demands on the same classes and functions appeared because of new standards and technology. One example is the use of C++ templates. When the template technology became sufficiently mature, the new requirements were placed for C++_complib: All the classes were to be re-implement as template classes. The reason for this was the requirement for using basic classes in a more general and efficient way. Another example was a Unicode support in addition to ASCII-support. These new functions were added by new member-functions in the existing classes and by adding new classes using the inheritance mechanism for reusing the already existing classes.

## 4.5 Development Environment

When developing reusable components several dimensions of the development process must be considered:

- Support for development of components on different platforms;
- Support for development of different variants of components for different products;
- Support for development and maintenance of different versions of components for different product versions.

To cope with these types of problems, it is not sufficient to have appropriate product architecture and component design. Development environment support is also essential.

The development environment must permit an efficient work in the project - editing, compiling, building, debugging and testing. Parallel and distributed development must also be supported, because the same components are to be developed and maintained at the same time on different platforms. This requires the use of a powerful Configuration Management (CM) tool, and definition of an advanced CM-process.

The CM process support exists on two levels. First on the source-code level, where source-code files are under version management and binary files are built. The second level is the product integration phase. The product built must contain a consistent set of the component versions. For example, Figure 8 shows an inconsistent set of components. The product version P1-V2 uses the component versions C1-V2 and C2-V2. At the same time the component version C1-V2 uses the component version C2-V1, an older version. Integrating different versions of the same component may cause unpredictable behavior of the product.

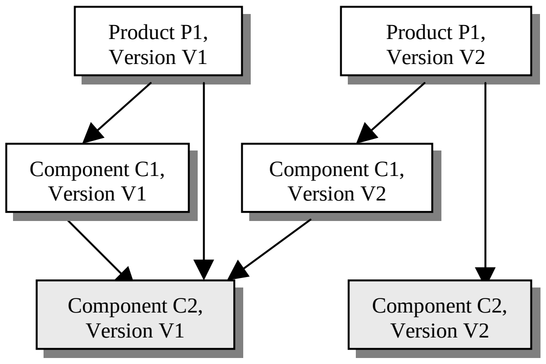

**Figure 8.** An inconsistent component integration.

Another important aspect of CM in developing reusable component is Change Management. Change management keeps track of changes on the logical level, for example error reports, and manages their relations with implemented physical changes (i.e. changes of source code, documentation, etc.). Because change requests (for example functional requirements or error reports) come from different products, it is important to register information about the source of change requests. It is also important to relate a change request from one product to other products. The following questions must be answered: What impact can the implemented change have on other products? If en error appears in one product, does it appear in other products? Possible implications must be investigated, and if necessary, the users of the products concerned must be informed.

The development environment designated Software Development Environment (SDE) [8] is used in developing Advant products. It is an internally built program package, which encapsulates different tools, and provides support for parallel development. The CM tool, based on RCS [9] provides support for all CM disciplines, such as Change Management, WorkSpace Management, Build Management, etc. SDE runs on different platforms, with slightly modified functions. For example, the build process is based on *Makefiles* and *autoconf* on Unix platforms, while Microsoft Developer Studio with additional *Project Settings* is used on Windows NT. The main objective of SDE is to keep the source-code in one place under version control. Using baselines, and change requests, the different versions of components are managed. The whole development process is complex and requires an organized and planned support, but it is unavoidable for an efficient and successful development of and with reusable components.

## 5 A New Paradigm -Standard Components

In recent years the demands of customers on systems have changed. Customers require integration with standard technologies and the use of standard applications in the products they buy. This is a definite trend on the market but there is little awareness of the possible problems involved. An improper use of standard components can cause severe problems, especially in distributed real-time and safety-critical systems, with long-period guarantees. In addition to these new requirements, time-to-market demands have become a very important factor.

These factors and other changes in software and hardware technology have introduced a new paradigm in the development process [10]. The development process is focused now on the use of standard and de-facto standard components, outsourcing, COTS and the production of components. At the same time, final products are no longer closed, monolith systems, but are instead component-based products that can be integrated with other products available on the market.

This new paradigm in the development process and marketing strategy has introduced new problems and raised new questions:

- The development process has been changed. Developers are now not only designers and programmers, they are also integrators and marketing investigators. Are the new development methods established? Are the developers properly educated?

- What are the criteria for the selection of a component? How can we guarantee that a standard component fulfills the product requirements?

- What are the maintenance aspects? Who is responsible for the maintenance? What can be expected of the updating and upgrading of components? How can we satisfy the compatibility and reliability requirements?

- What is the trend on the market? What can we expect to buy not only today but also on the day we begin delivering our product?

- When developing a component, how can we guarantee that the "proper" standard is used? Which standard will be valid in five, ten years?

All these questions must be considered before beginning a component-based development project. Josefsson [11] presents certain recommendations to the component integrator for use as guidelines: Test the imported component in the environment where it is to run and limit the practical number of component suppliers to minimize the compatibility problems. Make sure that the supplier is evaluated before a long-term agreement is signed.

The focus of development environment support should be transferred from the “edit-build-test” cycle to the “component integration-test” cycle. Configuration management must give more consideration to rune-time phase [12].

## 5.1 Replacing Internal Component With Standard Components

In the middle of the eighties, ABB Advant products were completely proprietary systems with internally developed hardware, basic and application software. In the beginning of the nineties, standard hardware components and software platforms were purchased while the real-time additions and application software were developed internally. The system is now developed further using components based on new, standard technologies.

During this development, further new components become available on the market. ABB faced this issue more than once. At one point in time, it was necessary abandon the existing solutions in a favor of new solutions based on existing components and technologies. To illustrate the migration process we the discuss possibility of replacing OMF and C++_complib with standard components.

Experience from these examples showed that it is easier to replace component if the replacement process is made in small incremental steps. Allowing the new component to coexist with the old one makes it easier to be backward compatible and the change will be smooth.

## 5.2 Replacing OMF with DCOM

Moving from a UNIX based system to a system based on Windows NT had serious affect on the system architecture. Microsoft components using a new object model were available, namely COM/DCOM [13]. DCOM has functionality similar to that of OMF and this became a new issue when DCOM was released. Should ABB continue to develop its proprietary OMF or change to a new standard component? The problem was that DCOM did not have all the functionality of OMF and vice versa. The domains overlap only partially as shown in Figure 9

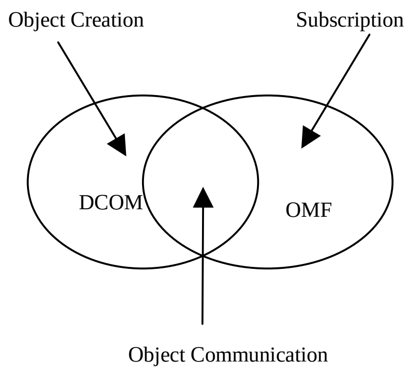

**Figure 9.** The functionality domains of OMF and DCOM do not overlap completely.

A subscription of data with various capabilities can be made in OMF, and this subscription functionality is not supported by DOCM. On the other hand, DCOM can create objects when they are required and not like OMF where objects are created before the actual use of them. Both technologies support object communication and in this area it is easier to replace OMF. with DCOM.

If the decision was made to continue with OMF, all the new components that run on top of COM could not be used, which would drastically reduce the possibilities of integration with other, third-party components. On the other hand, it would require considerable work to make the current system run on top of COM. This was the dilemma of COM vs. OMF.

To begin with OMF was adapted to COM with an adapter designated OMF COM aware. This functionality helped COM developers access OMF objects and vice versa. However, this solution to the problem using two different object models was not optimal since it added overhead in the communication. Nor it was possible to match the data types one to one, which made the solution limited. A decision was taken to build the new system on COM technologies with proprietary extensions adding the functions missing from COM. All communication with the current system was to be through the OMF COM. Adapters are very useful when a new component is to used in parallel with an existing one [14]. This solution makes it easy to remove the old OMF and replace it with COM in small steps over time.

### 5.3 Replacing C++_complib with STL

To switch from C++_complib to STL [5] was much easier because STL covers almost all the C++_complib functions and provides additional functionality. Still, much work reminded do since all the code using C++_complib had to be changed to be able to use STL instead. The decision was taken to continue using both components and to use STL whenever new functionality was added. After a time the use of old components was reduced and the internal maintenance cost reduced. In some cases in the same components both libraries were used, which gave some disadvantages, especially in the maintenance process.

## 6 Conclusion

The Advant OCS has been used as an example of a successful component based system and we have shown what it means to develop with components that fit into a large software system. A careful design and awareness of future demands on components are necessary to be able to integrate the existing system with new technologies. When Advant OCS was developed no one really though about Windows NT and ABB had to pay the price for that when it suddenly became clear that Windows NT would be the next operating platform. It was possible to move from one platform to another, but the cost was greater than if the design would have been more independent from the platform. We have shown certain problems with developing reusable components and given examples for this. The experiences from the development of Advant OCS has been that it is better to put more effort to create an open and extendable architecture than to rush the development focusing on only current technologies.

## 7 References

[1] Advant, ABB Automation Products, http://www.advantocs.com

[2] Sommerville I., *Software Engineering*, Addison-Wesely, 1999

[3] CORBA, http://www.corba.org

[4] International Electrotechnical Commision (1992), *Programmable Controllers Part 3, Programming Languages*, IEC 1131-3, IEC Geneva.

[5] Austern M., *Generic Programming and the STL*, Addison-Wesely, 1999

[6] OPC Foundation, http://www.opcfoundation.org

[7] Szyperski C., *Component Software*, Addison Wesely, 1999

[8] Crnkovic I., *Experience with Change-Oriented SCM Tools*, Software Configuration Management ICSE'97 Symposium, 1997, proceedings, Springer

[9] Tichy W., RCS - *A System for Version Control, Software and Practice Experience*, 15(7):635-654, 1985

[10] Aoyama M.: *New Age of Software Development: How Component-Based Software Engineering Changes the Way of Software Development*, 1998 International Workshop on CBSE

[11] Josefsson M., Oskarsson Ö., *Programvarukomponenter i praktiken – att köpa tid och prestera mer*, Report from Sveriges Verkstadsindustrier 1999

[12] Larsson M., Crnkovic I., *New Challenges for Configuration Management*, System Configuration Management Symposium, 1999, proceedings, Springer

[13] Box D., *Essential COM*, Addison-Wesley, ISBN 0-201-63446-5

[14] Rine D., Nada N., Jaber K., *Using Adapters to Reduce Interaction Complexity in Reusable Component-Based Software Development*, Proceedings of the fifth symposium on software reusability, ACM Press, 1999