The Emperor's Clothes Are Object Oriented and Distributed

Michael L. Brodie
GTE Laboratories Incorporated
40 Sylvan Road, Waltham, MA 02254, USA
Brodie@gte.com

Abstract
Distributed computing, and distributed object computing in particular, holds remarkable promise for future Information Systems (ISs) and for more productive collaboration between our vast legacy IS base world-wide. This claim is not new to those who have read research, trade, or vendor literature over the past eight years. GTE has made a significant attempt to benefit from this technology. We have found that it is currently considerably more difficult and less beneficial than the literature or our proponents would have had us believe. This chapter outlines challenges that we and others have faced in attempting to put objects to work on a massive scale. The challenges were confirmed in a world-wide survey that I conducted of over 100 corporations that are attempting to deploy distributed object computing applications based on technologies such as CORBA, DCE, OLE/COM, distributed DBMSs, TP monitors, workflow management systems, and proprietary technologies.

Distributed object computing has offered a vision, significant challenges, some progress toward a computing infrastructure, and some benefits. Whereas distributed computing infrastructure and its interoperability is critical, application interoperability is the fundamental challenge to users of distributed computing technology. More than 10 large corporations spend on the order of $1US billion annually addressing application interoperability. Although application interoperability is claimed to be the objective of distributed computing infrastructures, there has been little progress toward this critical ultimate requirement.

This chapter presents a view of distributed object computing from the vantage point of a large organization attempting to deploy it on the large scale. Requirements are presented in a distributed computing framework that is necessarily more comprehensive than anything currently offered by the distributed object computing vendors and proponents. A distributed computing framework is seen as having four parts:

- Distributed and Cooperative Information Systems
- Computing Environment
- Distributed Object Computational Model
- Domain Orientation

Relative to this framework, I outline GTE’s approach to distributed object computing, challenges GTE faces and faced, why it is so hard, alternative distributed object computing infrastructure technologies, and an estimation of how far we are from achieving the “interoperable” applications. This non-technical requirement has always been a fundamental challenge for software.

No, Virginia, there is no distributed object computing, yet.

1 The Challenge
Future computing hardware and software will be scalable, service-oriented, and distributed. That is, computing requirements, on any scale, will be met by combining cooperating computing services that are distributed across computer networks. Distributed object computing (DOC) is a critical component in this long-term view, particularly for Distributed and Cooperative Information Systems (sometimes called CoopISs). The current challenge is to develop an adequate long-term computing vision and a sensible migration toward that vision [BR95, BR96]. This, however, is a technology-centric view. A more business-oriented, and hence realistic, re-statement might be as follows: To efficiently run our businesses, we would like to deal directly with the business process, not ISs, to define, alter, and execute them. Ideally, business processes would be directly and automatically implemented by underlying information technology, which we currently refer to as ISs. Business processes cooperate or interact, often in complex ways. Hence, ISs must interact correspondingly. Hence, IS cooperation is one of our current key technical challenges.

1 An earlier version of this chapter appeared as Foundations of Intelligent Systems, 9th International Symposium, ISMIS ’96 Zakopane, Poland, June 1996, Proceedings (Eds. Z.W. Ras, M. Michalewicz), Lecture Notes in Artificial Intelligence, Springer-Verlag

2 © 1997 GTE Laboratories

Cooperation is the high-level requirement. Semantic and application interoperability are terms that refer to lower level (e.g., implementation) aspects of the problem.

This chapter presents an evaluation of progress toward the above goals from the point of view of a large “end-user” organization that is attempting to deploy DOC applications on the large scale. Each viewpoint has its biases. This chapter does not share the biases of a DOC technology vendor or consortium, an academic, or a consultant. End users, more directly than the others, pay for and live with the resulting ISs. Specifically, end users are responsible for the entire life cycle of an IS. Characteristic users of DOC technology, end users must, themselves, compose a significant number of component parts to achieve their requirements.

We are currently at the beginning of a 20-year cycle, at the end of which some version of DOC will be the technology of choice for ISs. However, from our current status, significant intellectual and behavioral change is required. It may take 5–10 years for the technology to become complete and robust. Methodologies, tools, education, and the shift in the user base to the new technology may extend that period to 20 years. This is similar to the 20-year shift to relational database technology, except that DOC has a comprehensive scope (i.e., all of computing) and is orders of magnitude more complex.

The chief architects of 12 successful large-scale DOC applications all agreed that DOC is considerably more complex than previous approaches. DOC may be so hard because it requires a philosophical shift. Theories in computer science are rational (e.g., deductive) and form the basis of programming languages and IS design. ISs are typically designed in a top-down fashion by means of functional decomposition which came from IBM’s 360 project. Object-oriented ISs require the philosophical approach on the other side of the dialectic, namely, empiricism. Distributed and cooperative ISs will be composed, bottom up, from existing or newly created components. The rational and empirical approaches are fundamentally different and require different ways of thinking. This age-old dialectic was initiated by René Descartes [DEC], who introduced rationalism in 1637, and by John Locke [LOC], who introduced empiricism in 1690. Immanuel Kant [KANT] attempted to mediate between the two views. In the Regulative of the Capability of Composing systems from components (e.g., reuse) is a basic premise of distributed and cooperative ISs, and computer scientists are simply not used to thinking about ISs empirically. We do not have empirical theories or tools to assist us with this approach. Our lack of familiarity with and tools for such an approach may lie at the heart of the difficulty of the paradigm shift and explain why reuse has been so elusive. But, then, I digress.

DOC technology is in an early and immature phase. This can be seen in terms of the technology adoption life cycle, defined by Geoffrey Moore [MORE] (see Figure 1). Early adopters, called innovators and visionaries, are change agents who get rewarded for instituting change to get a jump on the competition through radical changes, often called improvements. Later adopters, called pragmatists, conservatives, and skeptics, need technology to work well in their existing technology base, which they want to enhance, not overthrow. Between the early adopters and the later adopters is a chasm. The chasm represents the challenges in making the cost/benefits obtained by the early adopters acceptable to the later adopters. The chasm also represents a major change in the customer types due to their radically different motivations. For a new technology to be successfully adopted, it must bridge the chasm, since the marketplace can be captured by the later and not the early adopters. In general, 90% of advanced technology goes down the chasm, at least in the form that it was originally offered. For example, expert system technology went down the chasm, while expert system methodology went into widespread use in a variety of forms, but not in expert system engines.

DOC benefits are often discussed. The costs are not. The DOC vision claims to address current business goals, including improvements in time to market of the target product or service; development, deployment, and continuous operations costs; flexibility to accommodate constant changes in business processes, policies, and practices; quality; and lowering risk. DOC is also claimed to provide means to overcome problems of previous technologies, including technical (e.g., software crisis [BR96]), managerial, and administrative. Specific technical objectives include reuse, plug and play, component assembly, workflow-enabled business processes, and service or component orientation. The ultimate technical goal is interoperability at all levels and across the entire life cycle. DOC technology is a specific sub-case of client/server computing. In the late 1980’s, client/server was claimed to provide orders of magnitude improvements in price/performance as well as to address other major information problems. By 1997, client/server has not met the claims. Its use is currently at a minimum 20%–30% premium over the
mainframe systems it was claimed to annihilate. As with DOC, this may be a temporal issue. As DOC and client/server technology matures, the benefits and cost savings may be realized. Meanwhile, CMOS technology is making mainframes scalable and within 30% of client/server hardware price/performance levels.

Figure 1: Moore’s Technology Adoption Life Cycle

DOC technology is in the early adoption stage and is rapidly facing the chasm. DOC technology promoters must now focus on satisfying the requirements of the later adopters. They must address the real state of DOC technology, which our experience and survey suggests is as follows. DOC is inherently hard and is not understood. The relevant theory and technology is immature but evolving rapidly. There are rare successes that are due to genius chief architects and their staffs. The claimed benefits (e.g., reuse, productivity) are very hard to realize. Most DOC technology does not meet industrial-strength requirements. Hence, it is not ready for prime time. Since there is currently no dominant DOC infrastructure choice (e.g., CORBA, OLE/COM), how do you architect or plan a DOC application? This chapter outlines some of the requirements of the later adopters, based on the experience of an early adopter.

Not surprisingly, there is a pattern here if you replace “DOC” with any “promising advanced computing technology” in the past 20 years [BR96]. To address the current challenge of developing an adequate long-term computing vision and a sensible, incremental migration toward that vision, we must act differently than in the past. What is a reasonable time frame for the transition? What are reasonable increments? [BR95]

2 Distributed Object Computing Framework

An end user requires a complete distributed computing framework with which to guide an IS through its life cycle. The plug-and-play nature of DOC means that no single vendor provides such a framework since they all produce component parts. Hence, end users of DOC technology must define their own frameworks. There are at least four parts or models that constitute such a framework.

- Distributed and Cooperative Information Systems
- Computing Environment
- Distributed Object Computational Model
- Domain Orientation

An IS designer must have a conceptual model of distributed and cooperative information systems. Such a model (see Figure 2) could consist of a business process that solves a specific business problem. The business process can be expressed in terms of a workflow which, in turn, invokes business services which execute workflow tasks. Previous-generation architectures were complex and rigid. Next-generation architectures will support the execution time binding of a business service to the workflow task. There may be thousands of such workflows per second, which may mean that the architecture for a given workflow exists only for a nanosecond compared to forever!

Figure 2: Next-Generation Information System: The Nanosecond Architecture

The computing environment consists of a distributed computing infrastructure and a complete life cycle support environment. The infrastructure provides the services required to support the execution of workflows, the dynamic invocation of business services, and the distributed object space that supports the software components with which the business services and workflows are implemented. These are called CORBA services® in the terminology of the Object Management Group® (OMG®). The life cycle support environment provides all the necessary tools to support a comprehensive life cycle for ISs—from conception to cradle to grave. Some of these tools are included in what OMG terms CORBA Facilites®. Ideally, these tools will enforce application-specific or domain-specific standards for services at all levels.

The distributed object computational model refers to the object model that underlies the computing environment. Rather than being a single object model, it will be a family of interoperable object models, each member of which has a specific role in the computing environment. Due to different computational and programming requirements, there would be different object models for infrastructure services (e.g., persistence service), business services (e.g., telecommunications billing), and applications development (e.g., workflow services, component assembly).

Domain orientation concerns the tailoring of business services with respect to application domain requirements and standards to meet the unique requirements of the domain as well as application interoperability. Domain orientation involves not just standards within one domain (e.g., telecommunication billing) but also across multiple domains, since few business processes or value chains exist solely within one domain. For example, a telephone call involves billing, routing, possibly advanced services, maintenance, and testing, to mention a few. Another example is that most domains contain customers.

In the DOC context, domain orientation involves terminology, ontology, domain (object) models, object/systems interface specifications, and frameworks. Application interoperability requires that two applications mutually understand the messages that they exchange. At least with respect to those messages, they must share (e.g., map to) a common terminology, ontology — definitions of the essential elements of the shared domain; and business processes model — definitions of the way business is conducted in the domain. These shared models can be defined in terms of domain-specific object models which can be standardized in terms of interface specifications for classes from which the IS is composed. A framework for a given application domain is the life cycle support environment (i.e., tools and computing artifacts such as class libraries and interface definitions) that supports and enforces the relevant domain standards. Frameworks are developed by specializing a computing environment with the standard object models of the domain, as manifested in class libraries and interface specifications.
A basis for application interoperability can be defined across application domains by means of families of interoperable object models (see Figure 3) and corresponding interoperable domain frameworks. For example, a generic object model family can be tailored to specific domain requirements, further to support their unique requirements. This results in an interoperable object model family or hierarchy.

The above domain standards could be enforced, to a degree, by a computing environment that supports the entire life cycle of an IS. The life cycle support environment would ensure that the appropriate domain standards are enforced at each stage of the life cycle (e.g., design, implementation, testing, and operation). These bridges include object services such as asynchronous operation, multi-threaded dispatch, and naming.

Currently, semantic interoperability (i.e., the ability of systems to interoperate, e.g., by exchanging messages that are mutually understood to achieve shared goals) is achieved primarily by the following:

- Programmers and designers who must understand system A enough so that when they are programming system B they can design or program meaningful interactions with A (i.e., programmer discipline)
- Systems integrators who build bridges between systems to support interoperability

This low level of solving the problem is labor-intensive and error-prone. Semantic interoperability should be solved at the highest level possible. For example, if a hierarchy of terminology, ontologies, and domain models, and above all, an underlying shared theory is used, this supports a much higher level of semantic interoperability, e.g., of Web services. This permits flexibility of independent ISs but raises complexity, the current greatest challenge of IS cooperation.

Consider an example of an underlying, shared theory to enable cooperation between independently developed computing artifacts. Two-phase commit protocols and serializability theory were developed for database applications (e.g., in CORBA) and similar algorithms exist in other environments (e.g., in Web services). The shared theory (e.g., the transaction model) is used as an interface for the development of cooperating artifacts that have different internal models (e.g., different database models). This permits cooperation, but raises complexity.

A DOC technology end user requires a comprehensive DOC framework, as described above. Over the past 20 years, the relational database community has developed such a framework, which has resulted in the greatest integration of existing applications (e.g., through SQL). The relational computational model has been defined and extended to support new application domains, including E-commerce. The relational model has also been extended to support new application domains, including E-commerce.

In summary, the approach to semantic interoperability involves the development of a comprehensive framework that supports all application domains, including E-commerce. This framework provides a common foundation for the development of cooperating artifacts, which can then be integrated into a single, coherent system.
accepted in national and international standards. There has been little domain orientation in relational
database technology. However, it has been discussed in that context for approximately 20 years due to the
application interoperability challenges that naturally arose when applications could communicate so readily
via a schema. Although it is not stated in terms of domain orientation, the biggest frontier of challenges and
related technology advances in database technology lies precisely in this area. The capability of
object/relational database management systems to deal with domain-specific data types is the discovery of
this new frontier of domain orientation in the database world.

3 GTE’s DOC Experience

The decision to use DOC as a fundamental technology of ISs is a complex and expensive one in GTE,
due in part to its size. GTE Telephone Operations (Telops) is the largest US local exchange carrier. It is the
world’s 4th largest public telephone company and has been reported in the Wall Street Journal as the 44th
largest public company in the USA. It supports 23 million telephone lines, has 100,000 employees, and has
annual revenues of $22US billion. To support this business, GTE’s information technology is large scale.
The annual information technology expense is in excess of $1US billion. There are approximately 1,500
ISs and over 150 terabytes of data. The legacy ISs are highly interrelated (see Figure 6).

In 1993, Telops began the definition of an initial Telops computing environment, as defined in Section
2, and to specify the constituent technologies and services (see Figure 8). Initially, the computing
environment would consist primarily of non-DOC technology, but would be defined in DOC terms. The
non-DOC or legacy technology is included in the architecture via a gateway illustrated in Figure 8. Plans
were begun for architecture migration and for corresponding ISs and data servers. Unlike previous
technologies, distributed computing encourages resource sharing across applications. Hence, the IT
organization and decision-making procedures had to be redefined so that stakeholders across application
(e.g., organizational) boundaries could cooperate to achieve a shared technology base. This organizational
change was as significant as the technology transition.

The long-term computing environment assumes a model of distributed and cooperative ISs, as defined
above and illustrated in Figures 2 and 7, driven by business processes defined in terms of workflows and
business services. The work of defining the computing environment and specifying the constituent services
was challenging. It was originally assumed that vendors would provide DOC infrastructures. Ideally, there
would be a range of vendor products from which to generalize and select. However, this was, and is still,
far from the case for DOC technology. Due to the requirement to provide high-quality, robust, reliable
products and services, considerable attention was given to a comprehensive life cycle. The minimal
services provided in most DOC vendor products focused mostly on the initial 15% of the life cycle, namely
analysis, design, and development. As a result, GTE focused on the missing services, including class
libraries, repository services, comprehensive methodologies, tool support across the life cycle, run-time
services, testing, and continuous operations support (i.e., the part of the life cycle that consumes 85% of the
total IS costs).
A challenge, which can be greater than those of DOC technology, is that of migrating from the existing computing environment and applications base to the corresponding ones for DOC. This is illustrated in Figure 9. This is remarkably challenging not only due to the technical challenges [BR95] but maybe more so due to the business and organizational realignments [ORL]. The challenges and approaches to their resolution as covered in [BR95] are not addressed here. Considering the scale of the GTE environment, you can see that key requirements, defined in [BR95], include the following: ISs cannot be stopped during the migration; the migration must be incremental; the migration will take many years; the computing environment must be designed to support migration; continuous migration will be a way of life; and sequencing of migration increments for shared data and programs requires complex configuration management, to mention a few. Large-scale migrations will proceed incrementally. Hence, the organization will be in the intermediate stage between the source and target, as illustrated in Figure 10, in which the surviving parts of the old environment, business processes, etc., must coexist with the operational parts of the new environment. The critical requirement is that this curious mixture meet the then current business requirements of the organization.

In 1994, GTE began the Carrier Access Billing System (CABS II), its first distributed object computing project, as a joint development between GTE and Ameritech. CABS II was begun after an extensive study to ensure that such a system was achievable. For example, the DOC infrastructure (i.e., TC3I’s Object Services Package® (OSP®) — the logical equivalent of an OMG Object Request Broker® (ORB®) plus object services) — was selected based, in part, on the fact that it had been used in several large-scale DOC applications that had been successfully deployed for several years. In addition to OSP, the technical infrastructure included UNIX servers and clients, PC clients, and SQL relational DBMS.

A domain-specific distributed object computational model (i.e., a billing object model) was developed for CABS II as well as a corresponding domain specific (i.e., billing) distributed object framework. Figure 11 illustrates the evolution, through time, of the movement from a domain model that incorporates everything above the operating system level. Introducing a general-purpose application framework reduces the complexity of the CABS domain model. The CABS domain model is further simplified by introducing a billing domain specific application framework. The resulting CABS domain model consisted of a collection of basic billing object classes. The basic billing object class library assisted significantly in the design and development of CABS II. The rest of the object model family and the corresponding frameworks helped to establish a base for application interoperability across applications and domains. The

CABS II chief architect claims that the CABS II billing object model and framework was one of the major advantages of the CABS II project. Using the framework, those developing business solutions work almost entirely with billing objects, not with a general-purpose object model. What is more, all business solutions in CABS II use the same billing objects.
CABS II is a large-scale DOC application, as measured by the statistics in Table 1. These August 1995 numbers have since increased significantly, thus indicating our inability to adequately estimate DOC systems characteristics at the outset. In Table 1, “Message” means an object instance service invocation message plus an object instance service response message (if any).

Table 1: CABS II Sizes

<table>
<thead>
<tr>
<th>Category</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Business/domain classes</td>
<td>250</td>
</tr>
<tr>
<td>Implementation classes</td>
<td>4,500</td>
</tr>
<tr>
<td>Class instances (estimate)</td>
<td>10^6</td>
</tr>
<tr>
<td>Data</td>
<td>3 x 10^6</td>
</tr>
<tr>
<td>Nonusage data</td>
<td>10^7</td>
</tr>
<tr>
<td>Servers</td>
<td>50–75</td>
</tr>
<tr>
<td>Clients</td>
<td>1,000s</td>
</tr>
<tr>
<td>TP rate</td>
<td>2,200/second</td>
</tr>
<tr>
<td>Message rate</td>
<td>6,100/second</td>
</tr>
</tbody>
</table>

4 DOC Deployment Status: A Survey

Although I did not start out to do so, I conducted a survey of DOC deployment around the world. Initially, in support of GTE’s DOC program, I initiated an information exchange between GTE and a few organizations of comparable size with a comparable commitment and investment in DOC. Following the popular notion from Jack Welch, CEO of GE, we tried to identify and possibly adopt the best DOC practices. Much to my surprise, I found few, if any, comparable organizations with comparable experience. Indeed, two such exchanges ended with the other organizations acknowledging that they were years behind us. A second motivation was to confirm the experiences and approaches followed in CABS II. A third motivation came from GTE’s end-user membership in OMG. In OMG meetings, I had heard so little from other end users in terms of technical requirements and challenges. I experienced acknowledgment but little action toward or apparent understanding, by the vendor-dominated organization, of end-user requirements. Specifically, there appeared to be no substantive work in support of application interoperability. This may have been due to the lack of end-user experience with large-scale DOC applications. A survey might shed light on these questions. Finally, I was very interested from the point of view of GTE’s nine-year research effort into these topics [DOC] as to the state of technology vs. the state of research. What are the pragmatic research challenges? The survey was fruitful in each of the above areas.

The survey was informal. I contacted any organization that was an OMG end user or for which there was a rumor or claim that they were attempting to deploy DOC applications (e.g., customers of DOC infrastructure products such as ORBs). A survey form was used which evolved as I learned more about DOC deployment issues. The survey was sent to over 100 end users from whom I received 61 responses on 201 DOC applications in various stages of deployment, as indicated in Table 2.

DOC applications can be built using a wide range of combinations of alternative infrastructure technologies, as indicated in Table 3. Table 3 lists the percentage of systems that used a particular technology. Most systems use more than one technology. These technologies include OMG CORBA-compliant ORBs, Microsoft’s OLE/COM®, OSF/DCE® products, a wide range of database management systems (SQL, object-relational, object-oriented, and several flavors of distributed DBMSs), transaction processing monitors, workflow managers, messaging backplanes, and proprietary DOC infrastructures (e.g., TCSF’s OSP®; SSA Object Technology’s Newi®; and NeXt’s Portable Distributed Objects®, NextStep®, and OpenStep®).

Table 2: Survey Results

<table>
<thead>
<tr>
<th>Category</th>
<th>CORBA</th>
<th>Proprietary</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Large scale</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Deployed 1–3 years</td>
<td>1</td>
<td>13</td>
<td>14</td>
</tr>
<tr>
<td>Deployed, not confirmed</td>
<td>25</td>
<td>5</td>
<td>30</td>
</tr>
<tr>
<td>To be deployed 0–3 years</td>
<td>6</td>
<td>59</td>
<td>65</td>
</tr>
<tr>
<td>Limited scale/features, deployed</td>
<td>6</td>
<td>16</td>
<td>22</td>
</tr>
<tr>
<td>Prototype/evaluation/pilot</td>
<td>17</td>
<td>53</td>
<td>70</td>
</tr>
<tr>
<td>TOTAL</td>
<td>55</td>
<td>146</td>
<td>201</td>
</tr>
</tbody>
</table>

Table 3: Major Infrastructure Technology Used

<table>
<thead>
<tr>
<th>Major Infrastructure Technology</th>
<th>Applications Surveyed (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proprietary</td>
<td>80</td>
</tr>
<tr>
<td>DBMS</td>
<td>50</td>
</tr>
<tr>
<td>TP monitor</td>
<td>30</td>
</tr>
<tr>
<td>CORBA</td>
<td>20</td>
</tr>
<tr>
<td>OLE/COM</td>
<td>10</td>
</tr>
<tr>
<td>Workflow managers</td>
<td>10</td>
</tr>
<tr>
<td>DCE</td>
<td>0</td>
</tr>
</tbody>
</table>

All successfully deployed large-scale DOC applications that I found ran on proprietary DOC infrastructures. DBMSs are the obvious choice for data management in any infrastructure. However, distributed DBMSs may become the primary DOC infrastructure for some applications. Although distributed DBMS products now meet many distributed computing requirements, they are just beginning to be deployed. TP monitors are in widespread use (e.g., most credit card and ATM transactions). They form the backbone of many distributed computing architectures. During the survey, I found no confirmed large-scale applications deployed on a CORBA-compliant ORB. After the survey, I found one modest sized ORB-based deployed system (i.e., The British Immigration Service’s Suspect Index System, which runs on ICL’s DAI®). The ORB-based applications were of limited scale (the largest involved five servers), of limited features (e.g., distribution not used), or were not deployed. Although OLE/COM is in widespread use on desktops, network OLE, the corresponding DOC infrastructure, including Microsoft’s Component Object Model (COM), was not yet available. I heard claims of over 100 large-scale applications deployed on workflow management systems (WFMS), but was able to find only 10, and in those, the WFMS did not seem to be the critical infrastructure element. I found no DCE-based applications. Following the survey, I found five modest-scale DCE-based deployments and indications that there are likely to be many more. The clear winner was “combination.” Table 3 adds up to 200%, indicating that most applications use a combination of infrastructure technologies. After interoperability is possible, combinations of technologies will likely be the dominant infrastructure. Which will be the component infrastructures, and what degree of heterogeneity will be practical?

Building a DOC application from scratch in a DOC environment can be considerably easier than integrating legacy applications using a DOC technology. I found most DOC applications to be pure, and most of the rest to be legacy IS integrations (Table 4). The challenges include dealing with the complexities...
The survey found that CABS II was the largest DOC application, in terms of the statistics listed in Table 5. CABS II is not in production as of December 1997. CABS II was larger than Texas Instruments’ TI WORKS®, a suite of applications for running a semiconductor CIM fabrication plant. Based on the information gathered in the survey, TI WORKS was the most successful large-scale DOC application. It does not use an ORB. At the time of the survey, TI WORKS was about to release some small components (e.g., configuration management) of TI WORKS into production, with a plan for major components to be released at the end of 1996. CABS II was larger in all categories, including the number of classes and object instances, the latter by four orders of magnitude. However, this scale is considerably smaller than current large-scale mainframe-based ISs. The “Other” column in Table 5 refers to the 11 large-scale DOC applications that I found which were smaller again than TI WORKS.

I surveyed the respondents on several issues of significance to GTE’s DOC effort. I found that 92% of the successful large-scale DOC applications used asynchronous (e.g., queued) messaging, while 40% of the small-scale ISs and prototypes used synchronous (e.g., RPC) messaging, such as provided in OMG’s CORBA.\(^4\) The reasons for asynchronous messaging included robustness (e.g., recoverable queues), performance, scalability, non-blocking behaviour, and flexibility (e.g., via queue management). I found only three organizations that were working on an enterprise-wide DOC architecture and three that were working on smaller architectures for divisions or business processes. Four organizations had formal class libraries; four were building ontologies or domain models; and four were developing frameworks, as defined above (including CABS and TI WORKS).

I found only three applications that were built on infrastructures that supported logicalphysical object separation. For more than 20 years, DBMS technology has supported a degree of data independence. Programs are insulated from changes in the physical structure, since they deal with logical schema entities which are mapped by the DBMS to the underlying physical representation. Hence, the physical DBMS can be optimized without impacting programs. In all but three DOC applications, logical and physical object representations are identical. This means that changes to objects’ logical or physical representation require changes to the entire system. This is practically infeasible at the scale of CABS II.

There were a few obvious conclusions from the survey of DOC deployment. First, for such a rapidly evolving technology, the situation changes constantly. The premise of this chapter is that DOC technology will be the base of future ISs. However, the current state, at the time of the survey, indicates that considerable maturation is required. Second, there are lots of object-oriented applications, but very few true DOC applications. This survey was not about object-oriented applications; it was about DOC applications. Third, almost all successful DOC applications were based on homogeneous, proprietary infrastructures and were not readily interoperable with other applications, the antithesis of the DOC vision and of OMG claims, or at least goals. Fourth, DOC is inherently very hard and lacks general solutions and tools (i.e., they must be developed by highly skilled staff). Fifth, there are a few success stories (e.g., HOSIS, TI WORKS), and their success is due largely to the highly skilled staff. Sixth, there may be more significant successful DOC projects that I did not find or which did not respond. For example, the financial community claimed 30 successful large-scale DOC applications, 25 based on CORBA ORBs (see Table 2 “Deployed, not confirmed”). However, they were unwilling to provide the details to substantiate the claims. I did obtain details of one such claim. If it was deployed before 1995, the “details” consisted only of a single page (i.e., copies on photocopied machines did not communicate). Finally, claims of success cannot be taken at face value. I followed up on a few public claims of and awards for DOC successes and found them to be either unconvincing, unsubstantiated, or significantly less than claimed. For example, a high level of reuse was claimed for a large-scale deployed DOC application that was built in a partnership between two organizations. I found that one partner, an end user, did not get any reuse. The other partner, a solutions vendor, got considerable reuse since they had sold the system to multiple customers.

The survey seems to suggest the following lessons. First, the major challenge remains the development of an adequate long-term computing vision and a sensible migration toward that vision. Successful large-scale DOC application projects devoted considerable effort to developing a model of distributed and cooperative ISs and a computing environment (e.g., architecture beyond the current application), and planned for a long-duration migration to the vision (e.g., one major application at a time). Second, missioncritical production applications should be pursued using DOC only if the requirements clearly demand it, and then only with great care. Third, small non-mission-critical pure DOC applications are the easiest, while the obvious near-term win, legacy IS integration, is considerably harder. The conventional requirement will be for a substantial mix of both, and that is the hardest type of application to build. Fourth, DOC infrastructures are being developed as products and standardized (e.g., in OMG and OSI’s DCE) apparently without having been tested on real DOC application requirements. Indeed, there are few in existence. Finally, the high risk involved in DOC application development and deployment requires explicit risk management. So, how should you architect and plan that system today for delivery in three to five years?

5 Industrial-Strength DOC Requirements

Based on our experience and on the survey, I identify, in this section, a number of requirements that DOC technology must satisfy to meet the needs of large-scale industrial applications. The requirements are given with respect to the distributed computing framework introduced above. As OMG is one of the world-wide foci of DOC technology development, many of the requirements are given with respect to the current state of OMG technology. However, the comments can apply equally to any DOC technology [MSFT]. Microsoft is also a major focus of DOC technology development. Unlike OMG and vendors of OMG-compliant products, Microsoft has existing products and less than 500 organizations in the decision

\[^4\] In 1Q96, OMG will consider an asynchronous messaging service, but it may not be a first-class citizen with its RPC-based service.
Indeed, they have significant products on the market (e.g., Microsoft® Windows NT® operating system, ActiveX®, Active Server® and the Microsoft® Transaction Server—formerly known by its codename, “Viper”). So ... at some specifics to illustrate the point and, it is hoped, to encourage effort toward fulfilling end-user requirements.

Industrial-strength applications require that all the pieces be in place, from the hardware up to the end-user applications and throughout the entire life cycle. A comprehensive distributed computing framework is missing and ... proceed the technologies that will support them. But, then, it’s never gone that way before, and look where we are today!

Industrial-strength applications are often built with large project staffs. The distributed object computational model that the staff uses must be complete and at a level appropriate to the problems being solved. The OMG ... object model. There is no high-level programming model for CORBA-based application development. Indeed, the services and facilities, come from disparate groups or individuals. Hence, the OMG object model, including the services and facilities provides the OMG technology user with a non-uniform, too low-level model, as indicated by the wavy line.

Figure 12: Distributed Object Computational Model

DOC Programming Model

Object Services

Objects and Messages

DOC programming environments such as NextStep, SSA Object Technology’s Newi, Forté’s Forté®, and TI’s Composer® provide more complete and higher level computational models required for industrial-strength ... OMG concept of profile is required to support domain-specific object models but is not yet adequately defined.

As this book went to press (late 1996), Microsoft released Microsoft Transaction Server, for which it claimed [MSFT] “Microsoft Transaction Server lowers server development costs up to 40%. Developers use Microsoft ... the problems mentioned above, and, if true, indicate a significant advance toward the maturity heretofore lacking in DOC.

Industrial-strength applications require a comprehensive life cycle, from inception to design, development, deployment, evolution, and ultimately termination or replacement. There is no widely accepted life cycle for ... activetopics within the OMG. However, this paper is not addressing the future. It is concerned with what can be used now!

There are other problems with the DOC infrastructure that are related to open research problems and that pose significant challenges for large-scale industrial applications. For example, OMG technology provides several ... should be represented as objects or in the basic representation of the DBMS. A related problem is that you may wish
to have persistence, query, and transaction services over all objects in the CORBA DOC environment. However, these services are provided over those objects that reside in a component that supports the service. This will not likely include components other than DBMSs and TP monitors for some time. This means that providing those services will mean crossing from the CORBA DOC environment and type system to that of the DBMS and TP monitors. This will generally mean translating between object-type and non-object-type systems. Another performance hit.

Developing DOC environment problem concerns one of the great successes of OMG, the OMG IDL® (interface definition language). OMG IDL is being adopted widely, independently of, or in anticipation of, the success of CORBA. Hence, IDL is becoming the vernacular API, the interface specification language of many, many systems. Since IDL can’t be all things to all people, it is seen, specifically in my survey, as very limited. Each systems project wants to extend IDL for its own requirements. Unfortunately, many variants of IDL are now evolving.

Our experience with respect to ORB products was confirmed by the survey. These products are at an early stage of development and are incomplete, just as CORBAServices and CORBAfacilities specifications are incomplete. Most ORB products do not support the minimal adopted CORBAServices and CORBAfacilities and may not for some time. In addition, by mid-1996, large-scale industrial-strength applications push the limits of all ORB products with which we or the survey respondent had experience. They did not meet requirements for robustness, scale, and reliability. I am aware of no ORB that supports adequate means of testing, quality assurance, appropriate metrics for sizing and tuning, or monitoring and maintenance (e.g., performance tuning — recall the lack of logical-physical separation). What serious organization would go to production without these facilities? As stated earlier, most CORBA products lack an adequate asynchronous queued messaging service as a first-class citizen with RPC. Some of these problems are overcome by proprietary products. For example, Forte provides a wonderful function called the “rolling upgrade,” which permits client applications to be upgraded from one version to another while the server continues to operate from a single point in the distributed system. Forte wants to have a CORBA-compliant environment or even an application on a proprietary product or even a CORBA-compliant product augmented by many proprietary services built either by you or the ORB vendor, awaiting OMG standardization? GTE decided firmly against such a risky strategy.

Synchronous versus asynchronous messaging is a key issue. Let me speculate in order to illustrate a potential process of maturation and evolution. We are at the beginning of the message-based computing paradigm in which we will be required to understand more deeply the nature of communication protocols and the requirements for communications by the increasingly large number of applications with increasingly complex requirements. Synchronous and asynchronous messaging are two ends of a spectrum, that blended may be more than one dimensional. As we better understand messaging requirements, we may produce a spectrum of choices for communication protocols in which designers can specify what combination and degree of properties that they want from the communication protocol and the system will automatically generate a corresponding protocol somewhere along the spectrum. Further, the system may be able to optimize the choice. As the system is operational, different communication loads and behaviours could be monitored on the system could alter the communication protocol or the protocol in order to meet optimization criteria set by the designers. This would require that programmers not specify any specific of the protocols so that those specific are not embedded in the program, thus permitting the system to optimize as required (like relational queries). Compare this to the complex programming requirements to use the CORBA RPC mechanisms.

Finally, the fundamental requirement of industrial-strength, enterprise-wide interoperable applications is interoperability. Comprehensive interoperability involves interoperability across the entire life cycle. All artifacts produced during the life cycle should be accessible, in principle, by all tools. All tools should be able to interoperate with others, again, in principle. Interoperability is required from the bottom to the top. At the bottom, there is hardware platform interoperability which is “vendor hard.” It is entirely within the capabilities of the platform vendors to resolve the problem. At the next level, infrastructure interoperability is “Turing hard.” Whoever solves the problems of interoperable object models and distributed object computing services and facilities should be awarded the Turing Award. It is a very significant challenge. However, interoperability at the next level, application interoperability, is “Nobel hard.” A solution here should solve the problem. The next section concludes this chapter by illustrating this challenge, indicating its significance, and emphasizing that it is not a technical issue. Indeed, it is a core problem at the interface of computing and real life. It raises, for me, moral and ethical issues such as: What are the limits of technology? To what extent can we genuinely represent real-world (e.g., business) activities in a computer and rely on the system to replicate or become the real-world manifestation of the desired function? This chapter does not pursue these deeper problems. I mention them here to raise the more pragmatic question of what should we expect of DOC as a basis for running our businesses, and can we trust the claims of DOC proponents? To what extent can they verify that their claims are true and reliable since they may influence people to deploy DOC technology in mission-critical contexts not only where business and trade is involved, but where human lives may be at stake?

6 Toward Industrial-Strength, Enterprise-Wide Interoperable Applications

In the period 1913–1915, Niels Bohr, the Danish physicist and Nobel laureate, published the papers that defined the theory of atomic structure, for which he received the Nobel Prize in 1922. The significance of his theory of the erratic changes in energy levels of electrons circling the nucleus was understood almost immediately by physicists world-wide. Within a few years, Niels Bohr’s ideas, one man’s ideas, had helped to evolve man’s understanding of the atom and of elementary matter. This was possible, in part, because physicists world-wide shared a common domain orientation, as defined in Section 2, for elementary particles. There was a common terminology, a shared ontology (i.e., the basic concepts of particle physics), and a number of shared domain models (e.g., Rutherford’s nuclear model of the atom).

The shared domain models were standardized in mathematical models (analogous to interface specifications of object models) and placed in frameworks (i.e., the larger mathematical models of physics, such as quantum mechanics). The shared domain orientation permitted physicists around the world to cooperate (i.e., interoperate). The shared domain orientation in physics was the result of hundreds of years of science, at least back to Sir Isaac Newton (1643–1727). The process that created it was that of science itself. Now, although there are many differences and constant attempts to change and improve the domain orientation of physics, any two physicists can cooperate based on a mutually shared domain orientation. In 1997, this is being pushed from physics to philosophy and psychology, as the domain orientation in physics is moving more and more from the conventional, particle view of physics to the wave theory.

Following the principles of component orientation motivating DOC technology, consider the creation of a telecommunications billing system from components. The components may be entire subsystems (e.g., a rating system, an account management system, a bill generation system) or one or more class libraries of billing classes (e.g., customer, bill, line item). The use of these components together to produce a simple billing system requires application interoperability. Each pair of components must have a shared understanding of the objects (e.g., functions and data) involved in any messages that they exchange. Of course, it is more complex when a communication involves more than two components. Also, a deeper understanding (e.g., of objects that they do not exchange or the business process within which they participate) may be required. However, it is sufficient for this discussion to restrict our consideration to the messages exchanged by two components, the minimal application interoperability requirement.

Mutual understanding of objects in exchanged messages requires a shared domain orientation. The components must share or be able to map to a common terminology. To the degree that it affects their behavior, they must share or be able to map to a common domain. The challenge involves defining mutually agreeable
terminologies, ontologies, and domain models. How long will it take to achieve such agreements between thousands of telecommunications companies in countries all over the world, each with different cultures, economic models, etc.? A billing model can be verified, but with considerable difficulty, especially when it is undergoing fundamental change.

Let's consider shared ontologies and common object models as a basis for the illusive and much claimed feature of object-orientation, re-use. Re-use is not a technology issue so much as a standards issue, as I will show... to scoop up horse manure faster and faster as the numbers of horses increases rather than solve the problem at the source.

Successful re-use can occur most readily in domains that are well understood and bounded. Examples include: operating systems, DBMSs, spreadsheets, and word processors. Indeed, these are widely re-used worldwide. The... semantics are bounded and well understood. Other aspects that facilitate semantic interoperability, hence re-use, are:

- **Existing domain-orientation**: The domain is widely understood with a standard definition, a widely accepted terminology, ontology, or domain model (e.g., as in the physics example above).

- **Market share**: A product becomes widely used, hence its terminology, ontology, and domain models become the basis for interoperability. The widespread use of them makes them a standard.

- **Attempts to gain market share**: Widely used products become a focal point for products that need to interoperate with them. Hence, vendors build the bridges themselves to facilitate interoperability of their products. For example, the introduction of ODBC provided interoperability with many of Microsoft's products which were de facto standards.

- **Modularity**: The domain in which the products are used is well enough understood that the functionality can be modularized in ways that are universally accepted. This permits the components (i.e., objects) to be... as a standard (e.g., spellers as components in all text-based systems, RDBMSs in all data-intensive applications).

So we can look at the ease and criticality of establishing a domain orientation to identify the likelihood of establishing general-purpose semantic interoperability, and hence, re-use in a given domain. Let's consider domain-orientation or re-use of telecommunications billing.

- **No market share**: There is no telecom billing applications that have market share. Hence, there is no motivation to semantically interoperate with it (i.e., adopt its domain orientation) to achieve re-use.

- **No modularity**: There is no widely agreed decomposition of the telecom billing domain that would encourage the development of re-usable components.

- **No existing domain-orientation**: There is not a widely accepted terminology, ontology, or domain model for telecom billing. Hence, semantic interoperability is simply not a realistic issue. All attempts at this will fail, unless the attempt is strong enough to establish itself as the standard.

To conclude this example, telecom billing is a real market opportunity for establishing world-wide standards since the market is vast (e.g., thousands of telecoms worldwide each require billing as a mission-critical function). Hence, we have to establish the ease, criticality, and feasibility of domain orientation in the domain in which you are working.

A large number of standards bodies or consortia are attempting to create domain orientations. A brief search of the literature and the World Wide Web uncovered activities in the areas listed in Table 6. Within healthcare, three examples are:

<table>
<thead>
<tr>
<th>Table 6: Areas Pursuing Domain Standardization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manufacturing</td>
</tr>
<tr>
<td>Engineering</td>
</tr>
<tr>
<td>Medicine</td>
</tr>
<tr>
<td>Space</td>
</tr>
<tr>
<td>Legal</td>
</tr>
<tr>
<td>Petroleum</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Telecommunications Billing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manufacturing</td>
</tr>
<tr>
<td>Engineering</td>
</tr>
<tr>
<td>Medicine</td>
</tr>
<tr>
<td>Space</td>
</tr>
<tr>
<td>Legal</td>
</tr>
<tr>
<td>Petroleum</td>
</tr>
</tbody>
</table>

Most of these activities are intended primarily to provide standardization for the domain and not necessarily for the associated ISs. There is significant value to establishing a shared domain orientation, independently of the ISs. A domain orientation can provide a framework which is produced by a consortium. RICHE has been adopted by more than 15,000 hospitals in Europe.

The examples in Tables 6 and 7 illustrate opportunities and challenges. The opportunities are obvious. A shared domain orientation assists all members of the domain, within some limits (e.g., errors and limitations). In the worst case, it can lead to a fragmented approach where different members of the domain use different domain orientations to achieve re-use.

One final challenge could be termed legacy migration. Let's assume that we have an adequate domain orientation and are able to define new interoperable classes, components, and, from them, applications. How do you migrate to the brave new world? At a minimum, it will be an iterative, evolutionary transition. This requires...
that the legacy ISs, which are unlikely to conform to the domain orientation, must interoperate with the new ISs that do. We are back to square one, a massive IS environment with one DOC application being added, further contributing to the heterogeneity and application interoperability challenges in hopes of ultimately reducing these problems.

| Common Basic Specification (GB) RICHE (Europe) |
| READ3 HELIOS II |
| NUCLEUS CANON |
| General Architecture for Languages, Encyclopedias and Nomenclatures GALEN-IN-USE CEN TC251 |
| GAMES DILEMMA |
| PRESTIGE: SYNAPSES SNOMED |
| The Good European Health Record Framework for European Services in Telemedicine Strategic Health Informatics Networks for Europe |
| Computer Based Medical Records Institute |
| Patient-Oriented Management Architecture (USA) |

Table 7: Healthcare Domain Standardization Activities

Whoever solves the problem of domain orientation, or even application interoperability within a domain, deserves a Noble Prize, perhaps the Nobel Peace Prize, for it will certainly not be a technical achievement, but something far more valuable.

In conclusion, application interoperability is a fundamental requirement for end users of DOC technology. It is not a technical problem. However, DOC technology should be developed to facilitate the definition of the domain models, the interface specifications, and the supporting frameworks. The DOC community should understand the nature and full scope of this challenge, work directly with the domains that they should serve, and focus effort accordingly on the relevant domain models (i.e., interoperable domain model families) and supporting frameworks. No small task!

Figure 13: Object Model Family

Figure 14: Domain Model Interoperability

7 Conclusions

We are at the beginning of a 20-year paradigm shift to distributed object computing. By that time, some variant of DOC will be the dominant computing paradigm and will be effectively and readily deployable. Long before that time, it will have met many of its current claims. Indeed, there are already major successes with large-scale industrial-strength DOC applications.

For the moment, DOC is in its infancy and does not meet industrial-strength requirements or the claims of its proponents. DOC is not yet ready for prime time. There are even very recent claims that a major breakthrough has occurred and that a DOC renaissance is upon us [MSFT]. Based on our experience, GTE has decided to halt the design, development, and deployment of DOC technology and applications. In part this relates to our recognition of the problems described in this chapter. In part, it also relates to our pursuit of commercial off the shelf (COTS) applications for which the vendors are largely responsible for the issues raised in this chapter. Following a significant study of and investment in DOC technologies and methodologies, we have concluded that the benefits do not currently warrant the costs to overcome the challenges described in this chapter. The claims for increased productivity, re-use, and lowered costs cannot be achieved with other than very highly skilled staff who must work with immature technology and methods. We will continue to investigate the area and observe its progress and will be prepared to take full advantage of the technology when DOC is more mature. I look forward to a highly competitive market for the DOC infrastructure and highly competitive products. However, I hope that end users such as GTE will be increasingly remote from the technology issues discussed in this chapter so that they can better focus on their businesses and the business requirements and leave as many technology issues to the experts, the vendors, and the COTS suppliers.

Regardless of when DOC technology is deployed, we continue to face on a daily basis the ultimate end-user challenge of application interoperability. Although this challenge is essentially not technical, DOC has the potential to succeed based on its ability to support domain orientation, as described above. The community developing DOC technology should consider establishing application interoperability as its primary goal and defining a comprehensive distributed object framework such as outlined above. DOC technology development should be driven by the requirements of industrial-strength applications and specifically to support the requirements domain orientation. Although the Microsoft announcement [MSFT] is encouraging from a technical point of view, it does not begin to address the application interoperability challenge, the ultimate end-user requirement.

References


[DEC] Descartes, R., Essais Philosophiques (Philosophical Essays), Discours de la méthode (Discourse on Method) (1637).


Kant, I., Critique of Pure Reason (1781).

Locke, J., Essay Concerning Human Understanding (1690).


