1999-10

Interoperability technology assessment for Joint C41SR systems

Berzins, Valdis; Luqi; Shultes, Bruce C.; Guo, Jiang; Allen, Jim; Cheng, Ngom; Gee, Karen; Nguyen, Tom; Stierna, Eric

Monterey, California. Naval Postgraduate School

https://hdl.handle.net/10945/25581

NPS-CS-00-001

NAVAL POSTGRADUATE SCHOOL
Monterey, California

Interoperability Technology Assessment for Joint C4ISR Systems

By

Valdis Berzins, Luqi, Bruce C. Shultes, Jiang Guo, Jim Allen, Ngom Cheng, Karen Gee, Tom Nguyen, Eric Stierna

October 1999

Prepared for: **Naval Postgraduate School**
Monterey, CA 93943-5000

NAVAL POSTGRADUATE SCHOOL
Monterey, California 93943-5000

This report was prepared for Naval Postgraduate School
and funded by Joint Battle Center & Institute for Joint Warfare Analysis

This report was prepared by:

Valdis Berzins

V. Berzins
Professor, Computer Science

Reviewed by:

Dan C Boger

Dan Boger
Chairman, Computer Science

Released by:

[illegible]

D. W. Netzer
Associate Provost and
Dean of Research

REPORT DOCUMENTATION PAGE	Form approved OMB No 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE October 1999	3. REPORT TYPE AND DATES COVERED Final Progress Report, 6 - 9/1999

4. TITLE AND SUBTITLE Interoperability Technology Assessment for Joint C4ISR Systems	5. FUNDING NUMBERS JW-00-004
6. AUTHOR(S) Valdis Berzins, Luqi, Bruce C. Shultes, Jiang Guo, Jim Allen, Ngom Cheng, Karen Gee, Tom Nguyen, Eric Stierna	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Software Engineering Group, Department of Computer Science, Naval Postgraduate School, Monterey, CA 93943	8. PERFORMING ORGANIZATION REPORT NUMBER NPS-CS-00-001
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Joint Battle Center & Institute for Joint Warfare Analysis	10. SPONSORING/MONITORING AGENCY REPORT NUMBER
12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.	12b. DISTRIBUTION CODE A

This study characterizes and assesses alternative approaches to software component interoperability in distributed environments typical of C4ISR systems. Interoperability is the ability of systems to provide services to and accept services from other systems, and to use the services so exchanged to enable them to operate effectively together. This study characterizes and assesses alternative approaches to software component interoperability in distributed environments. Candidate approaches include wrappers, translators, data mediators, replicators, messaging, Object Request Broker (ORBs), and JINI for legacy systems and new systems.

14. SUBJECT TERMS Interoperability, CORBA, COM/DCOM, Messaging and Wrappers, Data mediators, Translators and Replicators	15. NUMBER OF PAGES 80 16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL

Interoperability Technology Assessment for Joint C4ISR Systems

Valdis Berzins, Luqi, Bruce C. Shultes, Jiang Guo, Jim Allen,
Ngom Cheng, Karen Gee, Tom Nguyen, Eric Stierna

Computer Science Department
Naval Postgraduate School
Monterey, CA 93943

ABSTRACT

This study characterizes and assesses alternative approaches to software component interoperability in distributed environments typical of C4ISR systems.

Interoperability is the ability of systems to provide services to and accept services from other systems, and to use the services so exchanged to enable them to operate effectively together.

This study characterizes and assesses alternative approaches to software component interoperability in distributed environments. Candidate approaches include wrappers, translators, data mediators, replicators, messaging, Object Request Broker (ORBs), and JINI for legacy systems and new systems.

The "alternatives" do not uniformly address the same range of issues; they fall into several categories:

Category 1: Building blocks for interoperability

- Messaging and wrappers
- Data mediators are a special class of wrappers
- Translators and replicators

Category 2: Architectures for unified, systematic interoperability

- CORBA
- COM/DCOM
- Custom architectures

Category 3: Packaging approaches for encapsulating interoperability services

- Commercial or distributed ORB
- JINI
- Custom ORB in JAVA, Ada95, C/C++, etc.

Definitions of the alternative approaches can be found in this report. In all cases, the assessment was performed examining the candidate approaches from th perspective of a software engineer trying to utilize a given approach for C4ISR system interoperability.

Outline

1. Summary

This study characterizes and assesses alternative approaches to software component interoperability in distributed environments typical of C4ISR systems. The "alternatives" do not uniformly address the same range of issues; they fall into several categories:

Category 1 : Building blocks for interoperability
- Messaging and wrappers
- Data mediators are a special class of wrappers
- Translators and replicators

Category **2:** Architectures for unified, systematic interoperability
- CORBA
- COM/DCOM
- Custom architectures

Category **3** : Packaging approaches for encapsulating interoperability services
- Commercial or distributed ORB
- **JINI**
- Custom ORB in JAVA, Ada95, C/C++, etc.

Definitions of the alternative approaches can be found in Section 4. In all cases, the assessment was performed examining the candidate approaches **from** the perspective of a software engineer trying to utilize a given approach for C4ISR system interoperability.

The assessment is summarized using Kiviat graphs, which are also known as spider diagrams. The zero point is at the center of the circle. The "interpretation" along each axis in the graph varies slightly depending on the approach being assessed. This stems from the differing roles that the approaches play in software interoperability efforts, as indicated by the categories identified above.

Scale for Kiviat Graphs

0 Not Applicable
1 definitely poor
3 sometimes good, sometimes poor
5 definitely good

Wrappers

Performance **(3)** depends on the level of abstraction where a wrapper is inserted to an existing system. Performance can be good when high levels of abstraction are possible, but poor at fine-grained levels.

Reliability **(3)** wrappers do not intrinsically provide any reliability improvement. However, they can be used to implement a reliability schema.

Speed to field (2) wrappers require custom code for each interaction; technology to generate wrappers automatically is not yet in place.

Cost (2) same as above.

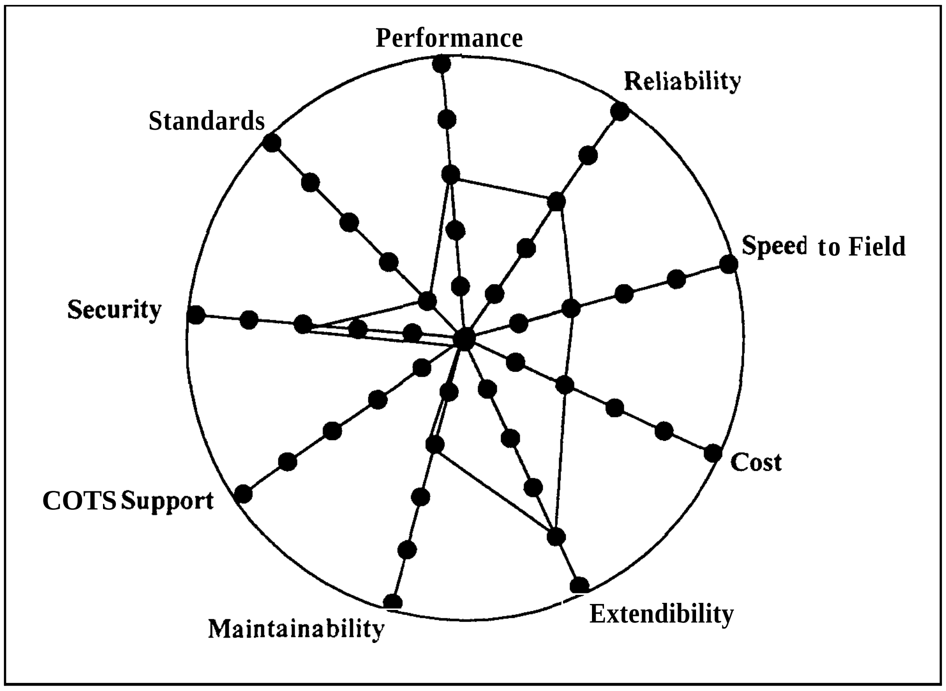
Extendibility (4) Extension is relatively easy if the wrappers are designed properly.

Maintainability (2) The legacy code inside the wrapper is hard to maintain, but wrapper enable incremental engineering that can be easier than full scale reengineering.

COTS support (0) **N/A** manual creation of wrappers does not need tool support, automatic generation is beyond current state of practice.

Security **(3)** Wrappers do not intrinsically provide security, but they can be used to implement a security framework.

Standards (1) We found no relevant standards.



Wrapper Kiviat Graph

Data Mediators

Performance **(3)** can be bad if a large amount of data has to be processed. Can be good if significant data aggregation takes place, reducing size of data.

Reliability (4) High assurance mediators are supported in some vendor applications.

Speed to field (2) Mediators are supported by COTS products but generally will require substantial customization to enable interoperability of military systems.

Cost (2) Same as above

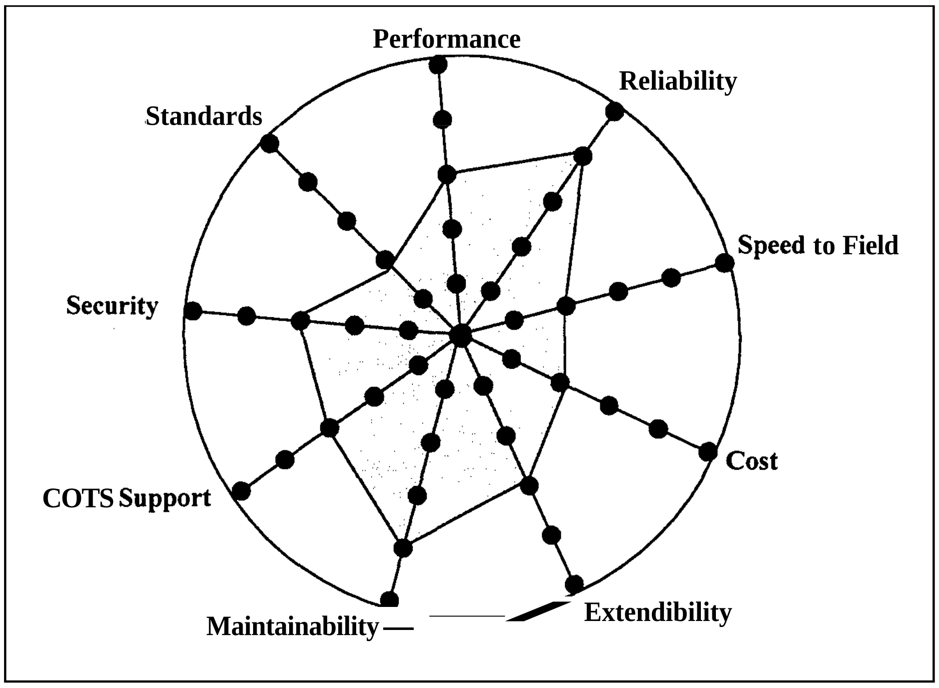
Extendibility **(3)** New capabilities have moderate cost to add if they are close to legacy services. Extensions can be difficult if complex new rules are needed.

Maintainability (4) Improves maintainability by decoupling client from the server.

COTS support **(3)** Many products are available but maturity is weak because this is a relatively new technology.

Security **(3)** No special services are provided by current products but the technology could be used to implement security rules.

Standards (2) Existing standards address only parts of the relevant issues.



Data Mediator Kiviat Graph

Data Replicators

Performance **(3)** Replicators can improve performance if queries are more frequent than updates and the number of copies **is** small. Performance can degrade in the other extreme. Space usage is increased.

Reliability ***(5)*** Redundant data improves fault tolerance.

Speed to field (4) Data replication can enable distribution of data without redesign of database schemas, particularly if combined with data mediators.

Cost (4) Software cost is relatively low, but may need extra hardware and disk space.

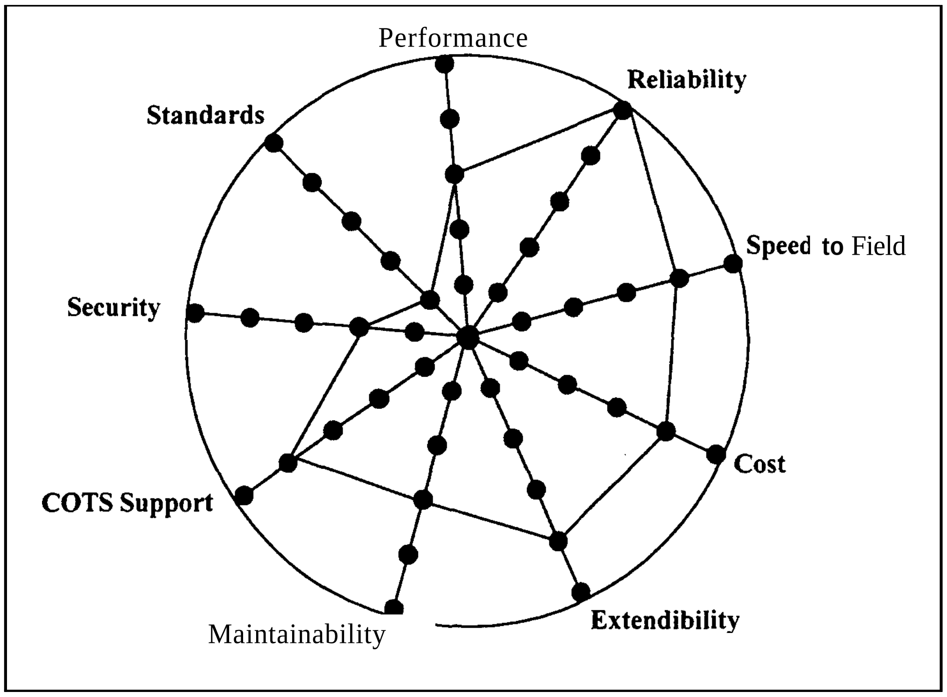
Extendibility (4) Coverage of multiple platforms, languages and protocols is supported.

Maintainability **(3)** depends on tighmess of integration between databases and applications.

COTS support (4) This is a reasonably mature approach.

Security (2) Replicators do not enhance system security.

Standards (1) We found no relevant standards.



Data Replicator Kiviat Graph

Data Translators

Performance (3) - Data translators' impact on system performance depends on usage. One time usage implies a preprocessing performance overhead and can yield good performance in operational use. Repeated use implies a steady performance overhead and can yield poor performance. Performance also depends heavily on the magnitude of the data source that must be translated.

Reliability (3) – Data translators tend to be domain specific. In domains where standards for data interchange have been developed, data translations tend to yield good reliability. With standards for data format and meaning, automated data translation is possible and can be highly reliable if tools are mature. Without standards, manual translation or validation is needed and data errors are easily introduced.

Speed to Field **(4)**– COTS software frequently supports data translation between proprietary data formats and competitors data formats as a way of supporting customer needs and gaining market share. Speed to field can be very fast if the needed translators are available off the shelf. GOTS software tends to require custom translators due to the "smaller" audience that uses the software. This can be moderately fast if best translator generation technology is used, although it depends on the complexity of the data schema.

Cost (3) – Translators for **COTS** software are either readily available or one has to be requested from the vendor. **GOTS** software usually requires a custom built translator. The cost is likely greater for legacy systems than for newer software.

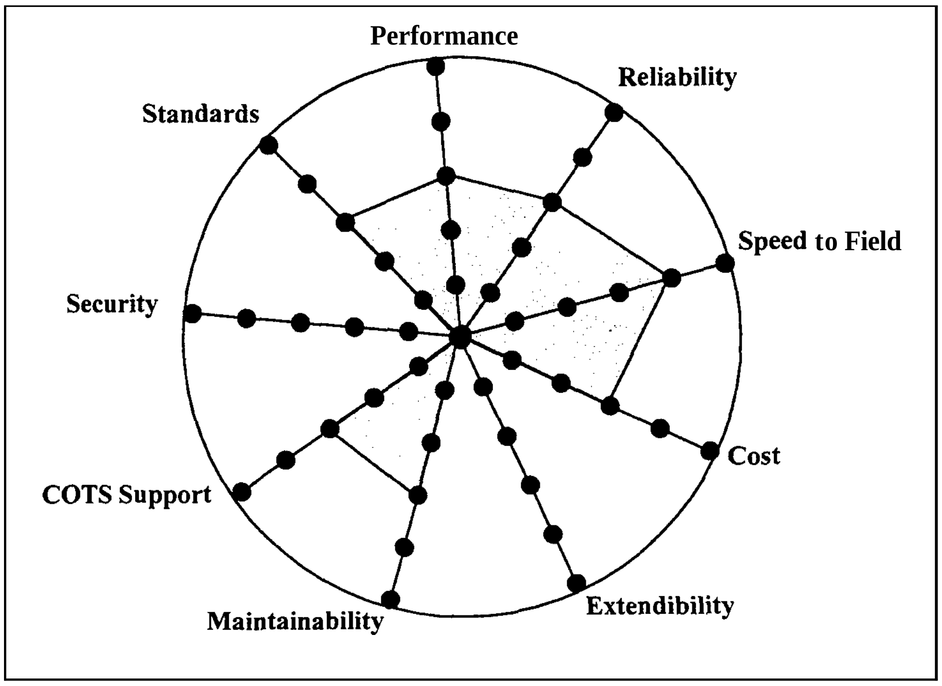
Extendibility (0) – N/A

Maintainability **(4)** – Typical usage of data translators would not require any maintenance of the translator. It is possible to use a translator repeatedly to provide periodic "snap-shots" of a data source, but this creates a time lag in data visibility that could be considered a data maintenance problem.

COTS Support **(3)** – Availability of data translators varies greatly among domain specific software classifications. COTS support for GOTS software is minimal at best. There do exist **COTS** tools for generating translators, and there are many promising research results that have not yet been commercialized.

Security (0) – N/A

Standards **(3)** – Data format standards can make data translators very productive. Lack of standards or violation of standards can render data translators useless.



Data Translator Kiviat Graph

Messaging

Performance **(3)** – The performance of messaging varies greatly based on the choice **of** the type **of** messaging, the messaging standard selected, the synchronicity required, and the hardware/network configuration servicing the message.

Reliability (4) – Any level of reliability desired is feasible, but reliability comes with associated costs and possible performance penalties.

Speed to field (4) – There is strong potential to rapidly field new or alternative messaging technologies. The limiting factors are legacy systems that require synchronous messaging to interoperate.

Cost (4) – The cost of fielding new messaging technologies can be considered to be nominal. The benefits **of** upgrading messaging capabilities usually outweigh the actual costs. Initial costs vary depending on the context.

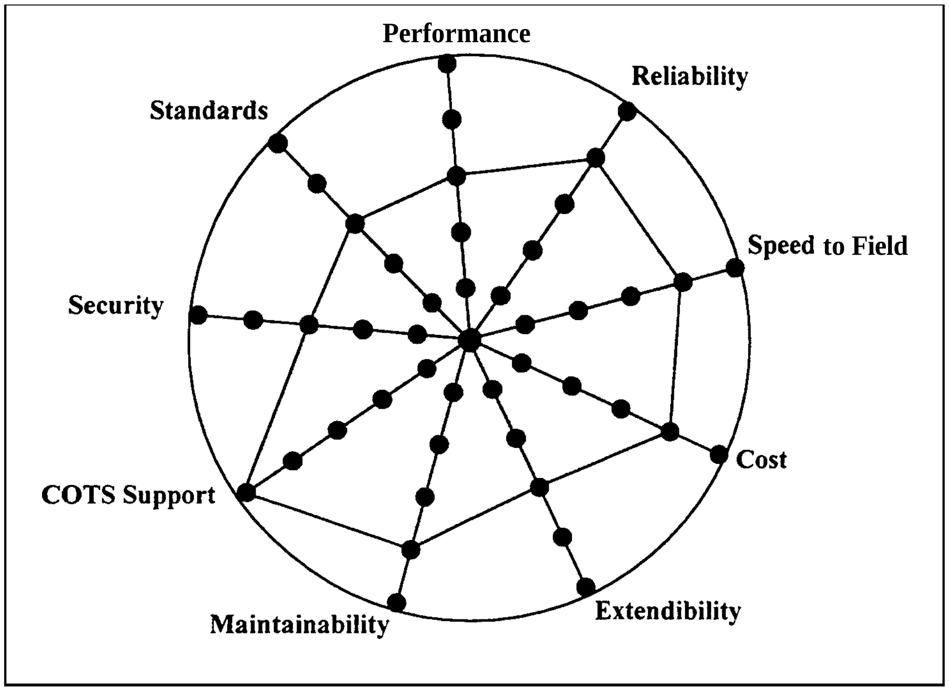
Extendibility **(3)** – If systems utilize several levels of abstraction in defining messaging, then there is significant potential for extendibility. When non object oriented approaches are used to define messaging then extendibility is much more complex and costly.

Maintainability (4) – Similar to the cost analysis, the long-term benefits in maintainability tend to outweigh the cost of switching to a messaging solution that exploits abstraction and encapsulation.

COTS Support *(5)* – Many products are available, see section 4.

Security (3) – No evidence of support for intruder detection and prevention.

Standards **(3)** – Emerging messaging standards such as XML are promising, but it is too early to tell whether XML will live up to its potential.



Messaging Kiviat Graph

ORBs

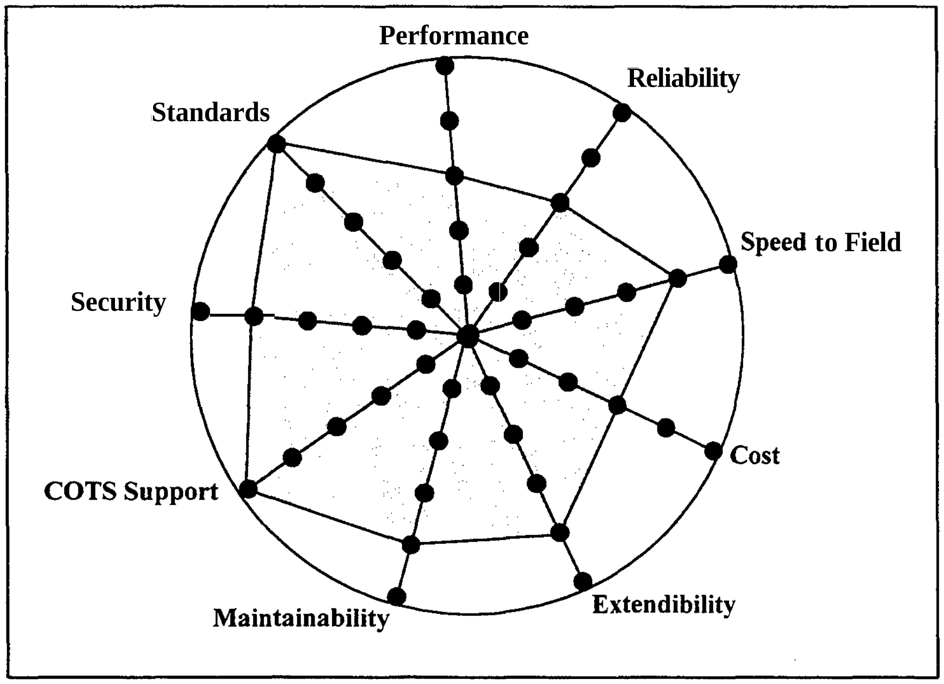
Three alternative approaches to **ORB** use were considered. Each alternative assumed exclusive utilization of one architecture for building interoperable sub-systems. Hybrid approaches are possible. In particular, one ORB could support CORBA, COMDCOM, and a custom architecture simultaneously. In this case, some elements from each alternative evaluated here would apply. The ability to utilize several CORBA ORBs simultaneously is showcased on a webpage (http://www.corba.net) maintained by the OMG consortium. The showcase collects success stories for ORB interoperability. In general, utilizing one ORB that includes all of the functionality an organization requires is preferable.

The following evaluation explains the weights for ORBs. For each evaluation factor, the first weight is for CORBA, the second weight is for COMDCOM, and the third weight is for a custom ORB.

Performance (3-3-5) CORBA and COMDCOM can perform very well if one only needs static invocation of service capabilities. Unfortunately, they are both very slow when utilizing dynamic service invocation. Performance can be improved if one utilizes a custom architecture in conjunction with best available messaging capabilities.

Reliability (3-4-3) CORBA appears to be reliable, but not all of the reliability evaluation criteria are currently supported in CORBA. COMDCOM appears to be reliable and provides mechanisms to satisfy the evaluation criteria. However, the popularity of CORBA provides strong empirical evidence validating CORBA's reliability while evidence validating COM/DCOM's reliability is harder to find. Anything custom designed for an interoperability problem will be as reliable as the vendor that develops it.

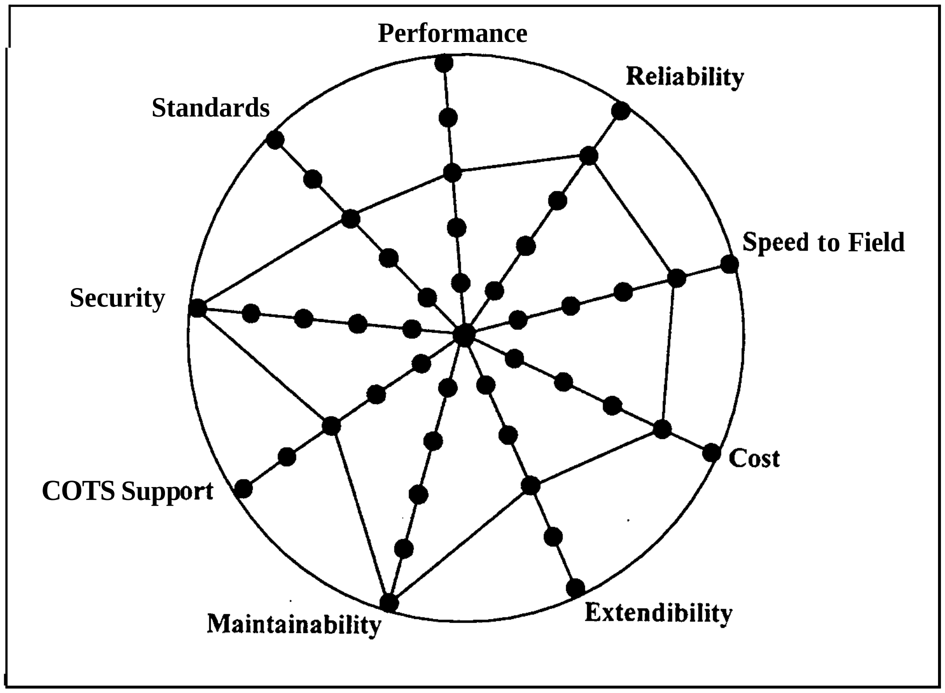
Speed to Field (4-4-1) ORBs utilizing CORBA and COM/DCOM are readily available, support information hiding, and software reuse. CORBA's popularity and Microsoft's backing of COM/DCOM have made these alternatives easier to use and have provided vendor support for their use. Developing distributed applications is still a time consuming process, but these alternatives are making this process easier. A customized architecture requires time to develop the architecture, time to train people to use the architecture, and potentially lengthens the time needed to field an ORB.



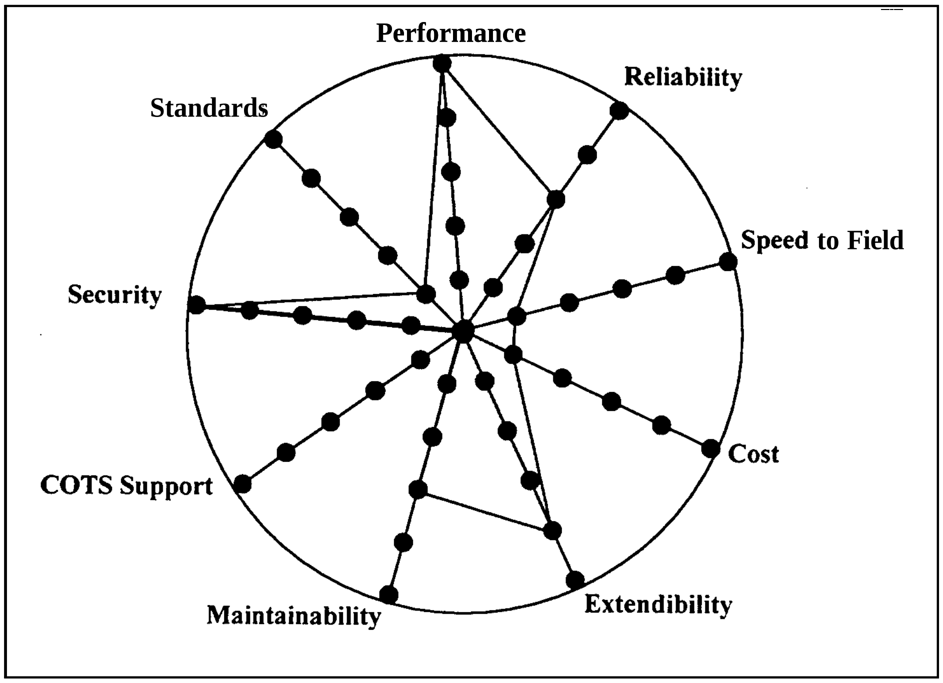
ORB using CORBA Kiviat Graph

Cost (3-4-1) CORBA ORBs vary greatly in cost and functionality. COM/DCOM ORBs are primarily developed for/by Microsoft. Microsoft's backing appears to make these ORBs more affordable for use on Microsoft platforms, but this does not appear to be true for Microsoft competitors. A customized architecture and ORB requires much more time and consequently money to deploy.

Extendibility (4-3-4) CORBA is a flexible architecture that comes a wide variety of platforms but it does not yet support intelligent agents. COM/DCOM provides a fair level of flexibility, but it is primarily available and used in Microsoft computing environments. Extendibility can be designed into a custom ORB, but this is at the discretion of the vendor developing the ORB.



ORB using COM/DCOM Kiviat Graph



Custom ORB Kiviat Graph

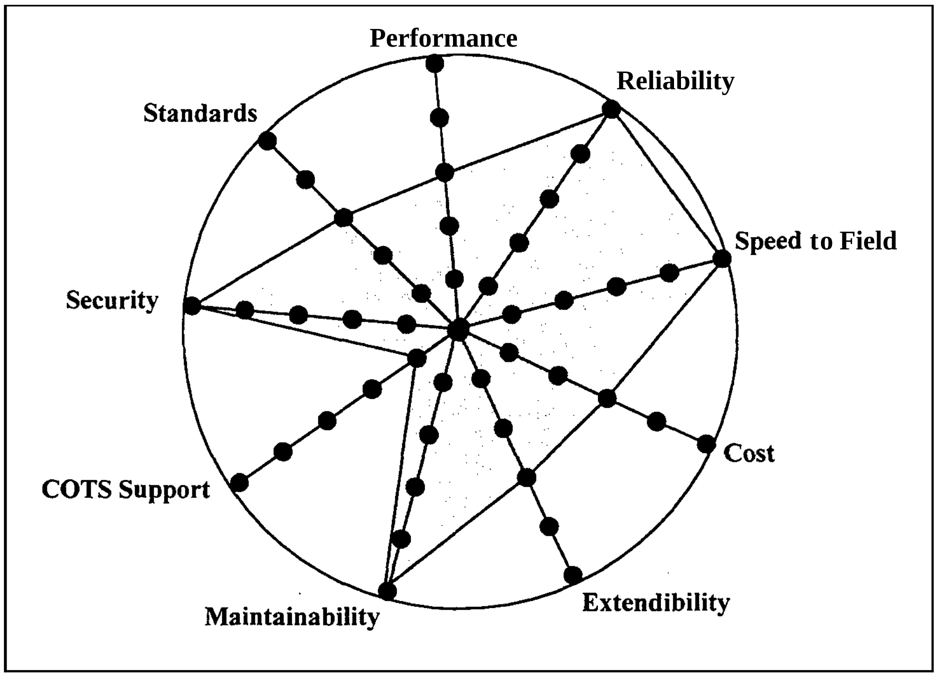
Maintainability **(4-5-3)** Applications that use COMDCOM are easier to maintain than CORBA applications because COM/DCOM has a simpler interface to desired services. Using one of these standards provides a mechanism for improving maintainability over custom ORBs. A custom ORB makes the customer reliant on the original vendor who provided the ORB for maintenance.

COTS Support (5-3-0) CORBA appears to be the most widely used architecture for interoperability. CORBA has been adopted by a wide variety of ORB vendors. COMDCOM suffers from the perception that it primarily supports Microsoft platforms. By definition, a custom ORB and interoperability architecture is not commercially supported, but it could support interoperability among an arbitrary set of COTS/GOTS products.

Security **(4-5-5)** CORBA provides a strong core set of security features, but these are not as sophisticated **as** the features in COM/DCOM. Arbitrary security features could be included in a custom ORB.

Standards (5-3-0) CORBA is a standard that provides enough flexibility so that it can be used in most situations. COM/DCOM is more rigid than CORBA in that it was originally designed for and predominantly supports Microsoft platforms. A customized ORB is not necessarily required to follow any standards at all.

JINI



JINI Kiviat Graph

No empirical data is available to validate Sun Microsystems' claims about projected characteristics of JINI. This assessment reflects this uncertainty.

Performance **(3)** not yet known.

Reliability (5) Sun anticipates good reliability based on past successes with Java based technologies.

Speed to Field (5) One of the motivating factors for JINI is rapid system interoperability. It is not clear if this goal will be achieved.

Cost **(3)** JINI is not get well disseminated into the commercial and academic sectors. In addition, JINI is a Java specific solution.

Extendibility **(3)** JINI is a Java specific solution. It is unclear whether an adequate linkage between non-Java and Java applications will develop or be practical.

Maintainability (5) projected to be good.

COTS Support (1) JINI is too new to have any COTS support yet.

Security ***(5)*** based on Java support for security.

Standards **(3)** related to all-Java issue.

Recommendations

All of the building blocks are relevant to interoperability, and the detailed assessments contained in the body of this report provide some guidance about the circumstances under which each is applicable. We recommend that these building blocks be used in the context of an ORB in the cases where available standards, tools and trained personnel are sufficient to practically do so. This recommendation is strongly time-sensitive, because of current rapid development in these areas. Detailed results and an additional potentially serious risk factor are described in the conclusions of this report (Section 5).

Overview of the Report

Section 2 provides background information about this study and explains the evaluation criteria used. Section **3** gives the details **of** the assessments. Section 4 defines the alternatives being assessed and describes the technologies in more detail. Section ***5*** gives conclusions. Appendices A and B list some of the vendors for the technologies. Appendix C contains experience reports from a sample of past DoD applications of the technologies. Appendix D gives a skeletal domain analysis for C4ISR interoperability that we used to provide a context for our assessments.

2. Introduction

The Joint C4ISR Battle Center (JBC), Suffolk, Virginia has been tasked by Joint Staff, J6, to report to the Joint Requirements Oversight Council (JROC) on the assessment of a particular approach achieving interoperability. The JBC needs a short term analysis of the various alternative approaches one could take to achieve interoperability among independently developed legacy systems, as well as, approaches to take during the design of new systems that would enhance interoperability. This report contains the results of a paper study conducted by the Software Engineering Center at the Naval Postgraduate School to provide such an analysis.

Interoperability is the ability of systems to provide services to and accept services from other systems, and to use the services so exchanged to enable them to operate effectively together. Many COTS/GOTS systems and products are not developed **for** Joint Operations. Each of these systems usually performs the specific task that drove its design. This effort will help identify useful approaches for developing new systems, which will interoperate, as well **as,** provide remedies for existing non-interoperable C4ISR systems.

This study characterizes and assesses alternative approaches to software component interoperability in distributed environments. Candidate approaches include wrappers, translators, data mediators, replicators, messaging, Object Request Broker **(ORBs)**, and JINI for legacy systems and new systems.

Evaluation Criteria

1. Performance
    - 1.1 Time
        - 1.1.1 CPU
        - 1.1.2 latency
    - 1.2 Space
    - 1.3 Bandwidth
    - 1.4 Throughput

2. Reliability
    - 2.1 High assurance
    - 2.2 Fault detection and recovery
    - 2.3 Redundancy
    - 2.4 Fault Isolation
    - 2.5 Mean Time between Failures

3. Speed to field
    - 3.1 Legacy systems
    - 3.2 New systems

4. cost
    - 4.1 Integration
        - 4.1.1 Retrofit vs. Design-in
        - 4.1.2 Ease of Integration
            - 4.1.2.1 Development
            - 4.1.2.2 Plug-in
    - 4.2 Maintenance
    - 4.3 Life Cycle Costs

5. Extendibility
    - 5.1 Coverage/Scope
        - 5.1.1 Hardware
        - 5.1.2 Operating systems
        - 5.1.3 Languages
        - 5.1.4 Synchronous/asynchronous protocols
    - 5.2 Support for intelligent agents
    - 5.3 Support "hooks" for plug-ins

6. Maintainability
    - 6.1 Cost of change
    - 6.2 Time to change
    - 6.3 Impact of change on reliability
    - 6.4 Difficulty of Change

7. **COTS** support
    - 7.1 Level of maturity
    - 7.2 Names of vendors of competing COTS products

8. Security
    - 8.1 Ability to detect and prevent unauthorized entrance to systems
    - 8.2 Ability to detect and prevent interception **of** critical data in transit
    - 8.3 Immunity to disruption of network services
    - 8.4 Ability to detect and prevent corruption of data in repositories **or in** transit
    - 8.5 Ease of use

9. **Standards**

9.1 Role of the Standard
9.2 Impact of not complying with the standard.

3. Evaluation and Assessment

Wrappers

Definition

Software wrapping is a technique in which an interface is created around an existing piece of software, providing a new view of the software to external systems, objects, or users. Wrapping can be accomplished at multiple levels: around data, individual modules, subsystems, or entire systems.

The narrow concept of a wrapped object is an object with its methods surrounding the legacy software represent its encapsulation as a single object. Object-oriented technology (OOT) may be counted among the best practices for software development by virtue of its efficiencies in development and maintenance and its inherent support for reuse. OOT consists of a set of methodologies and tools for developing and maintaining software systems using software objects composed of encapsulated data and operations as the central paradigm. The legacy software is accessible only through the object-defined methods (or operations). Any user access to the legacy software would be mediated through some of these methods, whether the user interface is a complex set of objects constituting a graphical user interface **(GUI)** or simple terminal line command input/output (I/O).

The broader conception of an **OO** wrapper is an object model consisting of multiple classes and objects. This object model is created **as** part of the wrapper to provide a natural **OO** interface to the principal conceptual entities implicit in the original system. The new objects and classes of such a wrapper can interface with the legacy programs and data in different ways. An application programming interface (API) may mediate communication between the wrapper object model and the legacy program. When the legacy software is a database, a database server might provide the functionality of an API, with objects accessing the database through SQL calls to the server.

Level of Abstraction	Wrapping Overhead Comments
Function	Overhead is high. Not recommended unless the function is unusually complex & difficult to re-code or the extra call can be in-lined by the compiler.
Procedure	Overhead is high. Recommended but should initiate a performance evaluation before implementation.
Sub-Module	Low overhead. Recommended.
Module	Overhead is moderate. Should consider the execution environment.
Complete System	Recommended. Should look at client-server model during implementation.
Data File	Highly recommended. Relatively low overhead. Care should be taken during data modeling in Ada.
Database	Highly recommended. Select SQL interface or other form of binding to database.

When wrapping a database management system, one potential drawback in separating data tables into distinct objects is that query response time may be adversely affected when table joins are involved since a join must be reformulated as multiple queries in order to access multiple tables via their separate encapsulating methods. A simpler alternative that might minimize this problem is encapsulating the entire database as an object. This approach may retain the same order of efficiency as queries in the legacy database because the same database queries could ultimately be invoked by the database object's methods. Complex queries could be posed directly to the wrapped

DBMS object and processed as joins. The only extra query costs are the small constant-time overhead incurred passing through the methods. The data are all encapsulated, so that the access methods can be independent of the data storage organization.

Wrapping a whole database system like this can be an efficient means of establishing compatibility with new interoperable data standards while providing a framework for transparent incremental modernization of the internal data representation. However, this approach might have to compromise some of the modularity and encapsulation implicit in an object model of the database since access to stored objects could not all be mediated by their distinct classes if complex queries are to be handled directly. Access to objects cannot always be mediated by the operations of their individual classes if complex queries about different classes of objects can be sent directly to such a DBMS object wrapper.

Reliability

For the wrapper, the whole system reliability mostly depends on the legacy systems since the wrapper only provides an interface layer between the legacy systems. Usually, this kind of system has high assurance. **A** basic wrapper will not provide extra fault detection and recovery ability beyond what the legacy systems have. It also does not have extra redundancy. More sophisticated wrappers to provide such capabilities can be created, but at appreciable cost.

Speed to field

Wrapping can establish compatibility of old code with new interoperable data description standards by supporting the translation between these standards and legacy formats at the methods interface to the wrapped software. Wrapping facilitates the rapid transition from legacy to migration systems by minimizing the amount of code rewriting and database restructuring required in the initial stages of migration. Thus, a partially modernized migration system may be fielded sooner than it could if the entire legacy system were reengineered at once. Maintenance and modification costs can be more quickly reduced since there are fewer separate systems to maintain when multiple redundant legacy systems are transitioned to fewer standardized systems.

In the course of investigating alternative strategies and tactics for OO wrapping, a number of guidelines have emerged for choosing and applying them in a variety of contexts. OO wrapping has been identified as an effective technique for encapsulating legacy software components within a partially modemized migration system. Wrapping can support staged migration of legacy systems to modernized OO systems as well as the incorporation of trusted legacy software into new systems. Another application for which wrapping is recommended is to establish data standardization of legacy code and data without reengineering legacy systems.

When the resources are available, it is recommended to use domain object models for wrapping legacy components rather than simply wrapping components as software objects. Such object model wrapping is identified as providing a better foundation for any subsequent legacy modernization or extensions. Costs of building such object models can be minimized by judicious abstraction of the domain objects, modeling only those features that are essential to wrapping.

One reason behind favoring wrapping over reengineering is the presence of any strong time pressure to modernize a legacy system quickly; factors include the following:

- Expiring hardware and software contracts
- Shift to new platforms
- New functionality requirements
- Requirements for interoperability with other reengineered AISs
- Data item standardization requirements

Other reasons to prefer wrapping to reengineering are as follows:

- Absence of documentation
- Departure of all domain experts

Complexity of code

- Fragility (or brittleness) of code
- Size of code or database

- Staffing resource limitations

Wrapping feasibility depends on conditions of the legacy and target migration environments, such as modularity of legacy code, and support **for** interfaces between legacy components and the migration **OO** environment. Under such favorable conditions, wrapping may be the most effective means of meeting modernization deadlines.

cost
Wrapping is most often discussed as a temporary measure-ordinarily as an interim solution to the problems **of** modernizing legacy software, a stepping stone on the path of a full system reengineering. In some cases it may also serve as a terminal treatment for obsolescent software, when interoperability is required temporarily for a system that is close to retirement. In either case, it can provide some of the benefits of **OO** systems without the costs of fully reengineering all the legacy code. The wrapped sections of a legacy system can participate as objects in a broader **OO** system, while the details of the legacy code and/or data are encapsulated. When data are so encapsulated, the system may be accessed via standard modernized data and object definitions without disrupting the legacy database. Wrapped objects may even be reused in other systems if the granularity wrapping creates objects of potential use elsewhere.

A drawback specific to the direct wrapping technique is the scale of large legacy components when wrapped as objects. If a legacy software component **of** a large size were to be fully reengineered using OOT, it would most likely break down into multiple interacting objects to better reflect its implicit organization of information and procedures. Direct wrapping can save some of the work of this decomposition and reorganization in the short term when wrapping at a coarse level using large software components. But such savings come at the cost of a coarser scale representation that may be more awkward to integrate with the rest of an **OO** system. Where other objects might only need to access some small part of the data or functionality of the wrapped object, they must refer to the wrapped component as a whole. Where other systems may only need minor functions from a wrapped component, they will have to incorporate the whole component if they are to benefit from its reuse. Furthermore, subsequent reengineering of a directly wrapped legacy program may require substantial rework of an existing class hierarchy in order to accommodate the new classes abstracted from a legacy program.

In some cases it may even be advantageous to wrap all, or most, of the procedures from a legacy program, rather than rewrite any of them initially. This could reduce the initial transition costs as compared to full reengineering, while providing a whole set of program or domain objects that conform to new data standards and might be reused elsewhere.

Full modernization of the legacy code could then proceed in small increments, object by object, with minimal adjustments to the object structure only when indicated by deeper analysis.

Extendibility
When legacy systems are tightly coupled with the hardware, operating system, communication system, terminal handlers, etc., of an obsolete legacy environment, it may be best not to wrap their components within a modemized hardware-software environment. In such cases, a client-server strategy should be applied if a suitable interface can be established between the legacy system and the migration system. Under this approach, a legacy system **(or** components thereof) could operate as a stand-alone client-server and interact with other client applications using a messaging system. An alternative approach can be based on identifying components **of** the legacy system that had little to moderately complex system-specific barriers to wrapping and salvaging them for **use** as wrapped components in a modernized **OO** system. In this case, the wrapped legacy functions and the new system will coexist in the same computer system. Naturally, when a legacy system contains substantial system-specific barriers to wrapping, this will result in less salvageable legacy code.

Maintainability
Migration systems containing wrapped components still face some of the maintenance headaches of the legacy systems since the legacy code is likely to be difficult to modify. Object wrappers can ease some of the modification burdens of software maintenance in so far as the modifications can be accomplished through specialization of a wrapper class without modifying the original code. The feasibility of this technique will vary depending on the system. However effective this technique might prove, bugs in the legacy software will ordinarily require direct

modifications, assuming the code is not reengineered. Thus, maintenance requiring modification of legacy code cannot be entirely avoided in systems using wrapping.

The principal drawback to any type of wrapping is in long-term maintenance, since the legacy software remains beneath the wrappers, and is likely to be difficult to maintain or modify.

This drawback can be substantially mitigated if future modifications can be implemented within the OO portion of a migrated system, external to the legacy code. The addition of new methods and operations to a class wrapper, for example, may be accomplished without touching the legacy code. Augmentation of the methods, attributes, or both of a class wrapper might also be achieved independently of the legacy code by creating new subclasses with the additional structure and/or functionality. Maintenance involving fixes to bugs in the legacy code may be more difficult, however, requiring direct modifications of the legacy code itself. These potential problems with maintenance and modification are reasons for considering wrapping as only a temporary solution for modernizing software systems, with full reengineering (or obsolescence) as an eventual goal (or expectation).

Creating object wrappers at the level of functions and procedures offers potential benefits over coarser-grained wrapping at the level of whole programs or systems. Finer-grained objects can be reused without the encumbrance of ancillary code that may be irrelevant in other applications. It can also ease the transition to a fully reengineered migration system to have already decomposed programs into sets of interacting objects. These advantages of fine-grained domain object wrapping accrue at the expense of higher initial development costs compared to some of the coarser levels of direct wrapping and some degradation of performance. Choices of appropriate levels of wrapping are likely to be driven largely by the ease of decomposition of legacy software and the time and cost constraints at any given stage of software migration. With more time and resources, finer-grain wrapping is feasible, while tight time and resource constraints may require coarser-grain wrapping. Thus, the OO technique of software wrapping provides the migration team with considerable flexibility in meeting these constraints.

COTS support

"SQL Ada Module Description Language" is a tool for database wrapping. The primary objective for SAMeDL is the partial creation of Ada DBMS applications where Ada applications are written without any mixed SQL statements, and SAMeDL modules are written to model the SQL queries. SAMeDL defines an abstract interface, a collection of Ada declarations through which an Ada program can access the DBMS. The meaning of a SAMeDL text is given by a translation into an Ada text, an SQL text, or both an Ada and an SQL text along with the relationship between them. A SAMeDL text may contain some data descriptions and it may also rely on previously processed data descriptions. The meaning of SAMeDL text may include Ada type and/or subprogram declarations. The actions of the subprograms nominally include calls to procedures defined in the SQL module language. The meaning of a SAMeDL procedure includes its Ada declaration, an SQL declaration, and the definitions of the input and output parameters of the procedures declared.

Security

The security mostly depends on the legacy systems since the wrapper only provides some interactive interfaces between the legacy systems. Wrappers usually do not provide extra security abilities to detect and prevent unauthorized entrance to systems, to detect and prevent interception of critical data in transit, to detect and prevent corruption of data in repositories or in transit. Security enhancements can be added in the wrappers, but with additional development cost.

Standards

This research has not found standards specifically for wrappers. Wrappers can be used to interface legacy code to standardized architectures. For example, IDL is implemented via tools that automatically generate wrappers to interface to CORBA.

Translators

Performance
Data translators can be used in one of two ways. They can either run standalone to initialize static databases or be integrated with the rest of the system and run multiple times. Data translators that run just once will have no impact on the performance of the system during normal operation. However, in this situation, there are still other performance considerations to address, i.e. the turnaround time. Depending on the volume of data to be translated, the type of data, and the type of translation (i.e. syntactic vs. semantic), a worst-case scenario can take upwards of days or weeks to translate all the data. For example, the Tactical Environmental Support System 3 [TESS(3)] needed to translate system CAD drawings from a proprietary CAD application to AUTOCAD. The actual translation took 1 week to translate all the system drawings over to AUTOCAD.

When the data translator is integrated as part of the system, system performance issues will need to be addressed. Some performance issues specific to data translators include:

- Operational downtime required to perform the translation, i.e. other operations dependent upon the data cannot be performed until the translated data is available.
- Storage space on system may at least double until confidence in translated data is achieved,
- Some data translations can tax the CPU, which can adversely affect the overall performance of the system. For example, data throughput of the system can slow down during translation.

Reliability
The reliability of data translators will vary depending on the problem domain and whether standards exist for the data formats in the problem domain. In some domains, the errors introduced by the data translator take more time to correct than the translation itself. For example, in 1990, the TESS(3) project needed to translate system CAD drawings from a proprietary CAD application to AUTOCAD. On top of the 1 week required to translate the data, it took an additional month for a person to review and clean up the translated drawings. With well documented and accurate standards, data translation is not as time consuming in the CAD domain.

Reliability can also depend on the sophistication of a data translator. Most data translators only perform a syntactic translation. In some situations, this may not be sufficient. A semantic translation may also be required [Higgs].

There is also the issue of semantic mismatch where the same data may not be the same information in the source and receiving system [Renner]. This can lead to problems if the data translator does not have the intelligence to decipher this type of mismatch between data interpretations. When this occurs, a person will need to review the translated data to ensure the accuracy.

Maintainability
When data translators are used once then discarded, maintenance is not a factor. This is normally the case for database translations. However, situations exist where the data translator is required during system operation because the original form of the data is still active. For example, a system with a legacy communications stream requires the data translator as an integral part of the system. In this situation, the data translator would have to be maintained with the rest of the system.

Maintenance of a data translator is no different then maintenance of any other part of the system. If it is a COTS component, the maintenance organization is dependent upon the COTS vendor for subsequent upgrades. If the data translator is written in-house and source code is available, the source code would need to be maintained by the maintenance organization. Maintenance of source code allows the maintainer more latitude in that any problems found with the translator could be addressed while with a COTS component, the maintainer is at the mercy of the vendor.

cost
The cost associated with data translators can be categorized as direct and indirect costs. Direct cost is the cost associated with the purchase of the data translator. The direct costs can be further categorized as:

- Initial buy - Data translator is purchased and used once then discarded.
- Initial buy plus upgrades - Data translator is purchased, used and maintained as part of system.
- Labor to produce - Data translator is developed in-house.

The indirect costs include maintenance of a data translator that is developed in-house as well as costs to set-up, operate, and correct errors.

COTS Support

There are many COTS data translator products on the market. However, each of them is designed to translate data within a specific problem domain. The following is a listing of the available COTS data translators on the commercial market, grouped by the specific problem domain.

CAD:
http://www.cad2000.com/datax.htm Company provides COTS data translation product for specific types of CAD data
http://www.catia.ibm.com/prodinfo/mcx.html IBM's data translator for CAD data
http://www.compunix-usa.com/products/products.htm CAD/CAM data exchange solutions for CATIA, UNIGRAPHICS, ACIS, PARASOLID, STEP, IGES and Rapid Prototyping applications.
http://www.cimsoftek.com/cimpro.htm translates CATIA to Pro/Engr in the Pro environment without requirement for a CATIA license
http://www.microcadam.com/tech_sup/pages/fod603.html a data translator specification for CADAM

Spatial data:
http://www.lanl.gov/projects/ia/stds/ia610110.html Standard for spatial data
http://www.tetrad.com/fme/fme.html spatial data translation for mapping data
http://www.safe.com/software/spacial_information.htm spatial data translation software
http://www.bentley.coni/products/qeoexch Microstation GeoExchange is a configurable spatial data translator designed to support Microstation GeoGraphics. The translator performs complex spatial and attribute processing of files from various GIS-related sources into a Microstation GeoGraphics project.
http://sofiware.geocomm.com/translators/index.html format translators for **GIS** data
http://www.navsys.com/Digitran.htm data translator for GPS data

Miscellaneous:
http://www.xs4all.nl/~edmundv/index.html#DTPicView freeware and shareware for data picture translators for the palm pilot
http://www.axcis.com/track/info/convert2.htm freeware data translator software for Trackmaster data
http://hsb.baylor.edu/ramsower/acis/papers/tsai.htm Translator software usually translates data from internal data formats into a standard EDI format (e.g., **X.12**) and also validates data with the standards.
http://www.digital.com/info/CU6409/CU6409HM.HTM DIGITAL BASIC Translator assists users in migrating established DEC BASIC applications to a distributed environment by accepting DEC BASIC source code as input and generating Visual Basic source code.
http://www.unidata.ucar.edu/packages/dods.../proposals/can-97-mtpe-01-html/node5.html data format translation for the scientific community

Translator generators:
Most of the tools in this category are based on attribute grammars. A survey can be found at http://www.first.gmd.de/cogent/catalog and a description of a particular tool can be found at http://www.research.philips.com/generalinfo/special/elegant/elegant.html

Data Mediators

Performance

Data mediation performance is relative to the type of mediation, the amount of data aggregation, the number and size of databases in the system, the volume **of** the data returned to the requesting process.

- CPU performance decreases as the rule complexity increases and as the amount **of** data aggregation increases.

- Latency tends to increase as rule complexity increases. Latency decreases up to a limit then increases depending on the amount of data aggregation based on space, and bandwidth savings.

- Space requirements with the client may actually be decreased when using information fusion to reduce the data volume.

- Bandwidth requirements are also decreased when using information fusion to reduce the data volume.

- Throughput is improved as rule complexity decreases and as information fusion aggregates data and reduces the volume of information sent to the requesting process.

Reliability
High assurance mediation is supported in certain vendor applications. The Constellar Corporation advertises a management tool known **as** a "Transformation Hub" that uses the Oracle7 server to manage the integrity and the quality of the information passing to and from their mediation program. No other mention of high assurance support was mentioned in the literature. The implementation of the survivability mediator discussed in [SGSh98] was not capable **of** using rule based reasoning to ensure that the state of the database is not changed by corrupted data but used knowledge of the state of the system for fault detection and recovery. Recovery can be accomplished with a survivability mediator with access to stored versions of the protected databases and a log of the appropriate edits. [SGSh98] Redundancy, Fault Isolation and Mean Time Between Failure was not discussed in the literature with the exception of the hardware used to support a mediation server device.

Speed to Field
Fielding of any mediation technology requires experts to evaluate the legacy or new systems to establish the rule based reasoning, meta-data and the data conversion methods required for implementation. However, once the rules are written and the system is fielded the process of adjusting the system to meet the users changing needs is very responsive and rapid. The technology exists to incorporate legacy systems into an existing mediated system, develop a new system comprised of legacy and new databases, or field a completely new **EAI.** Specific times to field are relative to the needs of the organization and the complexity of the existing and new systems. However this is a definite growth market and there are a multitude of vendors that can provide the required COTS products and expertise to field a system as rapidly as the requirements can be specified.

cost
The cost of implementing a mediator solution varies depending on the stated objectives for the system. No price quotes for hardware or services were obtained in the literature search but cost savings were identified. Cost savings can be realized through the use of information fusion, which reduces costly communication delays and removes the development cost of tailoring client applications to perform the required analysis. Additional cost savings can be achieved through retrofit of a stable legacy system rather than investment in a new database which may require development cost outside the mediator fielding. The architecture of a data mediator is that **of** a plug in technology that creates a three-tier architecture or a hub and spoke architecture. This gives the architecture to achieve additional cost savings through scalability, and reduces the long term maintenance cost as changes to the databases or user applications require only minimal adjustment of the interface with the data mediator. This drives down the life cycle cost of the technology over time. [Wied98].

Extendibility
A number of applications have been developed using mediator technology. A list of companies providing aspects of mediation technology are listed in Appendix A. Early applications of mediation were in military intelligence, but subsequent applications have focused on manufacturing, business, and information management. Other implementations are being developed for in healthcare management and plant safety and environmental cleanup.[Daw198a]

The implementation of mediators can differ significantly. Workstations are most common and often use UNIX. Many current mediators are coded in C and C++. When knowledge-based processing is crucial, mediators have been programmed in languages **as** LISP or with CLIPS, a C-compatible rule language [Malu97]. When optimization is crucial to processing, the mediators may use packages written in FORTRAN. Implementation should be primarily based on maintenance considerations because they are crucial to keeping down long-term costs. [Wied98].

Some companies are focusing on providing the framework for mediators. "For instance, IBrain's core technology is a single framework for querying and analyzing information of multiple types, from multiple places, using multiple analytical methodologies. The technology allows integration of heterogeneous information and analyses as text search and information retrieval for unstructured text data, data analysis and OLAP type analyses for structured database data, collaborative filtering for qualitative data, and prediction and mining type techniques for quantitative data. The domain focus has been on finance, but the expectation is that their technology will be applicable also in industries as healthcare, pharmaceuticals, manufacturing, and enterprise management." [Wied98].

Hooks for plug-ins were only discussed in the literature in the context of expanding the capabilities of the mediator through an object-oriented design. [Herm97]

Maintainability

Mediators have a high degree of maintainability due to the decoupling they achieve between client and server. The interface between the mediator and the user and base layers requires maintenance by knowledge experts to ensure the rules are relevant to the required tasks. But, the costs are far less than those for maintaining tightly coupled legacy client server systems that must be "hand crafted" to adjust to change. [Cons98] Time to change once a mediator is in place is significantly shorter. "Much anecdotal evidence also exists to show that some of the more complex aggregation processes(using mediators) can take as much as 50 hours of computer time to complete." [Cons98] This would be prohibitive if hand coding an interface were required. The impact of change on reliability is related to the quality of the experts that design the mediation system rule set. This is a critical task that requires regular attention to ensure the mediator is meeting the policies it is enforcing.

COTS Support

Level of maturity – data mediation is a new technology that is integrated frequently within larger mature enterprise applications. It relies on many new technologies that range'from knowledge based processing for data aggregation to code generation to extract information for data warehouses.

Names of vendors of competing, COTS products – Appendix A contains a list of the current vendors.

Security

Ability to detect and prevent unauthorized entrance to systems: No mention of this feature was mentioned in the literature but a mediator could be used to control automated access to a system through rule based logic.

Ability to detect and prevent interception of critical data in transit: The mediator is reliant on the network security policy to perform this task.

Immunity to disruption **of** network services: The mediator is reliant on the network security policy to perform this task.

Ability to detect and prevent corruption of data in repositories or in transit: The security mediator was able to detect corruption of data in repositories and recover from the problem successfully in the study of survivability enhancement conducted by Sullivan, Geist and Shaw [SGSh98].

Ease of Use: No ease of use information was found in the literature search.

Standards

The DOD Data Standardization Program reduces the translation requirements for mediation but fails to address the expense of conversion of legacy systems, the need to mediate security, and the ongoing role of mediators in helping collect and display views of information that enable users to make sound decisions based on heterogeneous data. There is a definite need to continue to develop the formal definition languages used to describe data(meta-data) and the interfaces(IDL).

Replicators

Performance

Time, **CPU** - Data replicators add some processing for each update to replicated data. When using replication to decentralize query processing, update transactions usually involve considerably more query processing (most applications are about **90%** query) than update processing. Once the query processing moves to a replica system, the original system is no longer involved. Thus, the source system has a much lighter load on the processor if there are fewer than about 10 replicas of the data. Replication requires capturing changed data, maintaining persistent queues to ensure fault tolerance, optionally enhancing the data, and distributing it to replica sites. Bi-directional replication adds the further need for data collision detection and resolution. The volume of this extra work depends on the replicator. When reviewing software options, obtain an estimate of the added workload from the vendor or test it during product evaluation. Also, local processing offloads work from a central database server so that competition for processor time is decreased.

Time, Latency - In synchronous replication, each transaction updates all copies at the same time, resulting in little additional latency. This is generally accomplished using a technique called two-phase commit. In asynchronous replication, there is always some degree of lag between when the transaction is committed and when the effects of the transaction are available at any replica. The lag is generally measured in seconds. Replication latency times vary depending on the speed of the network, workload, and the source and target systems.

Space - Data replication means, by definition, keeping at least two copies of the same data. If the replicas do not all need to contain all data, a replicator that provides both horizontal and vertical partitioning (selecting just the rows and columns that the replica needs) allows you to minimize storage costs. Systems that act as sources of data updates also require additional disk space. The replicator stores queues of changes at these sites until it can send them to the replicas. The size of these queues varies depending on both the frequency of updates and the number of the replicated copies. For example, if you schedule replication to occur on daily basis, the tables must be large enough to handle the busiest possible day's worth of updates. For continuous replication, which replicates updates as soon as possible, these tables will generally be quite small. However, you must size them to accommodate any updates that may back up in the queue tables due to network and replica outages, or overloaded systems and networks [Lakeviewb].

Bandwidth - As stated above, asynchronous replication improves response time for data requests. Requests are processed on a local server without accessing the wide area network, speeding up the transfer rates for queries. Queries can be performed locally against data replicas, querying traffic from the network can be deferred thus freeing up the bandwidth. Updates require increased network load because communication with all replicas is needed. Net increase or decrease in network load depends on the ratio of queries to updates and the number of replicated copies. Since a replicator copies only changed data rather than whole tables or table segments, less data is being transferred.

Reliability

High assurance - Data replication provides high assurance and availability because data is made available locally rather than through expensive and less reliable single central database systems.

Fault detection and recovery - How a replicator detects collisions varies depending on the replicator. It must be able to determine if the data to be updated on the target is the same as the source before the user's transaction. If not, a collision occurred. If a data collision is detected, some replicators may automatically resolve the conflict based on a pre-determined algorithm or log the problem so that the operator can manually correct. If a network or replica fail, the replicator can hold all updates in a persistent queue and the user can continue working with the local database. The local database utilities can then be used for backup and recovery. Some designs can in principle withstand catastrophic (permanent) failure of some of the replicated copies, although this is not guaranteed by all implementations.

Speed to field

Legacy systems - Data replication supports many heterogeneous databases. Legacy databases may be integrated with new systems in a very short time.

cost

Integration & Maintenance - The most obvious cost of the replication solution is the replicator itself. Some vendors require the purchase of a gateway and/or develop in-house programs to replicate between heterogeneous DBMS. Each vendor will have different requirements and costs associated with their replication solution. Replication solution may require additional hardware purchases such as disk space, replica system, or possibly the source system.

Extendibility

Coverage/Scope, Hardware/Operating Systems - **Most** replicators such as OmniReplicator, InfoReplicator, or DataPropagator support the sharing of data across a variety **of** operating systems such as Windows NT, OS2, Mainframe, AS/400, and Unix.

Coverage/Scope, Languages – This depends on availability of interfaces/bindings for the DBMS used.

Coverage/ Scope, Synchronous/asynchronous protocols - Data Replication supports both synchronous and asynchronous replication. In Synchronous replication all replicas remain fully synchronized at all times using two-phase commit. Asynchronous replicator updates the replicas only after the user transaction commits the changes to the local database. With asynchronous replication, there will be some lag period, when the local and remote databases are inconsistent.

Support for intelligent agents - **Most** replicators provide support for a mobile user environment by allowing on-demand replication for occasionally connected and mobile systems. Sybase's replication server for example, has a replication agent that works with the SQL Remote feature in its workgroup database product, SQLAnywhere.

Support "hooks" for duo-ins - Most replicators comes with integrated data enhancement, data collision detection and resolution, and heterogeneous middleware capabilities.

Maintainability

Maintainability all depends on how tightly integrated the replication solution is with business applications, database management, and operating system. The vendor usually assumes the responsibility for the changes in technology generally on maintenance agreement to upgrade systems and applications.

COTS support

The level of maturity for data replicators is reasonably good. The following vendors provide replication products:

1) Computer Associates-CA-OpenIngres/Replicator 1.0
2) IBM-Datapropagator Relational
3) Informix-Informix Continuous Data Replicator
4) Lotus-NotesPump 2.0
5) Microsoft SQL Server *6.5*
6) Oracle-Oracle 7.x Enterprise Server advanced Replication
7) Sybase-Sybase
8) Omni-OmniReplicator

Security

Data replication's solution to the issues with security is to maintain the complete database on a highly secure system with very limited access while replicating less sensitive data to another system offering broader access.

Feature Checklist:

	Products		
Feature	InfoReplicator	OmniReplicator	DataProagator

Heterogeneity- Sources	DB2, Informix, Oracle, Sybase, SQL Server	DB2,Informix, Oracle, Sybase, SQL Server	DB2, Oracle, Sybase, SQL Anywhere, Informix, Microsoft SQL Server, Microsoft Jet, and Lotus Notes.
Heterogeneity- Targets	DB2, Informix, Oracle, Sybase, MS SQL Server	DB2, Informix, Oracle, Sybase, MS SQL Server, DB2/6000, Ingres, Rdb	DB2, Oracle, Sybase, SQL Anywhere, Informix, Microsoft SQL Server, Microsoft Jet. and Lotus Notes.
Data type conversion	Automatic	Automatic	Automatic
Accept different source & target naming standards	Check	Check	Check
Bi-directional capabilities	Check	Check	Check
Conflict detection	Check	Check	Multiple levels of conflict detection
Conflict resolution	Last update, or user-exit selection of winning transaction	Last update, or user-exit selection of winning transaction	user-controllable conflict compensation
Change Capture	Check	Check	Check
Horizontal partitioning	Check	Check	Check
Vertical partitioning	Check	Check	Check
Data Enhancement	Built-in routines & provision for user routines	Built-in routines & provision for user routines	User routines
Synchronicity	Asynchronous	Asvnchronous	Asvnchronous
Latency	Near-real time Replication	Near-real time Replication	Near-real time Replication
Log or Trigger-based	Triggered & Logged	Triggered & Logged	Triggered
Fault-tolerance	Uses persistent data queues with native database recovery support	Uses persistent data queues with native database recovery support	Uses persistent data queues with native database recovery support
Routing Options	Single Target, Broadcast, Hierarchical, Cascade	Single Target, Broadcast, Hierarchical, Cascade	Single Target, Broadcast, Hierarchical, Cascade
Scheduling Options	Continuous, periodic, time of day, one time or manual	Continuous, periodic, time of day, one time or manual	Continuous, Update-anywhere replication
Transaction integrity		Check	Check
Application Independence		Check	Check
Netting of changes	No	No	No
Push or Pull	Push	Push	Push
Replication Administration	Windows-based GUI InfoDirector	Windows-based GUI OmniDirector	Windows-based GUI DataPropagator
Supporting software	IBM Datapropagator Capture/MVS for DB2 sources.	IBM DataPropagator Capture/MVS for DB2 sources.	DB2 for MVS/ESA™V4, DB2 for OS/390 V5, and DB2 for OS/390 V6

Coding Requirements	Non for standard replication. Optional enhancement & collision resolution extension through user exits	Non for standard replication. Optional enhancement & collision resolution extension through user exits	

Messaging

Performance

Messaging performance evaluation is based on the type of messaging, the messaging standard selected, the synchronicity required, and the hardware/network configuration servicing the message.

Synchronous messaging, whether RPC or simple messaging, has the following performance factors when used in a distributed system:

- decreases the throughput of the sending process when it is blocked and must wait for a response from the receiving process. (-)

- decreases latency (the process has a shorter response time) because although the message is affected by all of the network latency factors (propagation, transmission, routing, and storage), it is demanding the quickest response that the requested process can provide and does not have to wait an indeterminate amount of time for processes that may not be currently available to respond to the message. (+)

- less space required due to smaller buffers and less complex code to implement. (+)

- increased bandwidth required due to the consumption of unneeded resources in persistent connections while processes wait for send and receive messages. (-)

Asynchronous messaging, whether RPC or simple messaging, has the following performance factors when used in a distributed system:

- increases the throughput of the requesting process(not blocked, can issue other requests without having to wait for receipt of acknowledge messages. (+)

- increases latency(the process must wait longer) because the requested process does not have to be connected at the time of the request. This may create lengthy latency delays, but certain types of applications are able to perform their mission with minimal concern for latency issues. (-)

- increases space required with message queue structures, increased buffer requirements, and increased complexity of code. (-)

- decreases bandwidth requirements due to the use of non-persistent connections that minimize usage of network resources. (+)

Messaging type Comparison: Simple messaging has better performance in all categories than remote procedure calls due to the increased complexity and overhead of RPCs.

Messaging Standards: Messaging standards affect all categories of performance. The key standards that affect performance are size and header protocols. As message size increases, latency, space requirements, and bandwidth requirements all increase, while throughput and CPU performance generally decrease. Improvements can be realized in CPU performance, latency, and throughput, by standardizing message headers to optimize the assembly and disassembly of messages into their required parts.

Hardware Design: Optimizing the design of hardware for the handling of messages can have dramatic impacts on latency and throughput(i.e. translation look-aside buffer in memory management). The largest impact hardware can have on messaging is when groups of computers are arranged to act as one computer and the individual terminals are nodes. This is known as clustering and can create blocks of computers with equal or greater computing power then that of a single large machine.

Reliability

Reliable message passing guarantees delivery of the message through error checking, acknowledgement, retransmission, and re-ordering of out of sequence messages. The requirements for reliability depend on the process or application. Certain messages will require a acknowledgement of receipt, while others (i.e. broadcast) are informative in nature and may require no response. Reliability is bolstered dramatically by the use of clustering, which allows **for** the highest degree of fault tolerance. Many of the messaging software applications have added or are currently upgrading their applications to take advantage of this new configuration for server applications.

Messaging supports high assurance reliability through both asynchronous and synchronous messaging that uses error correction at both the network level and the application level, message acknowledgement, message retransmission, and message queuing. Fault detection and recovery (fault tolerance) is achieved effectively through error checking and either asynchronous message queuing which provides message logging and recovery or synchronous messaging with degradation **of** CPU performance and increased latency and space requirements. Clustering has also been shown to be a very effective technique to improve redundancy and deal with a single node failure which could result in loss of service. No information was found on the ability of messaging to deal with fault isolation except for the idea of clustered server applications adjusting to deal with node failure. [Stal97] No statistics were available on mean time between failure for messaging applications.
[Nels99]

Speed to field

Vendor software messaging applications are currently fielded to meet the needs for message passing communications for both legacy and developing systems. Vendors are poised to rapidly implement messaging solutions with COTS packages that can be tailored to meet the needs of distributed heterogeneous systems.

Legacy systems - Upgrading legacy systems is usually done using a synchronous messaging system [TIBC99]. This results in the legacy application code controlling the messaging rather than a external message queuing application. Connecting legacy systems together will be slower to field using messaging systems rather than connecting legacy systems to newly developed platforms that have integrated the new message queuing and clustering into their architecture.

New systems - Integration of new messaging technologies into future/new systems should be accomplished with short integration timelines due to the rapidly expanding markets for messaging application in enterprise business applications. The pace at which the business market is integrating COTS solutions into their new products to achieve new levels of interoperability between heterogeneous systems and applications clearly demonstrates the speed with which fielding can be accomplished.[Brow99]

cost

Integration - Integration costs **for** retrofitting existing systems must be carefully weighed against the performance benefits of designing the new systems to work using the latest messaging technologies. No actual integration cost totals were discovered during research but each of the major developers recognizes the need to easily integrate legacy systems into the messaging environment to allow business to capitalize on previous investments.

Maintenance - Many of the messaging applications create high level abstractions of the network programming code required for messaging which significantly reduces the software maintenance cost by eliminating the need for network programming skills among developers. This should both speed products to field and deliver products for lower cost.

Life Cycle Costs - The increased scalability of messaging systems both through the use of clustering, expandable message queuing architecture and reusability of abstracted interfaces reduces lifecycle costs dramatically. A Senior Middleware Manager who implements technology solutions for telecom giant MCI, validated this recently when he spoke at the 1998 Application Integration Conference explaining that: "Each week, their Registry system uses a message broker to exchange over 70 million pieces of information across 140 different applications. Taken over a year, Registry is a three-billion-messagebroker. Last year alone, MCI estimated that they saved nearly $72m by eliminating redundant development and support costs. They predict savings of nearly $250 million over 5 years".[Brow99]

Extendibility

Messaging lends itself to extendibility when it uses high level abstractions to simplify the network programming that handles addressing and marshalling of the message.

Hardware - All messaging solutions can be limited by the hardware on which they run. However there is broad support across the industry for the widest base of system hardware, operating systems, programming languages and synchronous and asynchronous protocols.

A chart from TIBCO's Rendezvous Web Site illustrates the diverse level of support:

Platform support			
Hardware	Operating System	**OS** Versions	**Notes**
DEC Alpha	Digital UNIX	3.2 (3.0)	Formerly OSF/1
	OpenVMS	6.1	UCX 3.2 3.3, 4.0
	MS Windows NT	3.5.1 and 4.0	
DECstation	Ultrix	4.3	
DEC VAX	OpenVMS	6.1	UCX 3.2 3.3, 4.0
	VMS	5.5-2	UCX 3.2
Hitachi	HI-UX	4.0	
HP 9000/700, 800	HPIUX	9.0, 10.0	
	NEXTSTEP	3.2, 3.3	
IBM RS/6000	AIX	3.2.5, 4.1	
IBM System1390	OS/390	1.2	
	MVS	5.5.2	
IBM AS/400	OS/400	V3R1 (and later)	
Intel 386 (or greater)	Free BSD	2.1	
	Linux	1.2 and 2.0.x	
	MS Windows NT	3.5.1 and 4.0	
	MS Windows 95		

	MS Windows	3.1	Requires Winsock 1.1 stack
	MS Windows for Workgroups	3.11	Requires Winsock 1.1 stack
	NCR UNIX	4.0	
	NEXTSTEP	3.2,3.3	
	OS/2 Warp	3.0	
	SCO UNIX	2.0,3.2.5	Formerly Novell
	Solaris	2.4,2.5	
Motorola 88000	SVR4	4.0	
NCR	SVR4	3.0	
NeXT ("Black")	NEXTSTEP	3.2, 3.3	
PowerPC	Windows NT	3.5.1 and 4.0	
	AIX	4.1	
	Solaris	2.5	
Silicon Graphics	Irix	5.3	
Stratus I860	FTX	2.2	
Sun SPARC	SunOS	4.1.3	
	Solaris	2.3, 2.4, 2.5	
	NEXTSTEP	3.3	

Support for intelligent agents - Most intelligent agent activity in the messaging arena is centered on message queuing support to ensure high reliability through auditing message logs, rerouting message traffic in cluster node failures, and load balancing/mitigation for servers. Load balancing is one of the most promising areas where unattended transactional servers can start additional processes in the event of a process failure or creating new servers instances to handle increasing traffic without human intervention.

Support "hooks" for plug;-ins - One application explicitly mentioned hooks as a means to customize a messaging application to improve security and related services. However, this option is exclusively available to applications that are built using the specific design tools and is not a standard feature for legacy interoperability conversions.

Support for publish-subscribe messaging;: All applications supported this innovative new technology. Roy Schulte of the Gartner Group states: " middleware is the key enabler for the 'zero latency' enterprise and publish-subscribe systems will be the fastest growing form of messaging middleware through 2002 because of their inherent flexibility.". In asynchronous messaging systems where applications use self describing messaging and message queuing the idea of publish - subscribe messaging is the fundamental are that needs most attention. This feature allows for far greater flexibility but with message numbers growing at such a staggering pace, some systems flood queues with too many messages per second.[Brow99]

Maintainability

Maintainability always carries a cost and most messaging solutions achieve low cost maintainability due to their ability to abstract underlying complexity and reuse existing standards and code. All of the industries that have switched to distributed messaging systems have changed in order to remain competitive in their industry.

Messaging has enabled them to change their organizations, saving money and making them more effective. The only documented increased costs due to conversion have occurred as a result of modification of the hardware needed to handle the increases in message volumes generated by the use of store and forward, publish and subscribe, and subject based addressing. [Brow99] Reliability has been increased in all documented cases where message queuing using store and forward policy has been implemented.

COTS support

Level of maturity - Simple messaging and RPC messaging is a very mature technology with certain very recent developments that offer tremendous promise. Message queuing with associated store and forward, and publish and subscribe policies is a newer technology. Flow control and queuing administration policies are still under development as vendors have had to aggressively scale up systems to meet rapidly increasing demand once a system is fielded.

Names of vendors of competing COTS products - Success with developing reliable messaging redundancy has been achieved using currently deployed enterprise queuing applications such as the Microsoft MSMQ **2.0,** IBM MQSeries, BEA MessageQ, TIBCO Rendezvous, and Talarian Mqexpress and SmartSockets.

Security

Security for various messaging applications was available to achieve secure data transfer through encryption of the message in transmission and most applications are designed to comply with centralized security policy enforcement.

Ability to detect and prevent unauthorized entrance to systems - No information was discovered during research about applications that detected or prevented unauthorized entrance to systems. An external security kernel would perform features of this type. Digital signatures are one way to address this issue.

Immunity to disruption of network services - The use of clustering combined with the use of message queuing is the most effective way to achieve virtual immunity to network service disruption. Delta Airlines adopted this strategy in an effort to ensure that one and only one message is passed in their reservation and flight scheduling systems and that no data is ever lost in transit. [Brow99]

Ease of use - Measurement of ease of use for system administration using message queuing is documented clearly as the queuing policy allows logging and transmission of system errors to one centralized log using a publish and subscribe policy. This dramatically improves the administrator's ability to react to problems with information that was previously scattered across the network in inaccessible sites. Abstracted network programming constructs achieve dramatic ease of use for developers using most queuing applications.

Standards

With the development of self-describing messages and publish and subscribe policies the standards for messaging have become less restrictive. The use of XML in messaging promises to revolutionize messaging standards as the technology becomes more mature and expands to meet increasing need for messaging systems. However, it is a double edged sword because it enables proliferation of a variety of incompatible "standards" in the absence of organizational control.

ORBs

Evaluation Factors	CORBA	COM/DCOM
Authentication	Yes (via CORBA Security Service)*	Yes**
Logging and auditing	Yes (via CORBA Transaction Service)	Yes**
Encryption	Yes (via CORBA Security Service)*	Yes**
Access control	Yes (via CORBA Security Service)*	Yes**

The CORBA Services include the Security Service, which provides a complete framework for distributed object security. It supports authentication, access control lists, confidentiality, and non-repudiation. It also manages the

delegation ofcredentials between objects. However, the modelfor the services is still very basic, more ambitious and comprehensive mechanisms have not yet been adopted by the OMG.
*** The DCOM /COM+ v1.0 provides a more comprehensive security model. It supports role-based security, multi-level security, abstract security setting, confgurable access control, and confgurable authentication.*

Evaluation Factors	CORBA	COMDCOM
High Assurance	Work in Progress*	Yes (via queue component of the COM+ v1.0)
Fault detection and recovery	Yes (via CORBA Transaction Services)	Yes (via Transaction Services)
Redundancy	Some (not specified in CORBA Spec **2,** but some vendors have implemented load balancing and replication into their products)	Yes (via load balancing)
Constraint Checking	Yes	Yes

The CORBA Transaction Service and Concurrency Control Service provide reliable two-phase commit coordination among recoverably components using either flat or nested transactions. Interoperability between ORBs must be through the General Inter-ORB protocol (GIOP) or the Internet Inter-ORB Protocol (IIOP) as defined in the COMA 2.0 interoperability. Most of the implementation of GIOP are usually TCP, whereas the IIOP specifies how GIOP messages are exchanged over a TCP/IP network. Both the protocols provide reliable communication.

No major reports of unreliability are recorded for most of the major CORBA vendors. Although high assurance **CORBA** is still under construction, there are already a number of vendors such as **TIBCO, Inprise Corporation,** which incorporate some high assurance features into their products.

Evaluation Factors	CORBA	COMDCOM
Speed	Generally Fast (less than 5ms **for** Ping) but depend heavily on the vendors implementation	Fast (less than 5ms for Ping)
Overhead	Small for Static Invocation Large for Dynamic Invocation*	Small for Static Invocation Large for Dynamic Invocation
Real-time	Work in Progress**	No

The following test information is extracted from *Client/Server Programming with JA VA and CORBA by Robert Orfali and Dan Harkey:*

The program written is a simple count and ping program. The client invokes the server to increment the count and the results are then returned to the client.

Test program	Local interprocess	Remote interprocess over 10Mbit/s Ethernet
Static Count Ping (Java)	3.9ms	3.6ms
Static Count Ping (C++)	2.3ms	3.9ms
Dynamic Count Ping (Java) Invoke only	3.9ms	3.6ms
Dynamic Count Ping (C++) prepare and invoke	61.4ms	57.8ms

Datagram sockets (Java)	4.2ms	3.9ms
Buffered stream socket (Java)	3.7ms	3.9ms
Buffered data stream socket (Java)	1.7ms	2.1ms
RMI (Java)	3.1ms	3.3ms
DCOM	NA	3.8ms

Evaluation Factors	CORBA	COM/DCOM
Operating System/ Platform	Many	Mainly Windows system, have expended to other platform
Language	Most major languages are supported (C++, ADA, Java)	Most major languages are supported (C++, Java)
Intelligent Agent	No part of the CORBA v2.0 specification*	Yes, only in COM+ v1.0
Synchronous/ Asynchronous	Yes	Yes
Support hook	Yes, provides support for interface with existing legacy code.	No

Maintainability

Both CORBA and DCOM frameworks encourage readability and maintainability by encapsulating the services into an object with the interface described in an Interface Definition Language. The interfaces defined in IDL files serve as a contract between the server and its clients. However, DCOM is more manageable to use than CORBA. The CORBA IDL compilers produce more source files and require that the object and client developer write their code around the CORBA structure. The Microsoft IDL compiler does not produce piles of generated code, instead the DCOM API is used and objects are straightforward to implement. DCOM structures should be easier to maintain because they are simpler.

COTS integration

A large and growing number of implementations of CORBA are available in the marketplace, including implementations from most major computer manufacturers and independent software vendors (see Appendix B). Most of them have their success stories, for example companies embedding the Visibroker ORB into their products include Oracle, Hitachi, Netscape, Novell, Sunsoft, NetDynamic, Novera, Sybase, Borland, Gemstone, Silicon Graphics, Humming Bird, Cincom and Business Object. At the same time, it must be noted that not all CORBA ORBs are equally mature, nor has the OMA sufficiently matured to completely support the vision that lies behind CORBA. While CORBA and OMA products are maturing and are being used in increasingly complex and demanding situations, the specifications and product implementations are not entirely stable. Hence, for **an** object request broker (ORB) to be a candidate for mission-critical middleware it should have the following 10 essential features.

1. The ORB vendor should be a viable company, capable of providing long-term support.
2. The ORB should be CORBA 2.0 compliant.
3. The ORB should be present on Windows NT/95, Sun Solaris, and HP-UX platforms.
4. The ORB should have C++ and Ada95 bindings.
5. The ORB should have a Java/WWW binding or gateway.
6. The ORB should have an OLE bridge.
7. The ORB should have multithreading capability.

8. The ORB should support replication and fail-over servers.

9. The ORB should have a naming service.

10. The ORB should have an event service.

COM has its roots in OLE version 1, which was created in 1991. It was a proprietary document integration and management framework for the Microsoft Office suite. Microsoft later realized that document integration is just a special case of component integration. OLE version 2, was later released in 1995, with a major enhancement over its predecessor. The foundation of OLE version 2, now called COM, provided a general-purpose mechanism for component integration on Windows platforms. While this early version of COM included some notions of distributed components, more complete support for distribution became available with the DCOM specifications and implementations for Windows95 and Windows NT released in 1996. Beta versions of DCOM for Mac, Solaris and other operating systems followed shortly after.

COM and DCOM are most mature on Windows platforms. A component marketplace with a wide selection **of** COM-compliant objects has developed for Windows. However, the step from Windows platform-specific COM to distributed, heterogeneous support is quite significant. DCOM support must mature and component marketplaces must develop if DCOM is to thrive on other platforms. While the Windows component base provides a starting point, migrating existing Windows components to alternate platforms will require significant effort, since these components must be designed and compiled specifically for the host platform.
There are many PC-based applications that take advantage of COM technology. The basic approach has proven sound, and a large component industry has sprung up to take advantage of opportunities created by the Microsoft technology. On the other hand, DCOM is just now arriving on non-Windows platforms, and there is little experience with it. There are currently no reports of any large-scale distributed applications relying on DCOM support for heterogeneous computing environments at this moment.

cost

CORBA

The price of CORBA ORBs varies greatly, from a few hundred to several thousand dollars. Licensing schemes also vary. Some require only development licenses while others require separate licenses for development and runtime. In additional to the tools cost, training is also essential for the already experienced programmer.

DCOM

Low cost development tools from Microsoft (such as Visual C++ or Visual Basic), as well as tools from other vendors provide the ability to build and access COM components for Windows platforms. Construction of clients and servers is straightforward on these platforms. In addition, the initial purchase price for COM and DCOM is low on Windows platforms.

The situation for other platforms is less clear. The prices of DCOM versions on non-Windows platforms are generally more expensive than typical PC tools.

Beyond basic costs to procure the technology, any serious software development using COM/DCOM requires substantial programmer expertise.

However, Microsoft has a strong support organization to assist individuals developing COM clients and objects: many sample components, books and guides on the subject of COM development are available.

Speed to Field

Developing distributed applications is very complicated, taking a large amount of man months in coordinating, design and implementing. Beside the problems found in a heterogeneous network, such as different operating systems, different representation of data types, and different programming language, developers must also consider threading issues, threading options, concurrency problems, process management, and N-tier design complexity.

Both the CORBA and DCOM framework actually make development of distributed applications easier by hiding the information and complexity. The developers can now be only concerned with the object interface details and not the mechanisms. Reuse of interface components also speeds up the development process.

JINI

The Promise of Jini

Jini promises to make interoperability between systems in a networked environment simple to establish. Jini allows systems on different platforms and operating systems to share access to the services available on a network. The only requirement is that the participating systems have access to a JVM **or** even a sub-set of the JVM. In reality, however, much testing and evaluation is still needed in order to determine the feasibility of Jini in practice.

Assessment Factors

Performance

Very little data is available concerning the performance of Jini. Jini does rely heavily upon RMI for network communications, which can be slower than raw socket connections and is comparable in speed to many ORBs. RMI, Jini, and Java should not be relied upon for critical real-time systems. The actual execution of any code within Jini may be slower than Java code run on a stand-alone system, since some service methods may involve remote procedure calls to the remote service provider.

Reliability

The Jini system is as reliable as the Java Application Environment. Network connections and errors are handled via the underlying RMI protocol, which is built upon TCP/IP. Jini includes protocols for events, security, and transactions in order to ensure a stable distributed computing system. The goal here is that the user should not even be aware that the system is using a service from a network, so Sun Microsystems has built in reliable distributing protocols to make this possible.

Speed to Field

Jini presents a fast and easy way to field non-real-time critical systems that can interoperate. For example, a printer service can be written in Java and made available via the network. Any system that can run the JVM can then connect to the Jini system and use the printing service. A database on one system can be made immediately available to all Jini-capable systems as soon as the systems connect to the network. Other than the JVM and the ability to lookup Jini services, no other code is required on the client systems. The services are available as soon as they connect to the Jini system.

Legacy systems that can run the JVM can be Jini-enabled and can then access all Jini services immediately. Since all parameters and return values are sent across the network as Java objects or references to Java objects, no data translation is required if JAVA bindings are available.

The difficulty in fielding Jini systems is that few programmers have learned the programming model for building distributed systems using Jini, since it is very new.

cost

Jini and Java are free. Costs would be incurred by creating Jini Services that can talk to legacy systems, and since Java and Jini are free, it would be less expensive to use Jini than to use CORBA. Some systems, however, may not be able to use Java at all. Costs for new systems built with Jini would be relatively inexpensive.

Extendibility, Maintainability, COTS support, and Security

Since Jini is an extension of the Java Application Environment, it should be comparable to the Java Application Environment for these factors. One factor that may affect the implementation of Jini differently is its level of maturity. Jini is not in wide-spread use and has not been studied in academia as much as Java Applications, and any research conducted in industry has been kept closely held as trade secrets.

4. Description of Technologies

Wrappers

Wrapper Definition: Wrapping is an approach to protecting legacy software systems and commercial off-the-shelf (COTS) software products that requires no modification of those products.

A wrapper consists of two parts, an adapter that provides some additional functionality for an application program at key external interfaces, and an encapsulation mechanism that binds the adapter to the application and protects the combined components. Wrapping should require no changes to the application program.

[Meeson 1997] Candidate mechanisms for implementation via wrappers: authentication, logging and auditing, constraint checking, encryption, access control, fault detection and recovery, redundancy.

Key interfaces between application programs and their supporting environment were identified as the most accessible insertion points for adapters: library services, operating system services, services provided by separate processes, and services provided by external process outside the local software environment (i.e. network proxy services). The implementation of wrappers can involve some of the other lower level building blocks surveyed, such as messaging and data translation. Wrappers are most useful when they are used in a structured way to realize a standardized interoperability architecture.

Several techniques for wrapping legacy and COTS application software are available and have been employed in commercial products. Practical aspects of wrapping such as accessibility of interfaces, accuracy of interface specifications, and protecting adapter code from being bypassed, however are open problems that will need to be addressed to enable broad use of these techniques for security and survivability hardening. At present normal software maintenance processes can probably provide higher assurance for the functions considered to be placed in wrappers.

Candidate implementations fall into two categories:

- Scripting languages such as Perl, Tcl, UNIX shell scripts, Awk, etc.
- Compiled languages such as Java, Ada95, C/C++, etc.

Translators

Data translation is one method used to address data interoperability. Data translation is where stored data is translated from one representation to another, more widely accepted form or standard. By itself, data translation is a very crude method of achieving interoperability:

- A translator must not only literally transpose, but must be able to deduce or infer the semantics of the data in order to translate correctly.
- Translators are static. Data representations will continue to evolve and new standards adopted. A standard provides leverage on the part of the users in getting software vendors to provide a translator for widely used standards, which enhances the interoperability of different software systems. Adopting and utilizing a single comprehensive data standard minimizes the time and cost of bringing in new data from other sources. However, standards also evolve and as new standards are adopted, new data translators will have to be developed or old ones extended.
- Most translators are written to address a specific problem domain.

Data translators can vary in scope and functionality. For example, Microsoft and other product vendors incorporate data translators into their products to enhance interoperability between their software and software developed by competitors. This has been motivated by customer demands and needs.

Examples of available data translators include:

Microsoft provides
- translation among versions of Microsoft software
- import/export utilities to interface to competitors or complementary products

Adobe provides
- Translators among postscript, PDF, and other vendor specific document formats

Most vendors, including database suppliers, provide
- data export facilities (including ASCII)
- data import facilities

Program conversion utilities (i.e., Fortran to C, etc)

Program conversion utilities can provide a working version of an existing program in an alternative implementation language. Unfortunately, the translated code is very difficult to maintain unless code analysis and re-engineering routines are utilized. The refine system from Reasoning Systems, Inc. is typical of the best available environments to support creation of such high quality transformations. The tool is very powerful but users need extensive training to reap its benefits. Translator generation is another rapidly developing technology relevant to data translation. Software engineering research has demonstrated a domain-specific translator generator that allows application domain experts who are not expert in software and translator design to generate software for interfacing to C31 messaging formats [Kieb96].

Data Mediators

Introduction

Solving the problem **of** heterogeneous data sources can be accomplished using a mediator. Mediators are designed to allow application users to access multiple heterogeneous data sources without concern for the differences in schema. Mediators are also capable of transforming data into information by applying expert system knowledge on the heterogeneous data, protecting database sources from hostile action, as well as mediating access to data to maintain security policy. Mediators are special kinds of software agents.

Definition

In its most fundamental definition, a mediator is a computer program which translates data between two systems with different schemas.[RRSc96] The definition can be extended to performing translation between a indeterminate number of schemas interconnected using a given network protocol, performing data mining to create user views that draw from an array of translated data from multiple databases, performing security mediation on data to ensure that the aggregation of data does not violate security policy, and survivability mediation that ensures that a database survives a hostile attack. Mediators are usually applied to databases. They comprise more general versions of traditional computed views that can involve inference rules and arbitrary computations in addition to queries expressed in common query languages. Mediators are in a sense the database analog of wrappers.

Principles of data mediation

Mediator Architecture: Mediation architecture is generally divided into three layers: the User Layer which consists of independent applications, a Mediator Layer that contains multiple mediators that create useful (clean) information for the user layer, and a Base Layer made up of multiple heterogeneous databases. This architecture supports the legacy client server paradigm by decoupling the client from the server to allow for scalability of the system and creating an opportunity for affordable long-term maintenance of the system.

Mediation is broken down into a number of different types. Basic data mediation has been in existence since the first translator program was implemented to convert a schema. However, newer technologies such as information fusion (value-added mediation), security mediation and survivability mediation add significant value to the basic mediation transaction and actually provide partial solutions to the problems of information security and information management.

Basic Data Mediation: In the simplest form of data mediation between two systems with different schemas, the mediation process is divided into two steps: translation of a receiver's query into a source schema, and translation of the retrieved source query into the receiver's schema. This allows the mediator to act as a "semantic gateway" between the two systems and allows the receiver database to consider the source database as an extension of itself.

Accomplishing this task requires development of schemas that are written in a formal language, use of a standardized common vocabulary to describe the actual data, development of conversion functions to transform data into the standardized common vocabulary, and testing to ensure the rules adequately describe the required mediation. The standardized vocabulary is also known as a "semantic domain", and describes both the meaning and the properties of the data. [RRSc96]

Once the mediator is in place, each independent database is free maintain its unique schema and evolve as needed. The mediator requires updates to the interface when changes to the independent database schemas are required.

Basic data mediation is controlled easily with great flexibility when it occurs on an independent server in the mediation layer between the multiple processes. Programs designed to perform large data mediation services are known as Transformation Engines/Hub Tools. [Cons98] These engines frequently support data warehouses that collect carefully screened "clean" information that is selected by information fusion mediators.[Dawa96]

Information Fusion: Also known as "Value Added Mediation" [Wied98], information fusion mediation is capable of aggregating information about "same world" entities that exist in multiple heterogeneous databases and presenting/collecting useful information about the data. Mediators can use rule-based reasoning to remove redundancies and resolve conflicts. This use of a this type of mediator introduces the issue of domain and semantic integration.

Domain Integration: The process of physically linking a new source of data or reasoning system to an existing mediated system so that the resources provided by the new system may be accessed by the existing mediators. [Herm97] This process should be repeatable and clearly defined with a definite sequence of steps and should comparable to the process of connecting a new node to a network.

Semantic Integration: The process of specifying methods to resolve data conflicts, aggregate information, and define new compositional operations. [Herm97] This process is less clearly defined due to the fact that there may be no unique way to resolve a given conflict. Design emphasis is on conflict detection and handling. In addition, aggregation of information is accomplished by designing flexible modules using an expert's domain knowledge with the understanding that this is one of the largest areas of change in a mediation system.

Separation of these two integration issues allows for improved modularity and abstraction, which improves scalability and maintenance.

Data Warehouses: The requirement for data mediators to react to the changing needs of the decision-maker has forced designers to develop flexible tools that can construct data warehouses and data marts [Dama93] rapidly. The steps for development of a data warehouse using a mediator are:

- Conduct Source Data Extraction
- Perform Data Cleaning(Conflict resolution)
- Multi-dimensional aggregation
- Warehouse Load

A loaded data warehouse is a subject-oriented, integrated, time-variant, nonvolatile collection of data in support of management's decision needs. [Inmo95]. This makes additions to the warehouse possible but prevents future updates to the existing information. The role of a mediator is to rapidly and correctly establish the warehouse and provide additional data to enrich the value of the warehouse.

Security Mediation

There are two different variations of security mediation: mediation that provides rule based reasoning to recognize the aggregation of information that may violate security policy, and mediation that maintains secrecy and privacy while allowing access to anonymous collections of information. **As** heterogeneous data collections are combined using distributed systems the classification and compartmentalization schemes of various systems must be mediated to resolve inconsistencies in classifications of "same world" information, and recognize the creation of user views that contain classified information drawn from a collection of unclassified sources. In addition there are civil and military organizations that can benefit from the ability to evaluate mediated anonymous collections of information used to make decisions that collect information without violating security or privacy policies. This form of mediation is similar to information fusion in the integration tasks and requires extensive use of expert systems to develop successful rule based semantics. [Dawl98b].

Survivability Mediation
A survivability mediator is one that enforces a policy that satisfies a survivability requirement between two processes. This type of mediator is designed to protect an aspect of the mediated system from damage or corruption from hostile actions taken by the other process. **A** survivability mediator is inserted between a client and a server process with a set of rules to enforce on the messages between the two processes. It requires the survivability requirement is specified with a clear set of rules for implementation, but allows the progressive development of the survivability policy independent of the processes it mediates. The location of survivability mediators is generally at the architectural boundary between processes or systems. [SGSh98]

Mediation Structures and Tools
Summarization: Summarization provides aggregated data, following the hierarchy established by an object model within the mediator. In systems without mediators summarization is done by the client, and, to the extent that SQL functions are available, executed by the server. Often the source data must be filtered to delete anomalies and perform conversions if data come from multiple countries which places a large load on the client application.[Wied98] The information fusion mediator is used to relieve the load on the client and decouple the client from the server.

Exception Seeking: This is an alternate abstraction that is created by seeking for exceptions. It is a tool that allows the mediator to track deviations from the rule and use this information to aggregate information in the information fusion type mediator. [Wied98]

Data Standardization: Standardizing Data at the database level is good long-term technique to reduce the problem of heterogeneous representation of data. However standards will change over time which forces the requirement for the development of a formal language (meta-data) to describe the data and allow all legacy and new systems to inter-operate.[RRSc96]

Wrappers: Wrappers provide the mapping from the mediator to the specific data source view in basic mediation with multiple stand-alone databases. They are used in conjunction with data mediators when a user application must get updates from data sources that are undergoing changes during use. This raises the issue of data currency that must be resolved with rule based logic in the mediator.

Application [Sullivan 1998]
A survivability mediator is a mediator that enforces a policy that satisfies a survivability requirement. The mediator waits for connections from the primary or the secondary server, buffers the message being sent, then implements policy actions depending on the content seen. Other applications include security (encryption, authentication, monitoring for intrusion), network optimization (data compression, load balancing), and bridging between languages or subsystems (data translation, exception encoding, protocol translation).

A mediator approach eased the evolutionary survivability enhancement of a critical infrastructure system by allowing the imposition of a new integrity constraint without changes to the existing source code. Enhancement will not be so easy for all systems; but transparent insertion of mediators at natural architectural boundaries is a general strategy independent of the type of boundary: CORBA procedure invocation, Internet event notification, tile system call, etc. The mediator approach has the added benefit of localizing the survivability policy implementation, making it possible to change both it and the surrounding system independently. This dimension of evolvability was exploited in exploring a range of policy specifications and implementations. In general, policy evolution promises to be important for future infrastructure information systems--e.g., to keep pace with increasing capabilities of hostile adversaries.

Replicators

Data replication is a method of distributing data (usually database data) from one or more sources to one or more targets. The structure is similar to the stubs used in **RPC** and the service objects used in JINI, except that full copies of the data are maintained at the remote locations. There are two classes associated with data replication, synchronous and asynchronous.

Synchronous replication performs replication within the source update transaction. The source transaction is not considered complete until the update has to be applied to all replicas. Synchronous replication can be provided by database management system software (DBMS) or by distributed TP monitor/transaction manager. In either case, every update within the synchronous replication process occurs as one logical unit of work. In other words, it is an all-or-nothing transaction. Synchronous replication is used where high availability and concurrency are absolutely essential. It is tightly integrated with both the hardware and operating system and does not scale well.

In asynchronous replication, the source update is independent of the replication process. The user's transaction completes when the local update is complete. The replicator updates the replicas only after the user transaction commits to the changes to the local database. Replication may occur moments after the source transaction, in near real-time, or it may be scheduled for later execution. The delay might be in seconds for continuous replication, or longer for scheduled replication. For most applications, the momentary lag is outweighed by the benefits of asynchronous replication.

Asynchronous Replication lowers the cost, improves concurrency within resource managers (shortens the time any data is locked), and generally shortens the length of the originating database transaction [Buretta97].

Benefits of Synchronous Replication
- All replicas remain fully synchronized at all times.
- The use of two-phase commit eliminates the possibility of data collisions.

Benefits of Asynchronous Replication
- With less processing attached to the user transaction, the user regains control of the system sooner.
- Users are not directly affected by network delays or slow remote processors.
- Users can continue work even in the event of network or remote database outages.

The advantages of data replicators are that they support: cross-platform, cross-database data movement, bi-directional replication, high performance, network reduction, and requires no additional programming if an appropriate DBMS or TP is used.

Messaging

Introduction
Messaging or message passing is one of the key concepts that is being utilized to realize distributed processing systems. Messaging is used at many different levels within standalone systems and networks to achieve varying degrees **of** interoperability. It is found in the lowest level of network interconnectivity, in the complex mechanisms used in software interoperability and at the highest levels of business collaboration. It is an approach that is fundamental to emerging trends in enterprise application integration (EAI).

Definition
A message in its simplest form is a block of information that is exchanged between two or more processes as a means of communication. The two processes could be located on the same host computer or on opposite sides of the globe on two or more different host systems connected by a network. In addition, the term "messaging" takes on a variety of different meanings within the context of distributed processing systems and among different messaging enabled and messaging reliant applications.

Principles of Messaging

Message passing in distributed systems may be broken down into two basic concepts: simple message passing and remote procedure calls.

a. Simple message passing: Simple message passing is similar to the normal use of messages within a stand-alone system. The processes that need to communicate make use of external message passing modules that take primitives (operations) and parameters (data and control information). These modules respond to "send" and "receive" primitives, maintain buffers for "receive" and "send" messages and use a given communications protocol. There are two primary criteria that affect the performance of message passing, the level of reliability desired to ensure the message is received, and the actions a process takes when a message is passed.

1) Reliability: Reliable message passing guarantees delivery of the message through error checking, acknowledgement, retransmission, and re-ordering of out of sequence messages. The requirements for reliability depend on the process or application. Certain messages will require a acknowledgement of receipt, while others other (i.e. broadcast) are informative in nature and may require no response.

2) Blocking: Blocking primitives, otherwise know as synchronous primitives, suspend processes during send and receive operations while the message transmission is conducted. This technique is utilized to minimize timing errors, protect the message contents, and protect the state of the processes from change. In contrast non-blocking (asynchronous) primitives allow flexible use of the message passing modules without process suspension and notify processes on arrival of a reply message through interrupts or polling. The disadvantage of the non-blocking approach is the difficulty in solving subtle, complex timing problems. These two different blocking techniques are used in synchronous and asynchronous message delivery and can be used in very powerful ways both independently and in conjunction to achieve interoperability between heterogeneous systems.

b. Remote Procedure Call: Remote procedure calls (RPC) are a variation of the basic message passing concept. Generic remote procedure calls are familiar to most developers and programmers as routine calls from programs to subroutines. RPCs can be used across the spectrum from reliable, blocking message passing for certain applications to unreliable non-blocking message passing for less critical operations. In order to execute the RPC, each application has an internal remote procedure call stub, which has the external attributes of the remote module being called. The stub programs are used with the RPC mechanism (RTS - Run Time System) to transparently invoke a corresponding procedure on a remote process. The steps of a remote procedure call are:

- the process places the parameters into the stub procedure (marshalling),
- invokes the RTS to send the message,
- waits for the reply,
- unmarshals the return values,
- passes the return values back to the variables in the calling process.

The required interface should be clearly documented to allow remote users of the application to know exactly how to invoke the procedure and what primitives or types to pass. This scheme allows developers to write client/server modules that are independent of communications hardware, protocols, and operating system.

Remote Procedure Calls must deal with issues regarding parameter passing, parameter representation, addressing, client/server binding and synchronous/asynchronous behavior.

1) Parameter Passing: Parameter passing must deal with the issue of passing information by value or by reference. Passing pointer values by reference across heterogeneous systems is viewed as difficult, and frequently involves large amount of overhead to ensure that the pointer information is relevant to the remote system.

2) Parameter Representation: Parameter representation is another difficult issue that is handled through standardization of data formats, development of complex communication architectures, and most recently in the use of field manipulation languages (FML), and extensible markup language (XML) that self describe the data types, structures, names, and size of the messages.

3) Addressing: Addressing is critical to RPC success. When the client process is executed, the client stub must be connected via the RTS to the matching server stub. This requires the RTS to take the logical name of the service required and use it to find a suitable server due to the fact that the client is not aware of the server's actual physical address. Many different address resolution schemes are available. Subject based addressing is the latest technology in the field.

In this scheme, the subject name of a message is a character string that describes the contents and the destination of a message. This scheme frees programmers from the details of network addresses, protocols, packets, ports, and sockets that disrupt the interoperability of system. Clients and servers subscribe to various subjects' names, which enables them to listen for messages (respond to RPC) and publish information on subjects (request service using a RPC). This allows for dramatically improved maintainability and flexibility as message brokers can dispatch messages easily using subject names and look-up tables.

4) Client/Server Binding: Another key issue is the specification of client/server relationship that establishes the connection between the remote procedure and the calling process. This binding is classified either as non-persistent or persistent. Non-persistent binding terminates a connection between processes after the transaction is complete which reduces the resource requirements but makes frequent interactions incur a large overhead. Non-persistent connections are becoming more common with the ever-increasing amount of distributed mobile computing and make use of message queuing technologies. Persistent binding creates an open connection that endures beyond the end of the call and is an effective use of resources when a repeated series of calls must be made. Disconnection is usually handled through a timeout. Persistent binding is critical to certain types of broadcast messaging with both IP broadcast and multicast addressing.

5) Synchronous and Asynchronous RPC: Synchronous and asynchronous remote procedure calls are comparable to blocking and non-blocking messages. Synchronous RPCs require the calling process to wait until the remote process has returned a value, which makes the transaction easy to understand but may degrade performance in a distributed system when parallel computations could be running. Asynchronous remote procedure facilities allow greater flexibility and parallelism because they allow calling programs to issue multiple calls to the same or different remote procedures. This asynchronous ability allows for interruptions in service, improves scheduling flexibility, and help mitigate high loads on a requested process. Asynchronous **RPC** is expanding rapidly with the development of message queuing (see paragraph b below).

Message Addressing Solutions

Message addressing solutions are presented here to provide basic information about some of the vendor supported solutions available on the market today.

a. Message Brokers: A software entity that provides addressing services and handles dispatching of messages. Most message brokers use an asynchronous transport layer that corresponds to an open or proprietary messaging system. Many message brokers can leverage IBM's MQSeries message queuing software, or more open systems such as the Java Messaging Service (JMS) from JavaSoft, or the Object Messaging System (**OMS**) from the Object Management Group.

b. Message Queuing: The idea of message queuing is very straightforward. An application builds a message and sends it to a queue. Some other part of the application (or some other application) can then read the message from the queue. If needed, this receiving process can respond, again by sending a message into some queue that is later read by the original sender or another process.

There are certain situations that lend themselves to normal remote procedure calls rather than use of higher level message queuing. RPCs are typically simpler to use than message queuing, but provide fewer services. A short list of checks to use in determining the optimal solution is included from an article on MSMQ in the Microsoft Journal:

- If a requesting process must wait for a response from the receiver before proceeding, an RPC should be used (since a call blocks until a result is returned). If the requesting process does not need to wait for a result, but can perform other tasks, message queuing is most likely the better choice.

- If the client and server applications may run at different times, use message queuing since most simple RPC calls are designed to work when both client and server are available at the same time.

- If a client is publishing using subject addressing and isn't sending to a specific server, but instead to any one of a group of receivers, use message queuing. Once a message has been placed in a queue, it is potentially available to any application that can read from that queue. With **RPC,** by contrast, a client typically makes a call to a specific server.

- If requests need to be logged and possibly reprocessed to recover from failures, use message queuing. While it's certainly possible to build logging into an RPC-based application, mechanisms to do this are an intrinsic part of many message queuing applications.

- If your client process makes a simple call to a server and moves on with minimal future interaction an RPC will normally be most straightforward, but as the complexity of interactions between the parts of a distributed application increases, switch to use message passing/queuing. With Message queuing, A can send a message to B, who sends one to C and D, who both send response messages back to A. This kind **of** flexibility isn't possible with RPC (most current compilers are incapable of automatically making such optimizations for distributed computations).

c. Network Aware OS(Inferno): Lucent Technologies has taken the approach to develop an operating system, virtual machine and language that are designed to be already network-aware. Mechanisms are added that allow diverse processors and operating systems to communicate and interact, no matter in which language an application has been written. The operating system is Inferno, the language is Limbo, the virtual machine is DIS and the distributed-computing mechanisms are Styx-an all-in-one networking protocol-along with Inferno's "namespace" feature and its more recent extension to incorporate diverse OS's and languages, InfernoSpace.

The model that Inferno was built on has three basic principles. First, all resources are named and accessed like files in a forest of hierarchical file systems. Second, the disjoint resource hierarchies provided by different services are joined in a single private hierarchical structure called "Namespace". Third, a communication protocol, called **"Styx",** is applied uniformly to access these resources, whether local or remote.

The binding that connects the separate parts of the resource namespace together is the Styx protocol. Within an instance of Inferno, all the device drivers and other internal resources respond to the procedural version of **Styx.** The Inferno kernel implements a mount driver that transforms file-system operations into remote procedure calls for transport over a network. On the other side of the connection, a server unwraps the **Styx** messages and implements them using resources local to it. Thus, it is possible to import parts of the namespace-and thus resources-**from** other machines.

The **Styx** protocol lies above and **is** independent of the communications transport layer; it is readily carried over TCP/IP, PPP, ATM or various modem transport protocols. Basically, Inferno creates a standard environment for applications. Identical application programs can run under any instance of this environment, even in distributed fashion, and see the same resources. Depending on the environment in which Inferno itself is implemented, there are several versions of the Inferno kernel, Dis/Limbo interpreter and device-driver set.

When running as the native OS the kernel includes all the low-level glue, such as the interrupt handlers, graphics and other device drivers needed to implement the abstractions presented to applications. For a hosted system under Unix, Windows NT or Windows 95 Inferno runs as a set **of** ordinary processes. Instead of mapping its device-control functionality to real hardware, it adapts to the resources provided by the operating system under which it runs. For instance, under Unix, the graphics library might be implemented using the X-window system and the networking using the socket interface; under Windows, it uses the native Windows graphics and Winsock calls. Inferno is, to the extent possible, written in standard C and most of its components are independent of the many operating systems that can host it.

d. Universal Plug and Play(UpnP): Multicast DNS is a feature of UpnP that allows a simple service discovery protocol(SSDP) to run on a network and allows services to detect and identify other devices on a network. Clients send out a User Diagram Protocol(UDP) multicast packet that contains an identifier for a desired service on a standard channel. Other services would listen to the channel, read the request, determine whether they can provide the service, and if able, respond to the request for service.

e. MSMO 2.0: MSMQ is a rich set of runtime services that allows applications on different machines to send and receive asynchronous messages to and from each other. It first appeared in the Windows NT® 4.0 Option Pack in December of 1997, and is an integral part of Windows® 2000. An application that wants to receive messages creates a queue, a persistent administrative structure known to the operating system that is conceptually similar to a mailbox. An application that wants to send a message to another application locates this queue, either through system services or through application-based foreknowledge, and uses the MSMQ service to send a message to it. An MSMQ message can contain plain text or COM objects that support the IPersistStream interface. Messages pass through the network topology configured by the system administrator until they reach the recipient machine. **If** the recipient queue's machine is not running when the message is sent, the message is buffered by the sending machine's operating system or the intervening network machines until the recipient machine starts running. An application can receive MSMQ messages either through polling its queue or using callback notification.

MSMQ also contains a rich set of features to support this basic message delivery service. For example, the sender of a message may specify that the message will expire and be deleted if it does not reach the recipient's queue within a certain time interval, or if it is not read from the queue by the receiving application within a different time interval. The sender may ask to be notified when the message is received, or only if it expires unread, or not at all.

Messaging Features

These messaging features are actually application or language solutions that are being used in conjunction with the addressing solutions in currently fielded vendor applications.

a. XML(Self Describing Messaging): XML is a markup language for documents containing structured information. Structured information contains both content and information about how the information is used. Almost all documents have some structure. A markup language is a mechanism to identify structures in a document. The XML specification defines a standard way **to** add markup to documents and to define the intended operational meaning of that markup. A document refers to traditional documents, vector graphics, e-commerce transactions, mathematical equations, object meta-data, server APIs, and many other kinds of structured information.

The numbers of applications currently being developed that are based on XML documents is rapidly increasing. XML is being used currently **as** a method to self-describe messages. As a result, XML and messaging are tremendously complementary to each other, especially when used to define messages between applications. The ability to self describe messages simplifies the addressing process while continuing to abstract networking details away from the programmers and developers.

b. Store and Forward Messaging: Store and Forward Messaging is used with message queuing to achieve the highest degree of reliability in a messaging system. The store feature stores any message sent to the message queue and ensures that if any subscriber is not listening and does not receive the message it is forwarded to the subscriber the next time the connection to the queue is established. This feature is relevant to C4ISR because connections may be disrupted by jamming or intentional periods of radio silence.

c. Publish & Subscribe Messaging: Publish/subscribe messaging is generally event-driven rather than demand-driven. In this addressing solution, producers (also called publishers) disseminate data to multiple consumers (also called subscribers). Publish/subscribe interactions are driven by events (usually the arrival or creation of data) in a producer. Consumers place a standing request for data by subscribing, but publishing events are independent of subscriptions. Communication is in one direction only, and is often one-to-many. Example applications include:

- Securities data feed handlers publish the latest stock prices to hundreds of traders on a trading floor simultaneously,
- Materials movement systems distribute data to various materials handlers, controllers and tracking systems on a factory floor,
- Inventory levels flow continuously to accounting, purchasing, marketing and other departments in a retail store.
- Sensors publish data about tracks identified as threats.

Messaging Standards
A number of messaging standards are presented here that were encountered during research. This list represents only a sample of all DOD and commercial messaging and interface standards.

a. Message Passing Interface(MPI): A standard for writing message passing programs that allow application programmers and software developers to write message passing programs in Fortran 7 and C that operate in a distributed environment. (MPI-2) effective April 1997.

b. Common Messaging Call API 2.0: A standard for a high level application programming interface that allows programmers to integrate messaging into existing and future applications that creates a group of messaging enabled applications. (CMC 2.0) effective June 1995.

c. MIL-STD-6040, 31 March 2000 - USMTF Message Formatting Program - The 2000 Baseline includes changes to the 1999 baseline made by Interface Change Proposals (ICPS) approved by Configuration Control Boards (CCBS) 3-98, 4-98, 1-99, and 2-99. The Revision Date for the approved ICP changes are 24SEP1998, 05NOV1998, and 25FEB1999. This baseline also includes Service Unique Messages (Army, Navy, USAF).

d. CJCSM 6120.05, 31 March 2000 - Manual For Tactical Command and Control Planning Guidance For Joint Operations Joint Interface Operational Procedures For Message Text Formats. There are no additions in the 2000 standard, only revisions. The changes approved at CCB 1-99 deleted references to the QUEWORD set formerly used in the LOCATOR MTF, updated changes to the SPRT.GEOM MTF, made changes to the purpose statements in the OPREP-3 series and NUDETSUM MTFs, and deleted numerous references to specified commands. CCB 2-99 changed the purpose statement for the ACO message. There were no ICPs approved at CCBs 2-98, 3-98, and 4-98 which contained changes to the **JIOP.**

e. JIEO Circular 9152, 31 March 2000 - Repository of USMTF Program Items for U.S. Implementation Guidance. This document is not releasable to foreign nationals or organizations.

Messaging Systems
A DOD messaging system was discovered during this research and is included here as an example of an existing system in DOD. It represents one system that could benefit from the data exchange capabilities currently realized in COTS products used in enterprise applications.

a. Defense Message System: This system encompasses a variety of products to be furnished and the services to be performed to provide conformant, Defense Message System (DMS)-compliant messaging (hereafter referred to as DMS, see [dms]). DMS users may be tactical, fixed, mobile, or transportable.

DMS products will be used to support multiple levels of classification (i.e., UNCLASSIFIED BUT SENSITIVE to TOP SECRET) and compartments. The implementation of DMS products will take advantage of the messaging user's existing hardware whenever practical, including distributed client/server processing, as long as the required messaging functionality and security are maintained.

DMS products will operate in the Transport Control Protocol and Internet Protocol (TCP/IP) environment.
A goal of DMS is to avoid a long-term dependence on a single source of supply for messaging technology, products, and services. The messaging system is envisioned to be built with open systems using a variety of commercial-off-the-shelf (COTS) products. The DMS products will provide desired improvements as technology and standards

evolve. Products should be replaced or supplemented with upgraded components, without loss of needed functionality.

The target architecture emphasizes flexibility to support the products of current and future **DOD** information technology programs, as well as technological advances that may become available in the future. DMS may be used to support additional messaging requirements, including mail-enabled applications, electronic data interchange, multi-media applications, voice, imaging formats, facsimile, and any new features and capabilities supporting electronic transmission of data and information. As new interoperability standards are adopted by the **DOD,** DMS products will evolve to support those standards.

ORBs

An object request broker (ORB) is a middleware technology that manages communication and data exchange between objects. ORBs promote interoperability of distributed object systems because they enable users to build systems by piecing together objects- from different vendors- that communicate with each other via the ORB. The implementation details of the ORB are generally not important to developers building distributed systems. The developers are only concerned with the object interface details. This form of information hiding enhances system maintainability since the object communication details are hidden from the developers and isolated in the ORB.

ORB technology promotes the goal of object communication across machine, software, and vendor boundaries. The relevant functions of an ORB technology are

- interface definition
- location and possible activation of remote objects
- communication between clients and objects

An object request broker acts as a kind of telephone exchange. It provides a directory of services and helps establish connections between clients and these services.

The ORB must support many functions in order to operate consistently and effectively, but many of these functions are hidden from the user of the ORB. It is the responsibility of the ORB to provide the illusion of locality, in other words, to make it appear as if the object is local to the client, while in reality it may reside in a different process or machine. Thus the ORB provides a framework for cross-system communication between objects. This is the first technical step toward interoperability of object systems.

The next technical step toward object system interoperability is the communication of objects across platforms. An ORB allows objects to hide their implementation details from clients. This can include programming language, operating system, host hardware, and object location. Each of these can be thought of as a "transparency," and different ORB technologies may choose to support different transparencies, thus extending the benefits of object orientation across platforms and communication channels.

There are many ways of implementing the basic ORB concept; for example, ORB functions can be compiled into clients, can be separate processes, or can be part of an operating system kernel. These basic design decisions might be fixed in a single product; or there might be a range of choices left to the ORB implementers.

There are two major ORB technologies:

- The Object Management Group's (OMG) Common Object Request Broker Architecture (CORBA) specification
- Microsoft's Component Object (COM/DCOM)

An additional, newly-emerging ORB model **is** Remote Method Invocation (RMI); this is specified as part of the Java language/virtual machine. RMI allows methods of Java objects to be executed remotely. This provides ORB-like capabilities as a native extension of Java.

A high-level comparison of ORB technologies is available **in** Table 1.

ORB	Platform Availability	Mechanism	Implementations
COM/ DCOM	PC Windows platforms, but becoming available on other platforms	APIs to proprietary system	One
CORBA	All major OS platforms	specification of distributed object technology	Many
Java/ RMI	wherever Java virtual machine (VM) executes	implementation of distributed object technology	Various

Table1 : High-level comparison of ORB technologies

As shown in Table 1, there are a number of commercial ORB products available. ORB products that are not compliant with either CORBA, DCOM or RMI also exist; however, these tend to be vendor-unique solutions that may affect system interoperability, portability, and maintainability.

CORBA

The Common Object Request Broker Architecture (CORBA) is a specification of a standard architecture for object request brokers (ORBs). It is the product of a consortium called the Object Management Group (OMG) that includes over 800 companies, representing the entire spectrum of the computer industry. What makes CORBA so important is that it defines a middleware model that has the potential of subsuming every other existing form of client/server middleware. CORBA standard architecture allows vendors to develop ORB products that support application ***portability*** and ***interoperability*** across different programming languages, hardware platforms, operating systems, and ORB implementations.

CORBA relies on a protocol called the Internet Inter-ORB Protocol (IIOP) for remote objects. Everything in the CORBA architecture depends on an Object Request Broker (ORB). The ORB acts as a central Object Bus over which each CORBA object interacts transparently with other CORBA objects located either locally or remotely. Each CORBA server object has an interface and exposes a set of methods. To request a service, a CORBA client acquires an object reference to a CORBA server object. The client can now make method calls on the object reference as if the CORBA server object resided in the client's address space. The ORB is responsible for finding a CORBA object's implementation, preparing it to receive requests, communicate requests to it and carry the reply back to the client. A CORBA object interacts with the ORB either through the ORB interface or through an Object Adapter - either a Basic Object Adapter (BOA) or a Portable Object Adapter (POA). Since CORBA is just a specification, it can be used on diverse operating system platforms from mainframes to UNIX boxes to Windows machines to handheld devices as long as there is an ORB implementation for that platform. Major ORB vendors like Inprise have CORBA ORB implementations through their VisiBroker product for Windows, UNIX and mainframe platforms and Iona through their Orbix product.

Beside the basic ORB, CORBA specifics a set of system level services packaged with the IDL-specified interfaces. The services complement the functionality of ORB and help to manage and maintain objects through their life cycles.

The 16 object services include:

- Collection
- Concurrency control
- Event notification
- Externalization
- Licensing
- Life cycle
- Naming
- Persistence
- Properties
- Query

- Relationships
- Security
- Startup
- Time
- Trader
- Transactions

DCOM

DCOM, which is often called *'COM on the wire'*, supports remote objects by running on a protocol called the Object Remote Procedure Call (ORPC). This ORPC layer is built on top of DCE's RPC and interacts with COM's run-time services. A DCOM server is a body of code that is capable of serving up objects of a particular type at runtime. Each DCOM server object can support *multiple interfaces* each representing a different behavior of the object. A DCOM client calls the exposed methods of a DCOM server by acquiring a pointer to one of the server object's interfaces. The client object then starts calling the server object's exposed methods through the acquired interface pointer as if the server object resided in the client's address space. As specified by COM, a server object's memory layout conforms to the C++ vtable layout. Since the COM specification is at the binary level it allows DCOM server components to be written in diverse programming languages like C++, Java, Object Pascal (Delphi), Visual Basic and even COBOL. As long as a platform supports COM services, DCOM can be used on that platform. DCOM is now heavily used on the Windows platform. Companies like Software **AG** provide COM service implementations through their EntireX product for UNIX, Linux and mainframe platforms; Digital for the Open VMS platform and Microsoft for Windows and Solaris platforms.

Microsoft introduced COM+ v1.0 recently as an enhancement with a set of services like CORBA. These services and features include:

- Role-base Security services
- Transaction services
- Administration services
- Load balancing
- Proprietary Event services
- Event services
- Queue Components
- In Memory database

Java/RMI

Java/RMI relies on a protocol called the Java Remote Method Protocol (JRMP). Java relies heavily on Java Object Serialization, which allows objects to be marshaled (or transmitted) as a stream. Since Java Object Serialization is specific to Java, both the Java/RMI server object and the client object have to be written in Java. Each Java/RMI Server object defines an interface which can be used to access the server object outside of the current Java Virtual Machine(JVM) and on another machine's JVM. The interface exposes a set of methods which are indicative of the services offered by the server object. For a client to locate a server object for the first time, RMI depends on a naming mechanism called an RMIRegistry that runs on the Server machine and holds information about available Server Objects. A Java/RMI client acquires an object reference to a Java/RMI server object by doing a lookup for a Server Object reference and invokes methods on the Server Object as if the Java/RMI server object resided in the client's address space. Java/RMI server objects are named using URLs and for a client to acquire a server object reference, it should specify the **URL** of the server object as you would with the URL to a HTML page. Since Java/RMI relies on Java, it can be used on diverse operating system platforms from mainframes to UNIX boxes to Windows machines to handheld devices as long as there is a Java Virtual Machine (JVM) implementation for that platform. In addition to Javasoft and Microsoft, a lot of other companies have announced Java Virtual Machine ports.

However, since Java/RMI requires both the client and server to be written in Java, this technology will not be further discussed as it will not work in an heterogeneous programming language environment found in C4ISR scenarios. Note however that a force fit could be established by using Java wrappers.

JINI

Introduction

Sun claims that Jini will make networks as simple as turning on a light bulb. The claim is based upon the vision rather than current reality. Jini is a new technology that promises to simplify interactions across a network. However, Jini is still in its infancy as a technology and remains relatively untested and under-explored. This document provides an overview of the Jini technology and discusses some issues concerning interoperability.

What is Jini?

Sun describes Jini as an extension of the Java application environment from a single virtual machine to a network of machines. [Sun99a] Jini has also been described as a network operating system. [Sun99b] The overall goal of the Jini system is to make services available on a network, so that the services may be used by any system connected to the network. The network will be dynamic, in that services can be added and deleted and service users can connect, use any available service, and disconnect. For example, a digital camera could connect to the network, find a printer service, print a picture, and then be disconnected from the network. The goals are therefore similar to Lucent Technologies Inferno (see Section 4, Messaging)

Figure 1 [JavaWorld99] represents the Jini architecture. The PDA and the Cell Phone represent hardware devices that offer services to the network via interfaces that are written in the Java language. The services are

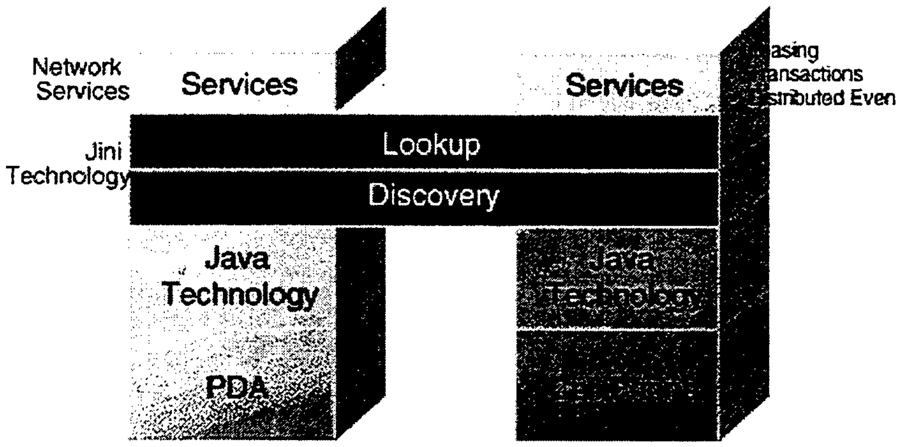


Figure 1: Jini Architecture

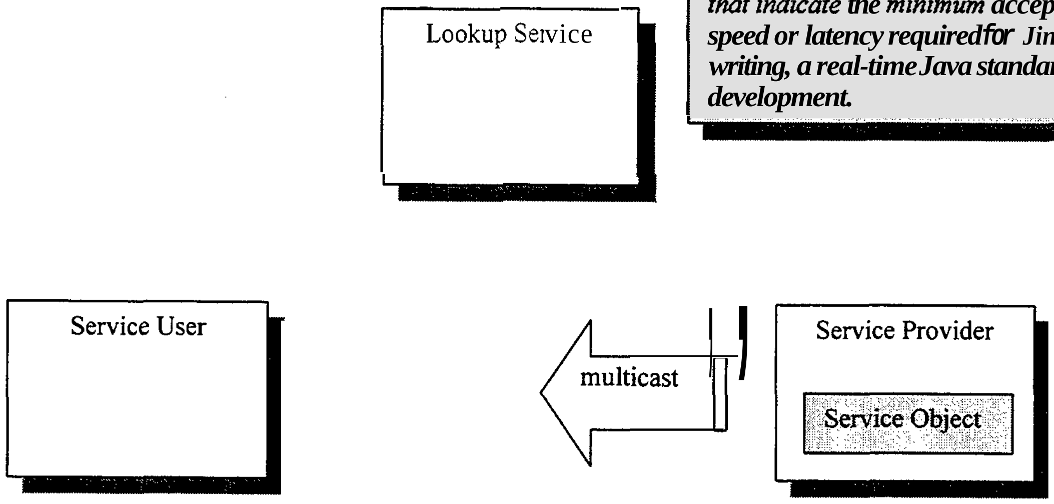
objects that are passed from a service provider to a user. Jini cannot be described in terms of a client-server architecture. Jini's methodology for distributed computing is closer to CORBA, where objects are registered with a lookup service and users find objects via the lookup service. The object containing the methods of the service is then passed to the requestor, and the requestor invokes methods on that service object. Jini, however, is neither designed nor intended to compete with CORBA. Jini is designed and intended for simplifying networks via the Java application environment; whereas, CORBA is designed and intended for distributing objects between different programming languages. Jini can theoretically use CORBA to extend services to systems written in programming languages other than Java. The requesting system, however, would still need access to a JVM even if it used CORBA. CORBA would provide the interface between the Jini system and the non-Java system. Jini can also support any programming language that has a compiler that produces compliant bytecodes for the Java Virtual Machine, such as Ada95.

How Does Jini Work?

A Jini system consists of Service Providers, Services, and Members. A member is any system that is connected to the network and can either use Jini services and/or provide Jini services. The concept of a service is the most important concept in the Jini architecture. [Sun99a] Members of a Jini system federate together in order to share services. The federation is the Jini system, and its structure and the services it offers are dynamic and can be created and tom down at run time. Figures 2, 3, and 4 describe how a Jini system is formed:

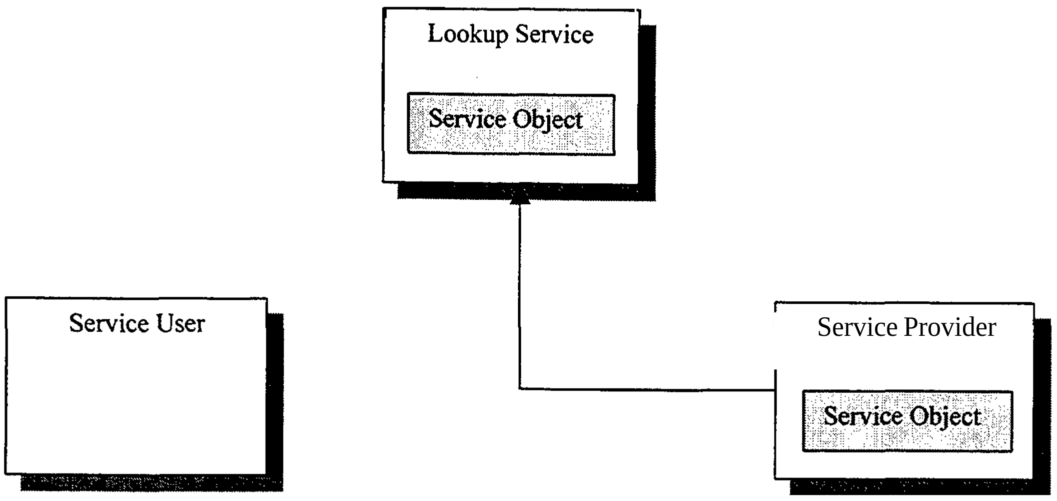
Assumption: All systems are connected to a **multicast network with** reasonable speed and latency..

**Unable to find test documentation or studies that indicate the minimum acceptable network speed or latency required for Jini. At the time of writing, a real-time Java standard is under development.*



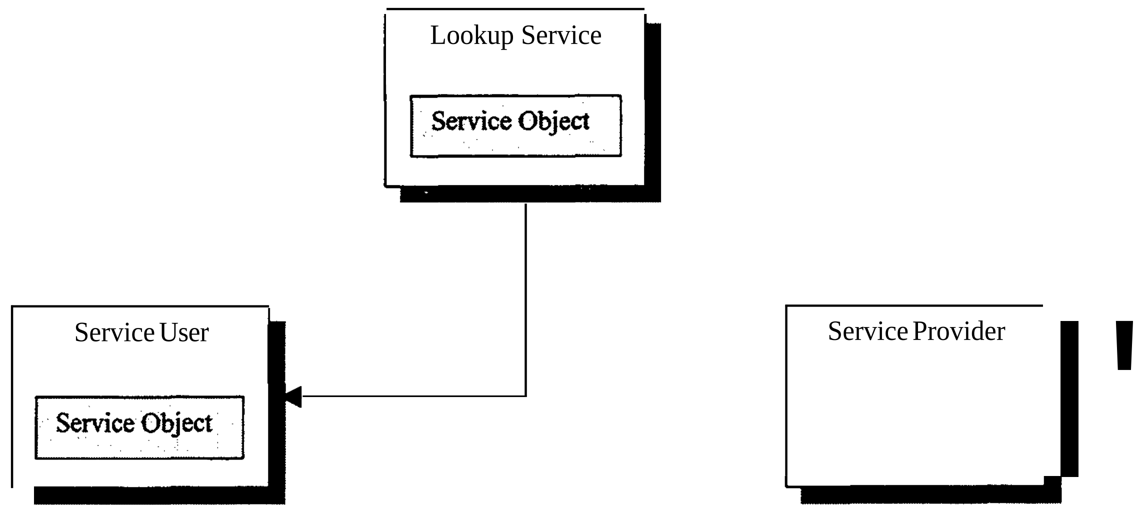
A service provider seeks a lookup service by sending out a multicast request (discovery).

Figure 2: Discovery



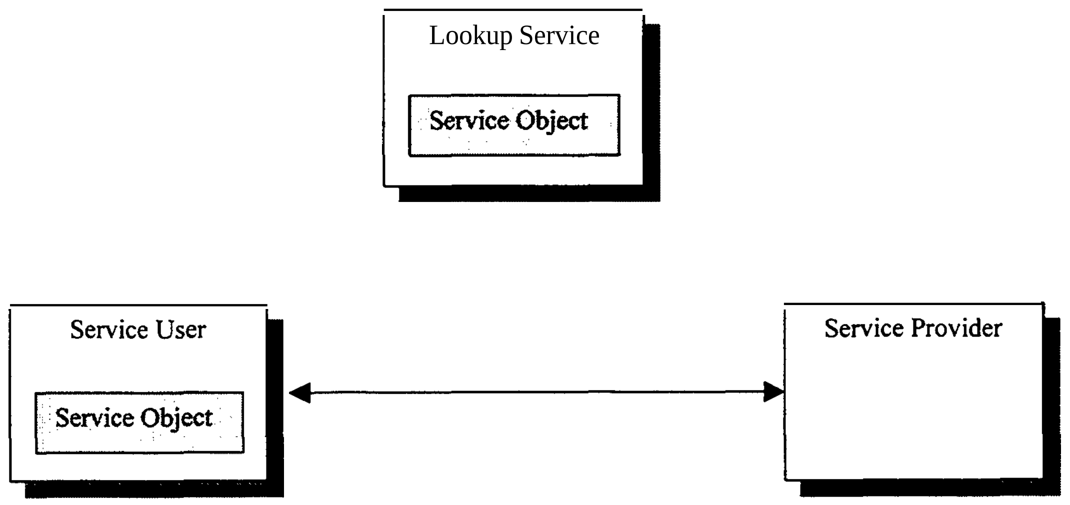
Once it finds **a** lookup service on the network, it registers a Service Object with the lookup service (join).

Figure 3: Join



User requests a service and a copy of the service object is passed to the user and used by the user to talk to the service (lookup).

Figure 4: Lookup



The user now interacts directly with the Service Provider via the Service Object (service invocation).

Figure 5: Service Invocation

5. Conclusion

The technologies surveyed are difficult to compare because they address different issues and different levels of the problem. The alternatives surveyed are of three fundamentally different types: building blocks, architectures, and packaging types for integration services. The second two are related to varieties of Object Request Brokers (ORBs).

The building blocks are also hard to compare meaningfully, because they address three essentially different issues.

The purpose of messaging is to communicate interactions between physically separate locations. Since it is the only building block among those surveyed that directly addresses this issue, all forms of physically distributed system interoperation must rely on some form of messaging, either directly or indirectly.

The purpose of wrappers is to provide a uniform interface to a set of legacy services that have a variety of legacy realizations with incompatible native interfaces. Wrappers are the only approach of those surveyed that directly addresses this issue. Consequently we expect all forms of interoperation between disparate legacy systems to directly or indirectly use wrappers.

The purpose of translators, mediators, and replicators is to enable interoperation between different forms of data that have the same meaning. This is also an essential service, and we expect all forms of system interoperation to directly or indirectly use at least one of these three building blocks when several incompatible data representations are involved. Since the strengths and weaknesses of the three approaches are to some extent complementary, simultaneous use of several of these approaches is quite reasonable.

The purpose of an ORB is to encapsulate the provided integration/interoperation services to make them independent of implementation details such as computing hardware, operating system, programming language, and data representation. ORBs must rely on the surveyed building blocks to realize the services they provide. The difference in using an ORE3 and using the building blocks directly is that the application programs (such as C4ISR systems) interact with standardized interfaces rather than directly with the building blocks underneath those interfaces. This decouples the applications from the mechanisms used to realize the needed services, and makes it much easier to take advantage of new approaches to realize the building blocks as technology improves. In such a case, the implementation of the ORB would have to change to exploit new or improved building blocks, but the application programs would not have to change. Since is typical ORB is much smaller and simpler than a typical C4ISR system this has its advantages.

The idea of an ORB is a generic one that admits many different architectural styles and realizations. CORBA, COM/DCOM, and JINI are examples of some variations that have appeared in the commercial marketplace, and this is by no means a complete list. Each of these can have a variety of implementations and can be supported by multiple vendors. The first two of these already have a variety of products in the marketplace, while JINI is a newer JAVA-based approach that has not yet reached that stage, but is in a state of rapid development.

ORBs have advantages with respect to maintainability in the long run, rapid reconfiguration, reuse, and incremental migration to new hardware and software products. The use of ORBs can also simplify the structure of applications software, which can reduce the cost of building new systems for Joint operation. Possible disadvantages of ORBs include performance considerations and maturity of the technology. The data in the body of the report shows that at least for static approaches to ORBs, relatively efficient implementations are available, at least with respect to average-case behavior. Issues of concern for C4ISR systems, such as guarantees of response within strict real-time deadlines, are just recently being addressed by software research and formulation of new standards such as Real-Time Java. This research is very active, initial results look promising, and the approach is expected to become quite attractive in the longer term, estimated as a five to ten year time frame.

Another advantage of ORBs is that the associated standards, architectures, and tools provide the potential to automatically generate interoperability code. For example, some IDL tools associated with CORBA can already do this for the declarations that realize a standard set of services in the context of a particular programming language.

In the context of C++ this would be a header file, in the context of JAVA this would be an interface declaration, and in the context of Ada this would be the specification part of a package or task. Some software engineering research labs have advanced toward the capability to also automatically generate the implementation parts of the methods that realize such a standard set of services in terms of the services provided by a legacy system. This capability is not yet available in commercial tools, but it is only a matter of time before it will be.

In the context of fielding systems in the one to three year time span, ORBS may not be able to reach their full potential for C4ISR applications, and may not be fully accessible in practice. Our concerns are that it takes time for the commercial sector to develop quality products that incorporate new approaches, and that it takes time to train contractor personnel to take advantages of new approaches. These steps must be completed before new technologies can be applied to development of fielded systems.

Nevertheless, ORBs are the highest level approach available and are attractive even in the short term for the contexts in which they can realistically be used. The main commercial approaches to ORBs, such as CORBA and COM/DCOM, are relatively mature, there are existing populations of engineers who already know how to use them, and fairly extensive choices of tools and implementations are commercially available, as detailed in the rest of this report. In contexts where guarantees of service within strict time limits are not required this approach should be usable now.

There is an additional risk factor related to interoperability of C4ISR systems that was not included in the tasking from JBC. We describe it briefly here because it can potentially block achievement of the desired interoperability of C4ISR systems. It is a common belief that interoperability problems are caused by incompatible interface and data formats, and can be fixed “easily” using interface converters and data formatters. However, the most difficult challenges in fixing interoperability problems are incompatible data interpretations, inconsistent assumptions, requirement extensions triggered by global integration issues, and timely data communication between components. The technologies surveyed here just scratch the surface of the real problem, and do not in themselves provide a full solution to the real problem, which is system integration. An integrated system is greater than the sum of its legacy parts: it needs to provide new services and additional value added.

For example, many stovepipe systems drive a given application from data derived from a single sensor. In the original context, it is reasonable to assume that distinct observations reported by the sensor represent different targets at different locations. If we now require many such systems to interoperate, and the sensors have different accuracies, it becomes possible to observe multiple images of the same target. The logic encoded in the legacy software will report a flock of ghost images for each target unless new requirements for data fusion and the corresponding software implementations are added to provide the desired integrated picture. This is just a single example that illustrates the subtler interoperability problems that arise due to inconsistent assumptions and requirement incompatibilities. There are many other similar issues related to C4ISR interoperability.

Different technologies, including software prototyping, requirements assessment, and architectures that support software adaptation, are needed to address these issues. We know of no currently available commercial products that address these issues for the kind of distributed real-time software typical of C4ISR systems. There is currently ongoing software research seeking to mitigate these risk areas. However, it will take longer than three years for the results of these research efforts to develop to the point where they can be used in fielded systems. In the short term, these issues have to be mitigated by informal means and creative engineering. Success depends on giving the issues high priority and high visibility to Joint sponsors, contractors and contract monitors.

6. References

[ATMN99] – News Release from ATM News Digest, Castle Networks Debuts Voice and Data Mediation Platform, 1999.

[Brow97] – Jim Browne, Compositional Development of Performance Models, 1997.

[Brow99] - Tony M Brown, MQSeries Customers Steal The Show, 1998

[Buretta97] Data Replication, Marie Buretta, John Wiley & Sons, Inc., 1997.

[Cons98] – White Paper, Constellar Corporation, Enterprise Application Integration, 1998.

[Damr93] – Marc Demarest, Building the Data Amrt, 1993.

[Datamation] Database Replication Explained, http://www.datamation.com/PlugIn/issues/1996/sept/09db-rep.htm

[Dawl98a] - Steve Dawson, Subcontract with **SRI** International, Trusted Interoperation of Healthcare Information (TIHI), 1998.

[Dawl98b] - Steve Dawson, Subcontract with **SRI** International, Secure Access Wrapper: Securing Databases by Access Mediation (SAW), 1998.

[Dawa96] – John Young, exploring New Age Mainframes, 1996.

[dms] – Defense Messaging System http://cms1.ssg.gunter.af.mil/DMSCONTRACT/Sect-c.htm

[Grif98] – Steve Griffin, Program Manager, Image Security Project, Trusted Image Generation (TID[D), 1998.

[Her295] – V.S.Subrahmanian, Sibel Adali, Anne Brink, James J. Lu, Adil Rajput, Timothy J. Rogers, Robert Ross, Charles Ward, A Heterogeneous Reasoning and Mediator System, 1995.

[Herm97] – HERMES Project Overview, **http://www.cs.umd.edu/proiects/hermes/overview/overview.html**

[Higgs] Higgs, M., and Cottman, B., Solving the Data Interoperability Problem Using **A** Universal Data Access Broker.

[HnFe99] – Lawrence Henschen, Tania Neild, Chris Fernandes, An Object-Oriented Graph Traversal Algorithm for Data Mediation, 1999.

[IBM98] **http://www.software.hostin4.software.ibm.com/data/dpropr/98factsheetl**

[Imno95] – W. H. Imnon, Tech Topic: What is a Data Warehouse?, 1995.

[ISM998] – Gio Wiederhold , Course Summary, Mediator in the Architecture of Future Information Systems 1998.

[JavaWorld99] "Why Jini Matters" by William Blundon, URL: http://www.javaworld.com/javaworld/jw-10-1998/jw-10-blundon.html, retrieved 9/24/99

[Kieb96] - Richard Kieburtz, Laura Mckinney, Jeffrey Bell, James Hook, Alex Kotov, Jeffrey Lewis, Dino Oliva, Tim Sheard, Ira Smith, Lisa Walton, A Software Engineering Experiment in Software Component Generation, Proceedings of 18th International Conference on Software Engineering, Berlin, IEEE Computer Society Press, March, 1996.

[Lakeviewa] Selecting a Data Replicator, **http://www.lakeviewtech.com/news/whites/wpsdr.pdf**

[Lakeviewb] The Economics of Data Replication, **http://www.lakeviewtech.com/news/whites/wpedr.pdf**

[Lint99] – David Linthicum, How to Free Your Information, Data Level Integration Part **I,** 1999.

[LMIT96] – INEEL Mediator Project Summary, 1996.

[Malu97] - David A. Maluf, Gio Wiederhold, Ted Linden, and Priya Panchapagesan: http://www-db.stanford.edu/~maluf/postscript/crosstalk.ps

[McSc97] – Maura McGrath, Christian Schock, The transformation Hub Architecture, White Paper for Constellar Corp. 1997.

[Medi99] – List of Mediator companies compiled by Gio Wiederhold. 1999.

[Meeson 1997] Reginald N. Meeson, Analysis of Secure Wrapping Technologies (Alexandria, VA: Institute for Defense Analysis).

[Merl96] – INEEL Product Information, Merlin: Transforming Legacy Data into Meaningful Information, 1996.

[Nels99] Rolin Nelson, **AISI White Paper Series: #2 Topics: Exploring High Availability Issues with BEA Tuxedo and Third Party High Availability Software** January 20, 1999, Version 1.0

[Noel96] – Rick Noel, Data Warehouses, 1996.

[PEDe97] – Bhujanga Panchapagesan, Joshua Hui, Gio Wiederhold, Stephan Erickson, Lynn Dean, The INEEL Data Integration Mediation System, May 1997.

[Renner] Renner, S., Rosenthal, A., Scarano, J., Data Interoperability: Standardization or Mediation.

[RRSc96] – Scott Renner, Arnold Rosenthal, James Scarano, Data Interoperability: Standardization or Mediation, 1996.

[SGSh98] – Kevin Sullivan, Steve Geist, Paul Shaw, Mediators in Infrastructure Survivability Enhancement, 1998.

[SHAD99] – Shared Data Environment (SHADE) Requirements Document, 1999.

[Stal97] - William Stallings, **Operating Systems: internals and design principles,** 1997

[Sullivan, 1998] Kevin J. Sullivan, Steve Geist, Paul Shaw, Mediators in Infrastructure Survivability Enhancement (Orlando, FL: ISAW3).

[Sun99a] "Jini Technology Architectural Overview", URL: **http://wwwswest2.sun.com/jini/whitepapers/architecture.html**, retrieved 9/28/99

[Sun99b] "Jini Connection Technology Fact Sheet", URL: **http://www.sun.com/jini/facsheet/**, retrieved 9/24/99

[Sybase] Sybase Replication Server, http://www.sybase.com/products/datamove/repsrvr.htm

[TIBC99] - From a 1999 White Paper, TIBCO Software Inc. 3165 Porter Drive, **Palo Alto, CA 94304 USA**

[Unis99] – Unisphere Solutions Product Sheet, SMX-2100 Management Mediation, 1999.

[Wibr98] – Gio Wiederhold, Lecture slides "Evolution of Mediation", 1998.

[Wied98] – Gio Wiederhold, Value-added Middleware: Mediators, March 1998.

7. Appendix A: Suppliers of Mediation Technology

Name: BEA systems
Town: Sunnyvale CA
Specialty: large-scale infrastructure (TUXEDO platform; MessageQ messaging)
Contact: Bill Coleman coleman@,BEAsvs.com

Name: Constellar
Town: Redwood Shores CA
Web: http://www.Constellar.com/
Specialty: rule-based warehouse population
Contact: info@constellar.com
....
Name: Epistemics
Town: Palo Alto CA
Web: http://www.epistemics.com/
Specialty: Mediator software, resource scheduling
Contact: Arthur Keller info@epistemics.com

Name: FastXchange (spinoff from ISI)
Town: Marina del Rey, CA 90292
Web: http://www.FASTXchange.com/
Specialty: procurement
Contact: Anna-Lena Neches info@fastxchange.com

Name: Genelogic
Town: Berkely CA
Web: **http://www.genelogic.com/bioinform.htm**
Specialty: genomics information in object form
Contact: Victor Markowitz victor@,genelogic.com

Name: Global InfoTek, Inc.
Town: Vienna, VA. 22 180
Web: **http://www.elobalinfotek.com**
Specialty: mediation and Java-based ad hoc query and visualization tools.
Contact: Ray Emami **oemami@pluto.dobalinfotek.com**
....
Name: IBrain Software
Town: Palo Alto
Web: http://www.ibrain.com
Specialty: integration and ranking of financial information, mediator software
Current Specialty: Financial services.
Contact: Vishal Sikka vishal@ibrain.com
....
Name: I-Kinetics Inc,
Town: Cambridge MA
Web: http://www.i-kinetics.com/
Specialty: scalable access methods
Contact: Bruce Cottman bruce.cottman@i-kinetics.com

Name: ISI
Town: Marina Del Ray, CA 90292
Web: http://www.isi.edu
Specialty: research, system engineering
Contact: Ygal Arens arensOisi.edu

Name: ISX
Town: Westlake Village CA
Web: http://www.isx.com
Specialty: intelligence systems, planning and logistics
Contact: Nancy Lehrer Nlehrer@,isx.com

Name: Junglee
Town: Sunnyvale, CA
Web: http://www.iunelee.com/
Specialty: web source integration and filtering for shopping, job placement, etc., used by Yahoo a.o.
Contact: Anand Rajaraman anand@,iunelee.com
....
Name: K2 Informatics, Inc.
Town: Bryn Mawr, PA **19010**
Specialty: genomic information
Contact: Karen J. Giroux 71072.234@compuserve.com

Name: Lockheed-Martin Idaho Technologies
Town: Idaho Falls, ID
Web: http://id.inel.gov/idim; **http://id.inel.eov/merlin**
Specialty: environmental, chemical data
Contact: Lynn Dean lad@inel.gov

Name: Lockheed Martin C2 Integration Systems
Town: Frazer, PA **19355-180**
Web: **http://www.paoli.atm.Imco.com/kqm**
Specialty: systems engineering, communication (KQML) for government
Contact: Robin McEntire **Robin.A.McEntire@,lmco.com**
....
Name: MCC
Town: Austin TX
Web: **http://www.mcc.com/proiects/infosleuth2/**
Specialty: mediating agent research for consortium members
Contact: Marek Rusinkiewicz Marek@,mcc.com

Name: Persistence Software
Town: San Mateo CA
Web: http://www.persistence.com
Specialty: relation to object transformation
Contact: keene@,persistence.com
—— ----------------
Name: Socratix
Town: Palo Alto CA
Web: www.socratix.com
Specialty: clinical and biological information
Contact: Russ Altman altman@smi.stanford.edu

Name: Tesserae Information Systems
Town: San Jose, CA **951 13**
Web: http://tesserae.com/
Specialty: electronic commerce
Contact: Narinder Singh sinoh@,tesserae.com

8. Appendix B: ORB Vendors

- Expersoft (PowerBroker)
- Siemens Nixdorf (SORBET)
- IONA Technolooies (Orbix)

- ICOG Broker
- Inprise (VisiBroker)
- BEA (ObjectBroker)
- PeerLogic (DAIS)
- Hewlett-Packard (ORB Plus)
- IBM (SOMobjects)
- Tandem (Non-Stop DOM)
- Nortel (RCP-ORB)
- Object Oriented Technologies Ltd (DOME) .
- Promia Inc. (SmalltalkBroker)
- PrismTech (OpenBase)
- TRW (UNAS)
- ParcPlace (Distributed Smalltalk)
- TIBCO (ObjectBus)
- Suite (SuiteValet)
- Fujitsu (Object Director)
- BBN (Corbus)
- ANSA (DIMMA)
- SuperNova
- Paragon Software (OAK)
- OutBack Resource Group (jEvents, jNames)
- Regis ORB
- ObjectSpace (Voyager)
- Objective Interface Systems (ORBexpress) also marketed by ObjecTime
- Bionic Buffalo
- 1-Kinetics
- RogueWave (Nouveau)
- ObjectScape (JBroker)
- Secant (Persistent Object Manager)

- Obiect Oriented Concepts (ORBacus, formerly OmniBroker)
- OMEX meta obiect facilities
- North Street Systems (Obiect Switch)
- Harlequin (Common Lisp ORB and Dylan ORB)
- Lockheed Martin (HARDPack)
- Washington University's TAO ORB supported by OCI
- RHAD Labs (ORBit)
- Electra and iBus
- Univ. of Colorado (Q/CORBA)
- JacORB
- Jorba **(GNU CORBA)** by Roland Turner
- AT&T (omniORB)
- the Flick IDL compiler
- COPE for Perl
- Java RMI over CORBA IIOP
- ROBIN (Rpc and Obiect Broker INterface)
- Xerox Inter Language Unification and ILU for Java
  - Scott W. Hassan at Stanford Digital Library Testbed Development
  - Common Obiect Services for ILU
  - Java binding for ILU
  - ILU Requester (Web server-side interface to ILU)
- Mico
- JavaORB
- Arachne
- Fnorb
- Data Access Technologies
- Mark Spruiell's JTrader
- dynaOrb
- Artefact CSCW framework

- Enabled Systems
- IOR decoder
- Jonathan ORB
- ISP C++ ORB

References:

1. David L.Levine, Sergio Flores-Gaitan, and Douglas C.Schmidt, An Empirical Evaluation of OS Endsystem Support for Real-time CORBA Object Request Brokers, MMCNOO, Jan 2000.

2. MLC Systeme GmbH Germany, and Distributed Systems Research Group Charles University, Czech Republic, CORBA Comparison Project Final Report

3. P. Emerald Chung, Yennun Huang, Shalini Yajnik, Deron Liang, Joanne C. Shih Chung-Yih Wang, and Yi-Min Wang, DCOM and CORBA Side by Side, Step by Step, and Layer by Layer, **http://www.cs.wustl.edu/-schmidt/submit/Paper.html**

4. Robert Orfali, Dan Harkey, Client/Server Programming with JAVA and CORBA, second edition, Wiley.

5. Richard Grimes, Professional DCOM Programming, Wrox.

6. Microsoft, *The Component Object Model Specification,* **http://microsoft.com/com/tech/com.asp**

7. Microsoft, Distributed Component Object Model Specification, http://m icrosoft.com/com/tech/dcom.asp

8. Microsoft, *COM+*, **http://microsoft.com/corn/tech/complus.asp**

9. OMG, *The Common Object Request Broker: Architecture and Specification*, Revision *2.0,* July 1995, http://www.omg.org/corba/c2indx.html

10. OMG, *CORBA: What Coming in Revision **3.0**, http://www.omg.org/news/pr98/compnent.html*

9. Appendix C: Experience with Solutions to Interoperability Problems in DOD Projects

Aggregate Level Simulation Protocol (ALSP) in Joint Training Confederation (JTC)

Problem: Each military service independently developed a war gaming system. How can the systems interoperate such that entities in one system are available to other systems?

Solution: A common simulation protocol consisting of text messages representing events was developed. Types of entities in one system were mapped to equivalent types of entities in other systems. The status of all common entities were updated every **60** seconds of simulation time. A master control program would grant a time advance after all actors had sent out updates. Each simulator sent updates to a communications daemon that buffered and logged all incoming and outgoing messages. All unrecognized messages were ignored.

This solution required developers to modify each actor to consume and produce ALSP messages. Special considerations were needed for each type of "ghosted" entity. For example, battle damage was assigned by the simulator owning the target.

This solution uses the following interoperability categories: messaging, wrappers and data replication. Messaging'is implemented by the ALSP message set. Wrappers are used to map foreign entities into entities understandable to the target system. Data replication is used to "ghost" entities on multiple systems.

The ALSP solution to the war gaming interoperability problem can be evaluated in the following areas:

Security - The confederation relies on physical security. Each member of the confederation enforces its own authentication policy.

Availability - This is essential to the success of an exercise. All members need to be operational, including computing and communications, at least ***95%*** of the time during an exercise. Recovery procedures are also conducted during testing.

Reliability - Software changes are closely controlled. Weeks of separate and joint testing are performed before each exercise. Initially, testing resolves around function point testing and load testing. Just before the exercise a two day "miniex" is conducted. During the miniex, the scenario is loaded, operators are trained, and representative missions are conducted.

Performance - Performance constraints must be met annually before JTC membership is allowed. A system checkpoint must be made within five minutes. A restart from any checkpoint may not take more than one hour. Each simulator must be able to run slightly faster than real time to make up for the mandatory checkpoint each hour.

RESA models Navy assets in the JTC. RESA has a 2,000 entity limit, where an entity is an individual ship, aircraft, missile, torpedo, etc. One factor constraining the entity limit is the N-squared detection problem. The number of active sensors and the proximity of entities must be closely monitored to sustain real-time performance.

Extensibility - Each year service interoperability is improved. Fielded C41 systems are playing a larger role in the exercise each year. RESA has a one-way interface with JMCIS using OTH Gold or LINK1 1 message sets. There are plans for RESA to directly accept messages from CTAPS.

Maintainability - Troubleshooting problems can be labor intensive. The most powerful tool to help resolve problems is the ALSP message logger. Since ALSP messages are the only means of interaction among simulators, the logs contain all interactions among all members of the confederation. Another powerful tool in RESA displays the current state of all objects within the RESA simulation.

Cost - The cost of conducting an exercise is extremely high. A large number of personnel in various roles are needed to conduct **an** exercise. Preparation for an exercise takes months of planning and coordination among the services. Reducing cost is a major goal of the next generation simulation system, JSIMS. The entities for an exercise needs to be assembled within **72** hours. The number of support personnel needs to be reduced to one-third the current requirement.

Deployment Speed - The first version of ALSP was deployed very quickly. Only basic capability existed, but a growth plan was also in place. Each year since 1993 ALSP has grown incrementally. New simulators have been added, ALSP messages have been expanded, and C41 equipment availability to the training audience has increased.

The recognition and avoidance **of** difficult tasks led to quick deployment. Enforcement of a common environment model was avoided. ***So*** was enforcement of a common entity model. Features not common to all simulators were avoided. For example, even though RESA can model communications delays, this feature was not used in the JTC.

High Level Architecture (HLA)

Problem: Hundreds of simulators have been developed independently within DOD. There has been duplication of effort by repeated implementation of services common to most simulators. Most simulators do not inter-operate.

Solution: Develop a blueprint architecture that facilitates interoperability and reuse. Include an interface for services common to most simulators,

HLA is intended to be the virtual glue common to all simulators. Run-time infrastructure (RTI) is a realization of all or part of the HLA architecture. The HLA provides Object Management, Time Management, Federation Management, Data Distribution Management, Declaration Management, and Ownership Management.

Federation Management is responsible for starting, pausing, resuming, saving, and restoring a exercise, joining and resigning federates.

Declaration Management is responsible for facilities to publish and subscribe to object classes and attributes.

Data Distribution Management is responsible for creating update and subscription regions. Objects are associated with regions.

Time Management is responsible for ordering events in time sequence, granting time advance requests, returning current time, and setting mode of operation.

Ownership Management provides facilities for federates to transfer ownership of objects.

Object Management is responsible for creation, modification and deletion of objects. Provides reflection of current state of objects to interested parties. Receives interactions addressed to objects.

Security - No specific security features are mentioned in HLA.

Availability - There is no mention of redundancy within HLA

Reliability - In distributed simulations, there is usually a trade-off between reliability and performance. Performance can be improved if all messages do not need to be acknowledged. Throughput is improved if all messages do not need to arrive in time sequence. This performance increase carries a decrease in reliability. Since messages are not guaranteed to arrive or may not arrive in the same order, a replayed scenario may not produce the same results. This may not be a problem in a simulation where outcomes are determined by random draws.

HLA allows the coordinator to select the level of reliability required. During testing, replays of scenarios should produce identical results.

Performance - HLA implementations have suffered performance problems. STOW planned to simulate up to 50,000 entities but actually simulated less than 10,000 entitles. Funding also limited performance by keeping the number of workstations purchased or leased to approximately **3** 10.

The module to extend JTC/ALSP to become HLA compliant failed to meet load tests.

JSIMS decided to treat all service components as a single federate due to prediction of performance problems.

Extensibility - Extensibility is the bedrock of HLA. By providing a standard for a large class of simulators, a simulator can become a component of a larger simulation. Each simulator must publish its public entities and interactions in the Simulation Object Model (SOM).

Maintainability - All objects and interactions are defined using FOM and SOM definitions. The public faces of all objects are well known and documented. This hiding of implementation details behind a public interface promotes maintainability.

Cost - **A** cost/benefit analysis of HLA is very difficult. Ideally, quality standards reap high benefits. Unfortunately the set of all DoD simulations is far more complex than a wall plug.

Speed to Deployment - Eagle, a legacy simulator used for analysis purposes, was transformed into a HLA and ALSP distributed architecture in 16 months. Eagle was then able to federate with MCSP, a C41 system, **NSS,** a Navy simulator, and NASM an Air Force simulator.

Lightweight Extensible Information Framework (LEIF)

LEIF has evolved from a Java applet supporting DARPA/DISA Enhanced Common Operating Picture (ECOP) to a formal object oriented design with abstractions for data producers, consumers and viewers that is targeted for the C2 IT environment. LEIF is written in Java and takes advantage of commercial standards and technology.

Security - LEIF is leveraging off industry standards including Lightweight Directory Access Protocol (LDAP), Java 1.2 security policies and Java Naming and Directory Interface (JNDI). LDAP is used to access security and user role information. Windows NT Active Directory will support LDAP. Java 1.2 security policy provides for control of access to programs and data, digital signatures and encryption. JDNI is a superset of common naming and directory services including LDAP, CORBA naming and RMI registry.

Availability - No information available

Reliability - No information available

Performance - No objective information available. A goal of LEIF **4.0** is to improve performance.

Extensibility - LEIF uses extensible Markup Language (XML) to standardize the storage and transmission of data formats XML is a W3C standard. LEIF can be extended with Java Plug-ins.

Maintainability - Applications developed using LEIF 3.0 will not run under 4.0. Usually, only minor modifications are necessary to upgrade an application. LEIF 4.0 applications have a smaller footprint due to improved packaging.

Cost - No information available

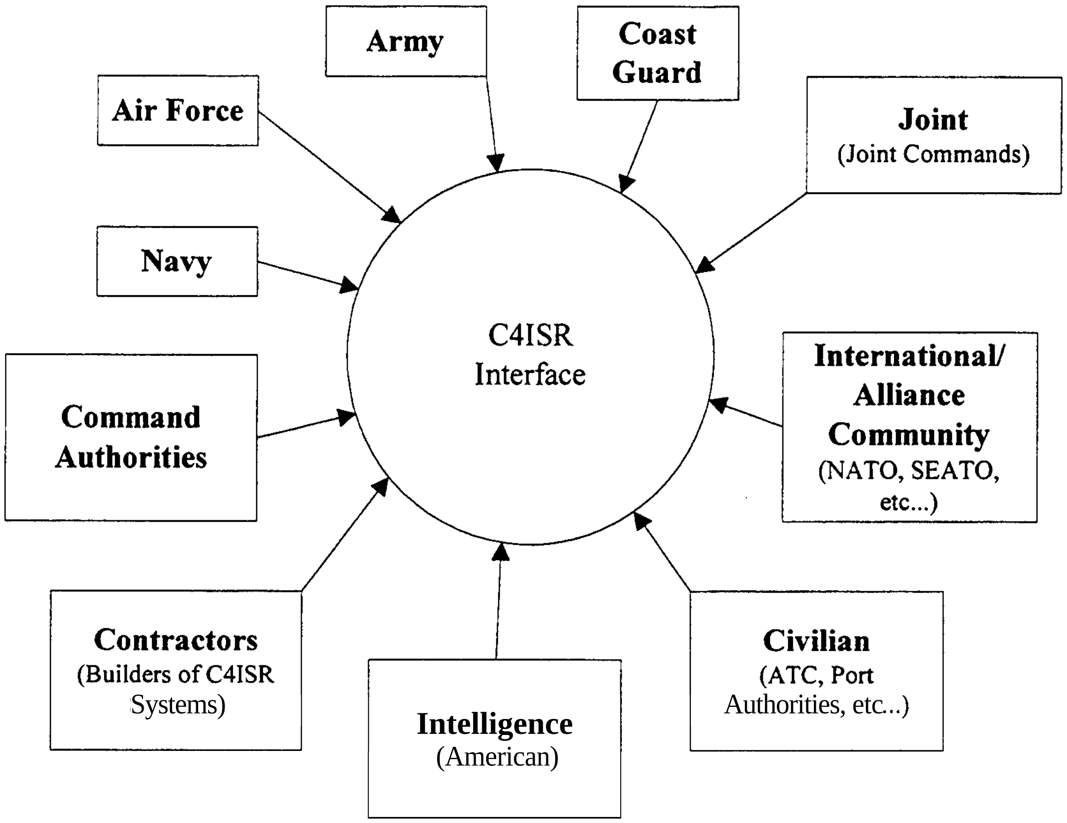
Speed to Deployment - The first application of Enhancing the Common Operational Picture was developed in less than two months. The result was a web-browser-based Java applet that provided interactive mapping and track manipulation, using maps generated on the fly by JMTK (Joint Mapping Toolkit) server in the DII COE, and tracks from TDBM (Track Database Manager).

10. Appendix D: Strawman C4ISR Interoperability Requirements

To help us understand the context for the interoperability technology assessment performed for JBC in this study, we asked students in an NPS software engineering course to do an initial requirements analysis for C4ISR system interoperability as a class project. The students were not experts in the process, but most **of** them were military officers with relevant first hand knowledge and reasonable depth of domain expertise. These **8** students provided a representative sample of the relevant stakeholders and their backgrounds covered a reasonable fraction of "joint" viewpoints: US Army, US Navy, Greek Army, Taiwan Airforce, and Singapore air traffic control engineer (government civilian). Constraints were left imprecise to avoid problems with handling classified data.

This appendix contains a cleaned up version of the stakeholder and goal hierarchies produced by the class projects, edited to integrate the most important points, to abstract and simplify, and to delete mistakes. The results are summarized here because they may be of some use to JBC. Software engineering experience indicates that identifying the relevant stakeholders can be quite difficult for complex problems such as this one, and that neglecting the viewpoints of some relevant categories of stakeholders can pose significant risk of requirements faults and conceptual errors. Some formal domain models were also produced, but these are not included here because all of them contained flaws and the current JBC project did not provide any resources for repairing them. The requirements analysis was not part of the JBC tasking and JBC did not provide any resources for this work.

Stakeholders



Army
Coast Guard
Air Force
Joint (Joint Commands)
Navy
C4ISR Interface
Command Authorities
International/ Alliance Community (NATO, SEATO, etc...)
Contractors (Builders of C4ISR Systems)
Intelligence (American)
Civilian (ATC, Port Authorities, etc...)

Partial Stakeholder Hierarchy

- Command Authorities (civilian government)
- Joint Commands
  - Joint Chiefs of Staff
  - EUCOM
  - ACOM
- US Navy
  - Numbered Fleet Commands
  - Battle Group Commanders
  - Amphibious Readiness Group
  - Anti-Submarine Warfare Squadrons
  - Ships' Commanding Officers
  - Warfare Commanders (Air, Surface, Sub-Surface, Space)
  - Ships' Crews
  - Detection Equipment Operators and Maintainers
  - Air Squadron Commanders
  - Naval Aircrews
  - Merchant Logistic Force (replenishment at sea)
- US Army
  - Major Commands (coordinate strategic level resources)
  - Army Commands (provide logistic and administrative support for Corps)
  - Corps Commands (many C4ISR requirements to fight the battle **72-96** hours in advance)
  - Division Commands (smallest self contained tactical unit to fight combined arms engagements)
  - Brigade Commands (lowest level in chain of command with access to tactical internet)

    - Battalion Commands (large logistic & intelligence requirements)
    - Company Commands (Special Ops/Aviation)
    - Crew Level Weapons/Sensors
- US Air Force
- US Marine Corps
- US Coast Guard
- DoD Centers
    - Defense Information Agency
    - National Imagery and Mapping Agency
- Intelligence Agencies
    - US Central Intelligence Agency
    - Defense Intelligence Agency
    - National Security Agency
    - Allied Intelligence Agencies
- Civilian Centers
    - Port Authorities (interaction with military ships)
    - Air Traffic Control Authorities (interaction with military aircraft)
    - Police and Fire Departments (disaster relief operations)
    - Hospitals and medical community (possibly mobilized in case of war)
    - Merchant Marine (possibly mobilized in case of war)
    - Ministry of Vehicles (possible mobilization of vehicles in wartime)
- International Allies
    - NATO
    - SEATO
    - SETAF
- Developers of Systems (contractors)
    - Software developers
    - Computer hardware developers
    - Network and communications equipment developers
    - Developers of military platforms (ships, aircraft, land vehicles)

Goal Hierarchy

1. Provide track interchange services

1.1. Accept inputs from all ground-, air-, sea-, and space-borne sensors.

1.1 - I Be globally accessible from below the surface of the sea to sensors positioned in geo-stationary orbit around the Earth.

1.2.2. Be accessible via open, ubiquitous, de-facto communication channels and protocols (currently TCP/IP, the Internet, and the Defense Information Systems Network)

1.1.3. Provide a mechanism for all existing sensors to interact with the system.

1.1.4. Provide an interface standard for all future sensors to interact with the system.

1.2. Provide all relevant track data to all systems linked into the system.

1.2.1. Classify tracks (unknown, friendly, hostile)

1.2.2. Report the three dimensional location of the target

1.2.3. Report the time on target

1.2.4. Report the fault tolerance of the sensor to support data fusion

1.2.5. Report three dimensional velocity and acceleration

1.2.6. Report track identity (call sign, name, etc.)

1.2.7. Report known information (status, mission, armament, ...)

1.3. Accept requests for information from all ground-, air-, sea-, and space-borne C2 systems.

1.3.1. Be globally accessible from below the surface of the sea to C2 systems positioned in geo-stationary orbit around the Earth.

1.3.2. Be accessible via open, ubiquitous, de-facto communication channels and protocols.

1.4 Audit and log track interchange transactions

1.4.1. Provide query and reporting services for transaction audits

1.4.2. Track sensor inputs
1.4.2.1. Record reporting sensor identification
1.4.2.2. Record time of receipt
1.4.2.3. Record reported track information

1.4.3. Track requests and transactions with connected C2 systems
1.4.3.1. Record identification of connected C2 systems
1.4.3.2. Record track information sent to connected C2 systems.

1.5. The system must be able to resolve reports of the same target from multiple sensors.

1.5.1 The system must be able to resolve target conflicts

1.5.2 The system must exploit information from multiple sensors to provide a more accurate picture of position and movement than can be attained using one sensor.

2. Provide intelligence interchange services

2.1. Provide access to national intelligence databases

2.1.1. Provide access to NSA databases

2.1.2. Provide access to DIA databases

2.1.3. Provide access to imagery databases

2.2. Provide access to TENCAP systems

2.3. Enable horizontal exchange **of** SIGINT, IMINT, and HUMINT at all classification levels.

2.4. Provide Joint intelligence asset management services

2.5. Provide Joint intelligence collection management services

2.6. Connect to the Joint Worldwide Intelligence Communications System (JWICS)

2.7. Connect to the Secret Internet Protocol Network (SIPRNET)

2.8. Safeguard classified and sensitive information accordingly

2.8.1 Provide encryption methods to prevent unauthorized access or modification to sensitive data.

2.8.2 Provide Multilevel Access Control for access to multiple levels of classified data

2.8.3 Prevent leakage of classified data via covert channels.

3. Provide logistics interchange services.

3.1. Communicate requests **for** logistics support

3.2. Report Logistics Status (current inventories and locations)

3.3. Track Logistic Request **Status** (allocations, location, time to receip., potential obstacles)

3.4 Handle all types of logistic support (food, individual TOE items, fuel/POL products, barrier material, ammunition, PX items, major end items, medical, repair parts, civilian items, other)

4. Provide planning interchange services.

4.1 The System shall provide means to request Mission Planning Services

4.1.1 The system shall provide means to originate, query and update mission plans
4.1.1.1 Mission plans include Fight Plans/Air Missions.
4.1.1.2 Mission plans include Ships Routes/Missions and Plans.
4.1.2.3 Mission plans include land Unit Objective/Missions and Plans.

4.2 The system shall provide means to request tactical geographic update Services
4.2.1 The system shall provide Danger Areas and Activation times.

4.2.2 The system shall provide firing ranges and firing information.
4.2.3 The system shall provide means to exchange map information.

5. Provide operational interchange services.

5.1. The system shall provide Battlefield Control Measure exchange services.

5.2. The system shall provide a common communication infrastructure to improve coordination.

6. The system must be ultra reliable.

6.1. The system must be able to sustain considerable damage while still providing its most critical operations.

6.2 The system must be able to provide error control (fault indications and alerts when data has been **or** may have been corrupted)

6.3 The system must provide accurate data exchange in absence of fault indications.

7. The system must be fast enough.

7.1. The system must provide "real-time" data exchange within areas of interest to war fighting commanders (1000 mile radius)

7.2. The system must provide high-speed (near real-time) data exchange with worldwide coverage.

7.3. The system must provide timely data exchange for secondary communications when there is available bandwidth.

7.4 The system must be able to transmit large files at **high** data rates to support high-resolution photos and surveillance video.

8. The system must support heterogeneous policies of member C4ISR systems.

8.1. The system must support different performance requirements for different centers.

8.2. The system must support different security policies for different centers.

8.3. The system must support different communications protocols and message formats for different centers.

8.4. The system must support different networks, operating systems, programming languages, and computing platforms for different centers.

INITIAL DISTRIBUTION LIST

1.	Defense Technical Information Center 8725 John J. Kingman Rd., STE 0944 Ft. Belvoir, VA 22060-6218	**2**
2.	Dudley **Knox** Library Naval Postgraduate School Monterey, **CA** 93943-5100	2
3.	Research Office, Code 09 Naval Postgraduate School Monterey, CA 93943-5000	1
4.	Gerard Hufstetler Architecture Division Chief Joint C4ISR Battle Center 116 Lake View Parkway, Suite 150 Suffolk, VA 23435-2697	1
5.	Institute for Joint Warfare Analysis Naval Postgraduate School Monterey, CA 93943	1
6.	Prof. V. Berzins Computer Science Department Naval Postgraduate School Monterey, CA 93943	**8**