



Whitemarsh  
Information Systems Corporation

## Data Management Program: Database Interface Architectures

Whitemarsh Information Systems Corporation  
2008 Althea Lane  
Bowie, Maryland 20716  
Tele: 301-249-1142  
Email: [mmgorman@wiscorp.com](mailto:mmgorman@wiscorp.com)  
Web: [www.wiscorp.com](http://www.wiscorp.com)

## **Table of Contents**

Acknowledgments .....	iv
1.0 Introduction .....	1
2.0 Point-to-Point Interfaces .....	1
3.0 Near Term Baseline Data Architecture: Integrated Data Environments .....	4
4.0 Target Data Architecture: Infosphere based on Standard Data Structures .....	7
5.0 Database Interface Architecture Summary .....	13



## Figures

Figure 1. Three forms of Data Interchange. ....	1
Figure 2. Point-to-point system translator. ....	2
Figure 3. Integrated Data Environments ....	4
Figure 4. Three Integrated Data Environment options. ....	5
Figure 5. Ideal IDE configuration of systems to minimize interfaces ....	7
Figure 6. Option One for InfoSphere: Integrated data environment. ....	9
Figure 7. Option 2 for InfoSphere: Direct data exchange via system internal translations. ....	11
Figure 8. Option 3 for InfoSphere: Direct data exchange via shared data structures. ....	12



## **Acknowledgments**

This material is an evolution of documents that were updated during the time frame: September 2003 through December 2004. Contributors were Bruce Haberkamp, James Blalock, and Michael Gorman of the Office of the CIO, United States Army. The foundational components of this work has been favorably reviewed by subject matter experts within the U.S. Department of Defense.



## 1.0 Introduction

The three data architectures for exchanging data illustrated in Figure 1 are:

- Point to point interfaces
- Integrated data environments, and
- Infosphere data environments based on ISO 11179 based shared data elements and segments

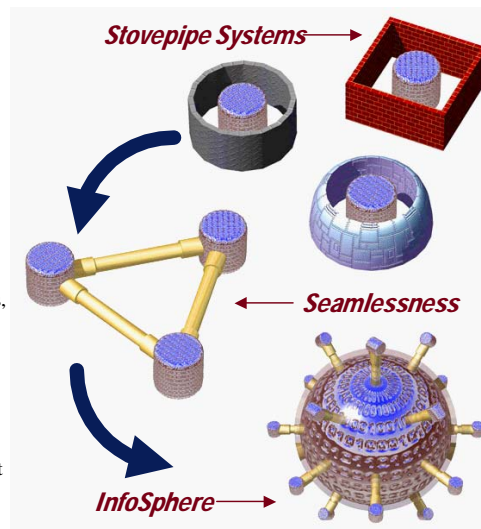
The data management program's data interface strategy is to proceed from the point-to-point environments to integrated data environments and possibly to "infosphere" data environments that are based on ISO 11179 based shared data elements and shared data segments. This gradual migration is also depicted in Figure 1.

## 2.0 Point-to-Point Interfaces

The myriad of existing legacy "stovepipe" information application systems are generally based on database designs of tables and columns, which, for the most part, are intended to characterize common data distilled from the data requirements derived from common, or similar processes. It is, however, because these database designs of tables and columns are not exactly the same, one

### Future Vision of Joint Interoperability: Beyond Seamlessness

- **Today (Baseline)**
  - we still rely on stovepipe systems
  - many data ownership issues
  - interoperability through "gadgets"
  - technological fixes of limited useful lifespans
- **Near-term: Seamlessness**
  - integrated (but still separate) systems
  - common message sets, operating systems, and hardware
- **Far-term (Target): InfoSphere**
  - single, virtual database
  - platform independent
  - ownership of data elements, data may not reside on system you own



**Figure 1.** Three forms of Data Interchange.



with the other, that impedes direct data sharing between application systems.

Approaches for achieving interoperability between application systems focus on creating interface translators (often called "data mediators"). These allow legacy application systems to share data even though the structure and semantics of their respective databases of tables and columns are defined differently.

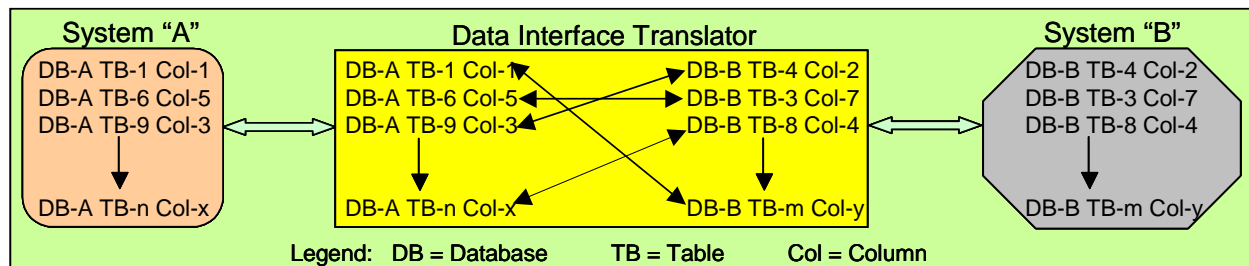
In the legacy environment, interoperability is commonly achieved by direct point-to-point translation systems between any application system "A", and any application system "B". These translators effectively map the database designs supporting application system "A" to the equivalent database designs in application system B and vice versa. Figure 2 illustrates this approach. Each database table column set mapping represents a semantic translation of the same or different database table columns across the same set of data values.

The way this would be accomplished in a DBMS environment would be to have the System A create a SQL View of the same granularity as a System B view. The Interface Translator in the middle would map the view columns from System A to the view columns from System B. The System A SQL view then maps to the specific database table columns. The System B SQL view maps to its specific database table column.

It is traditional that database applications are program centric. Thus, there is very little incentive to build a database design to support wide community. Because of this narrow focus, different systems from within different environments have negotiated point-to-point interfaces. That is, direct point-to-point translation systems between any application system "A", and any application system "B".

The number of point-to-point translators required is a direct function of the numbers of distinct pairings of application systems sharing data. For  $n$  number of distinct systems, this could mean as many as  $n(n-1)/2$  point-to-point interfaces. For example, a network of five applications systems, each required to share data with the other four, would require 10 point-to-point interfaces. If 50 systems are to maintain interfaces with each other the quantity of such point to point interfaces is  $(25*24)/2$ , or 300 interfaces.

Each interface consumes considerable resources. In a USAF study done in the middle 1990s, it was determined that each interface environment cost the \$335,000 each and every year.



**Figure 2.** Point-to-point system translator.



Consequently, the set of interfaces across 5 systems would cost about \$3.35 million. The cost of the 300 interfaces would cost \$100.5 million per year. That is a considerable amount of resources. In addition to financial resources, the cost in scarce technical talent is considerable. In the first case, \$3.3 million in new battle field systems cannot be built. In the second case, \$100.5 million cannot be built each and every year.

Simple point-to-point interfaces are complex, and complex interfaces are very difficult to create and vastly more costly to maintain. Complicating factors for point-to-point interfaces are mismatches that can occur in the granularity, time synchronization, and code set values between mapped database table columns. This explains why these kinds of interfaces are generally costly to create and maintain, and difficult to manage and keep current.

XML has been touted as the low-cost solution to point-to-point interfaces. However, without the prerequisite data standardization, the only savings resulting from XML is the ability to actually compose and read ASCII files. If neither side of a data exchange understand the semantics, granularity, and precision of the data, then all the work of understanding the source's data is still necessary.

The medium of data exchange would do little to make this approach desirability. If the exchange were XML based, System A would have to have a XML schema for the export. There would then have to be a XSLT to translate System A data to System B data, and finally, there would have another XML schema for the import. Arguably, in this scenario, XML represents more work.

If net-centricity merely requires that programs create and post data asset metadata catalog data, create and post XML schemas for program data exchange transactions, and post the URL for data asset access, then not only will no progress be made on eliminating point-to-point data exchanges, but also programs will have the blessing of those espousing this simple tag and post approach, and then exoneration from any obligation to subsequently create the necessary consensus based data standards necessary to make net-centricity a success. Without consensus based data standardization at least at the community of interest level, there will be a blizzard of metadata catalog entries and a similar blizzard of data asset XML Schemas that will have to be sorted through by data asset users. While there may be increased interoperability, it will only be the connectivity type. It will not be the understandability interoperability with a minimum of complexity and latency.

In short, to achieve any meaningful set of benefits from net-centricity then first the quantity of interfaces must be severely reduced, and then because of these interface reductions, there will then be a similar reduction in the quantity of metadata catalog entries and XML Schemas.



### **3.0 Near Term Baseline Data Architecture: Integrated Data Environments**

Recent initiatives for achieving information interoperability have focused on creating data interface translators that translate data between groups, or "families", of legacy source systems that exist within some common process, subject area, or functional business area. Such systems are known as Integrated Data Environments (IDE). The common primary characteristic of these systems is the creation of some kind of common IDE schema to which each of the participating source system can be mapped (See Figure 3). So for a network of five source systems that share data with each other, the single IDE interface performs the same function as the former ten point-to-point interfaces. This is a definite improvement, but it is still not a complete solution. This does not, for example, immediately solve the mismatch problem cited above. Rather, it makes the problem area in which it is to be solved more focused and manageable.

IDEs tend to be narrowly focused on application systems that support specific functional business areas or processes. There is an interoperability environment among the individual application systems that participate in the IDE. The IDE becomes a "super database" of all data that is to be shared among all IDE member systems. Such super databases are not free. They require definition, system development, and ongoing maintenance. These databases represent a high level of policy-consensus across a wide community.

But this solution still does not provide for interoperability between two application systems that are not members of the same IDE, except in the unlikely event that the database schemas of the two systems are identical or equivalent. Assuming that the database schemas of two application systems, C and F, are not identical or equivalent, Figure 4 demonstrates three options for how data might be shared when application system C in IDE #1 develops a requirement to share data with application system F in IDE #2.

The three options are:

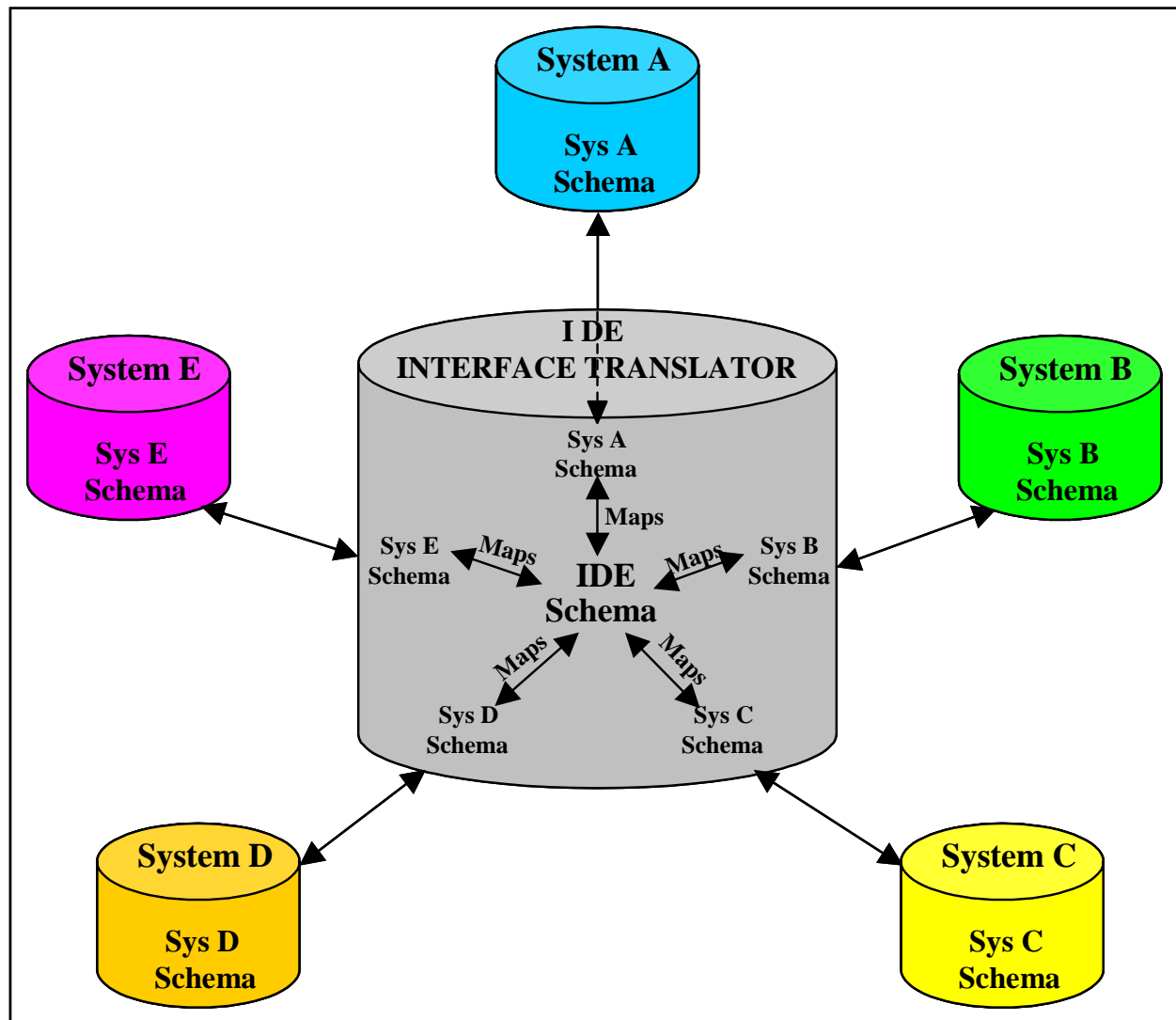
**Option one:** This reflects the situation where system C would participate in both IDE #1 and IDE #2. This option creates the added flexibility for system C to share data with systems D and E as well as with system F should that need develop.

**Option two:** This is the simple point-to-point interface and the least flexible of the three options.

**Option three:** This would require an interface translator between IDE #1 and IDE #2 for the IDE systems' shared data. In this case, if data is to be shared between System C and System F, then that shared data must be stored in IDE #1 and IDE #2 respectively. This would allow system C to share data with system F via the interface between the two IDEs. The added benefit to this solution is that it would also permit data sharing between all the other systems participating in the two IDEs should such a requirement develop.

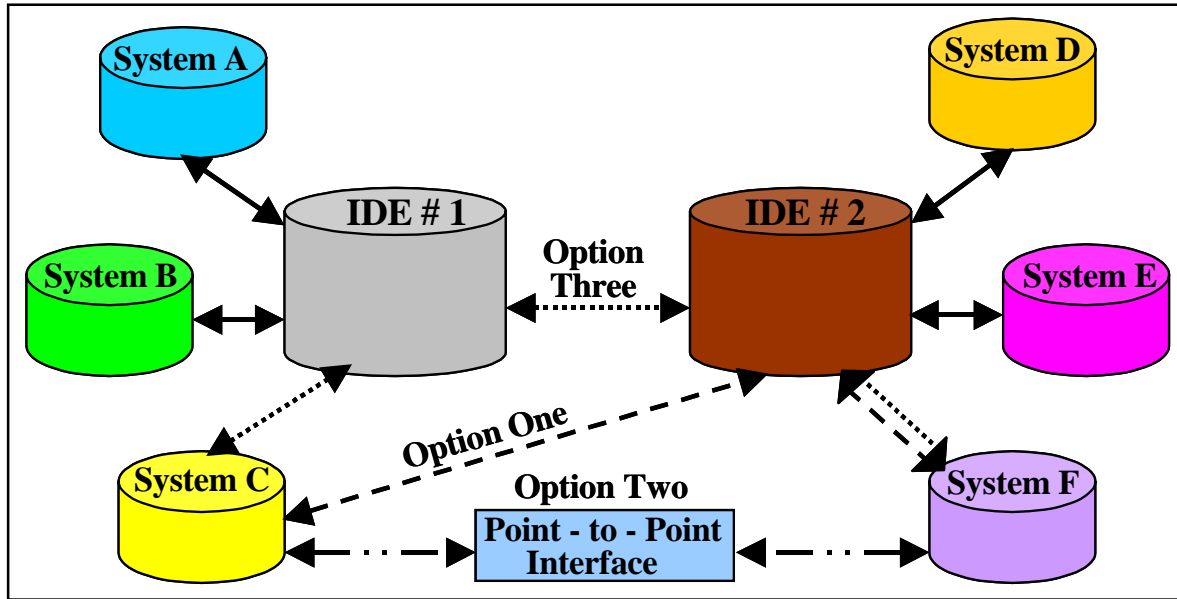






**Figure 3.** Integrated Data Environments





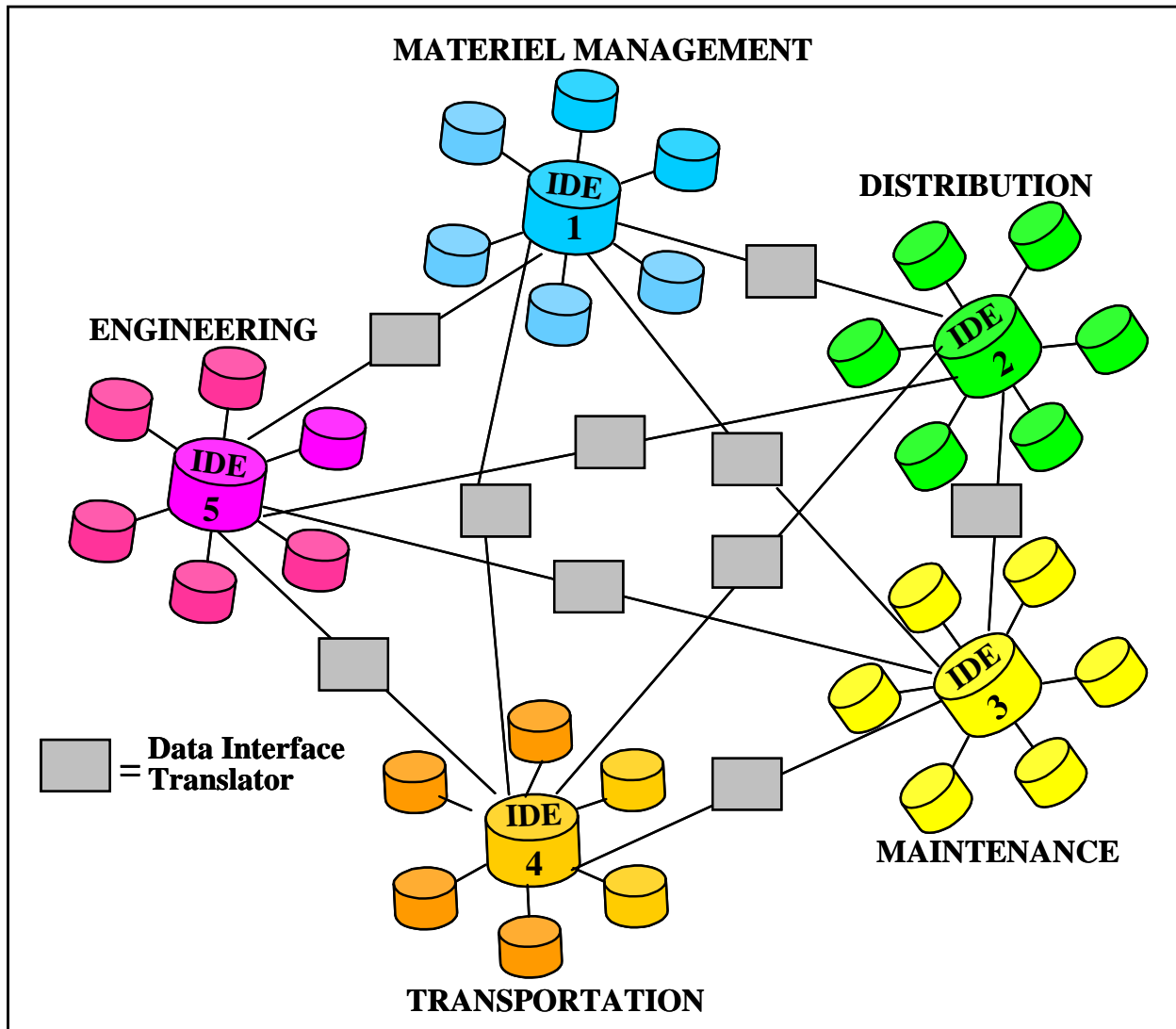
**Figure 4.** Three Integrated Data Environment options.

In spite of their complexity and cost, the advent of IDEs is an important incremental step on the path toward achieving transparent and seamless information interoperability among automated information systems. Their main advantage is that they offer the promise of creating interoperability among systems supporting distinct functional business areas, and by so doing they greatly reduce the number of required point-to-point interface translators between individual application systems. Consider that for a network of 30 application systems, the potential number of point-to-point interfaces needed to support understanding-based data interoperability between any two systems could be as many as 435. If these same 30 systems could be organized into five IDEs, each consisting of six application systems, then the number of data interfaces required to support interoperability between each pair of IDEs would be reduced to 10 - a significant reduction from a possible 435 point-to-point interfaces between individual systems. (See Figure 26).

In spite of their complexity and cost, the advent of IDEs is an important incremental step on the path toward achieving transparent and seamless information interoperability among automated information systems. Their main advantage is that they offer the promise of creating interoperability among systems supporting distinct functional business areas, and by so doing they greatly reduce the number of required point-to-point interface translators between individual application systems. Consider that for a network of 30 application systems, the potential number of point-to-point interfaces needed to support understanding-based data interoperability between any two systems could be as many as 435. If these same 30 systems could be organized into five IDEs, each consisting of six application systems, then the number of data interfaces required to support interoperability between each pair of IDEs would be reduced



to 10 - a significant reduction from a possible 435 point-to-point interfaces between individual systems. (See Figure 5).



**Figure 5.** Ideal IDE configuration of systems to minimize interfaces

#### 4.0 Target Data Architecture: Infosphere based on Standard Data Structures

Even as IDEs represent a significant improvement over point-to-point interfaces for achieving information interoperability, a better and more efficient means for achieving this goal would be to define database tables based on shared data segments, which, in turn, are based on ISO 11179 based shared data elements. Thus, for every atomic business fact required to conduct the business, only database table column is required to represent that atomic business fact. No



duplicate or redundant database table columns need to exist to represent the same business fact. If indeed such shared data segments were created, then the ability to develop semantic translators, based on shared data segments with their ISO 11179 based shared data elements, to facilitate data exchange between individual application systems or between IDE systems, would be greatly enhanced.

Shared data segments standardize the complete set of data for policy instances. When constraints and state transforms are added that then manage the life cycle of instances, the shared data segments can be transformed into business rule infused database tables.

The existence of shared data segments would create several options for exchanging data between systems.

**Option 1:** This option requires individual application systems to have their database tables based on shared data segments that enable mapping among database table columns. (See Figure 6).

**Option 2:** This option has the shared data segment based logical database schema as the core schema for an "Enterprise IDE." Every source application system would then create a localized shared data segment based data portal to which they would have translated the data they wished to share. With this option, the various systems would have been perceived as all having shared data segment based common schemas.(See Figure 7).

**Option 3:** The third option would require all source application systems to develop their database designs using shared data segment based database schemas. This would entirely eliminate the need for any data translation since the structure and semantics of any particular database table column represented in any system would be identical. Direct exchange of data could then take place. (See Figure 8).

The extent to which data would be "instantly" translatable would still depend, for example, on common data item value sets, granularity, and time synchronization. Data warehouses, the class of databases designed especially to support analysis and reporting, are increasingly common and in such instances, atomic data is often cast into different time and summarization dimensions. In these situations, while data warehouse database table columns may be based on shared data segments, they still require further analysis before value set translation between IDE systems and data warehouses is possible. However, this analysis would be greatly simplified by the deployment of shared data segments.

The three options depicted in Figures 6, 7, and 8 represent incremental steps for how an enterprise information environment composed of legacy stovepipe systems might migrate to an information interoperability environment through the use of standardized data structures.

**Incremental Step 1:** This corresponds to Figure 6 where legacy systems would create internal mappings of their legacy system database table columns to the ISO 11179 based shared data elements represented within the shared data segments for those legacy system database table

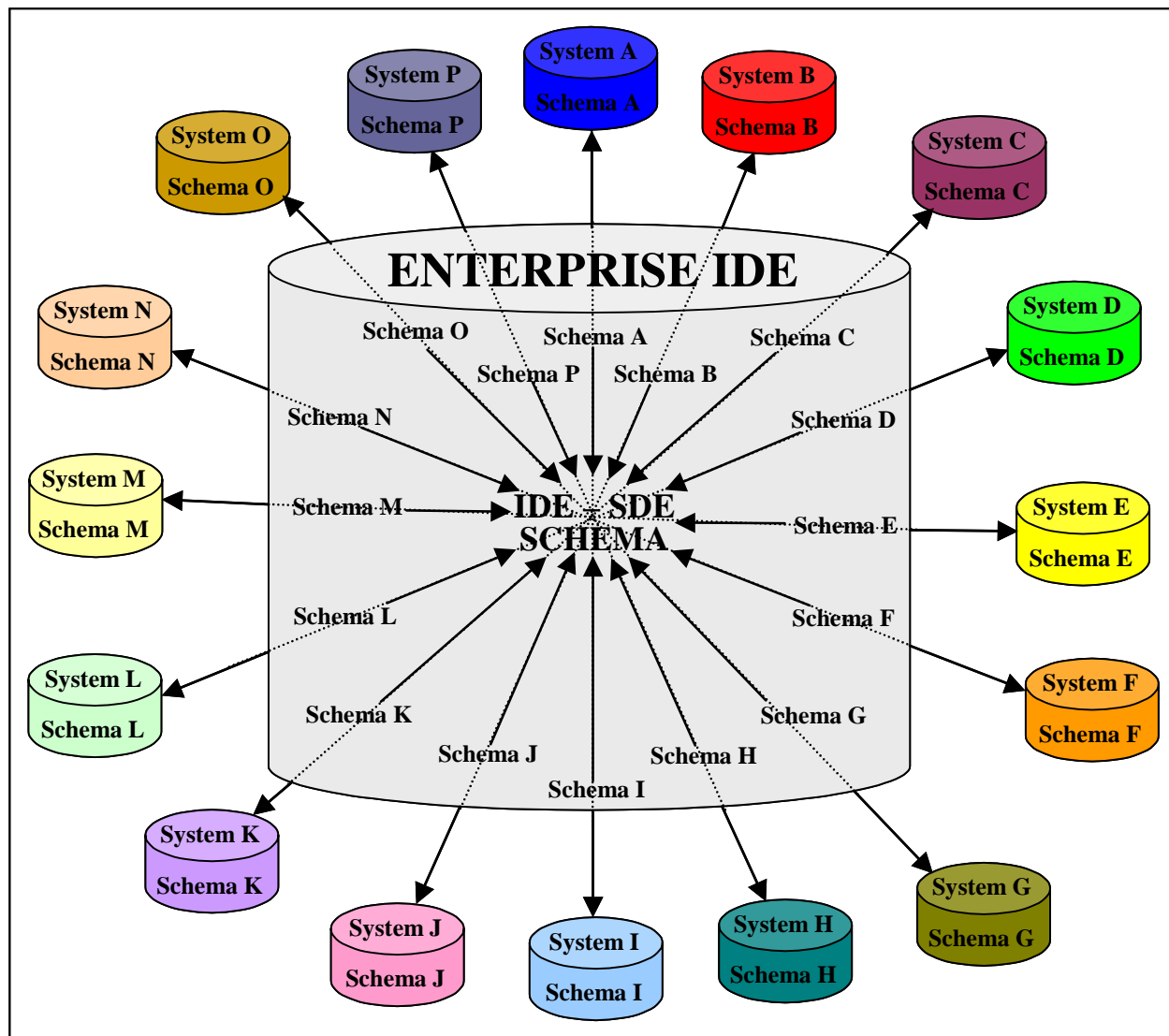


columns with data sharing requirements. Data could then be directly shared between systems that had implemented internal mappings between legacy system database table columns and columns within the IDE's shared data segment based database table columns. Point-to-point interfaces would still be required to share data with systems that had not implemented internal mappings.

**Incremental Step 2:** This is accomplished over time as upgrades and modifications increase the amount of implemented standardized data in use in source legacy application systems and reach a critical mass, migration would occur through the development of functionally oriented IDEs, evolving to a single Enterprise IDE corresponding to Figure 7.

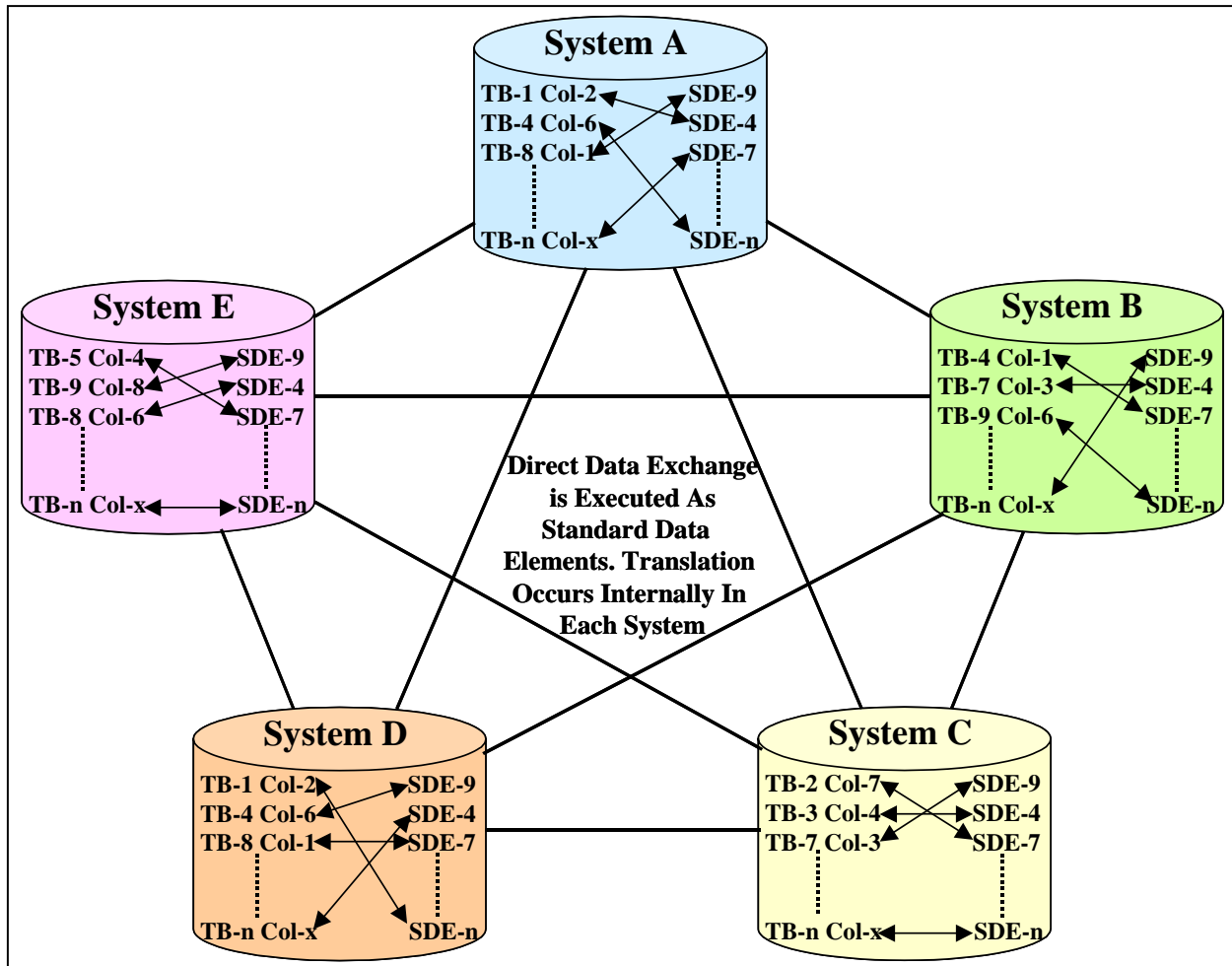
**Incremental Step 3:** This corresponds to Figure 8 when all enterprise systems have fully implemented standardized data as the basis of their internal schemas, either through modification or by replacement with new systems. This would equate to an interoperable information environment characterized by direct point-to-point data exchange between systems based on shared data structures. This kind of information environment was a goal of the DoD Data Standardization Program that has resulted in the creation of the Defense Data Dictionary System (DDDS) repository of DoD ISO 11179 based shared data elements. It is unfortunate that suboptimal engineering caused this program to fail as the understanding-based data-interoperability problems that the DDDS program were targeted to address still exist, have grown much larger, and are more critical today than ever to solve in net-centric environments.





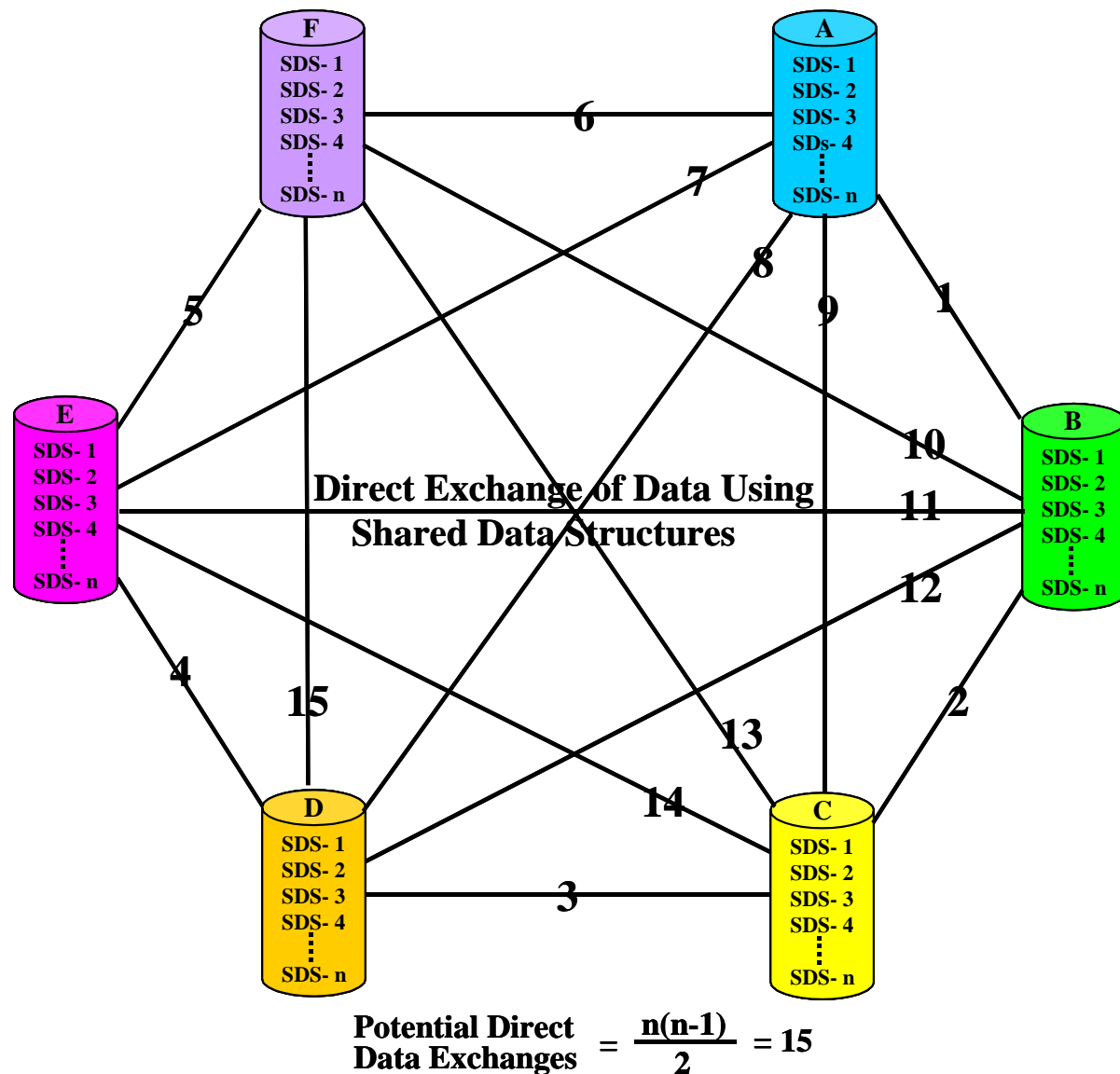
**Figure 6.** Option One for InfoSphere: Integrated data environment.





**Figure 7.** Option 2 for InfoSphere: Direct data exchange via system internal translations.





**Figure 8.** Option 3 for InfoSphere: Direct data exchange via shared data structures.





## **5.0 Database Interface Architecture Summary**

Starting in the early days of data processing, all data was owned, possessed, and processed by the program within which the data was captured. As time moved on, other programs wanted to share data. This was done, not by sharing, but by extracting, transforming and loading a subset of that data. Each such effort was really another system. Eventually, there became a massive quantity of these ETL systems, each built with a point-to-point orientation. The enterprise's view, semantics, granularities and precision were no where to be found. As shown above, the financial obligations to manage this essentially zero-value effort was significant. The USAF projected in 1995 that this amount was \$175 million. Across the DoD it would likely then be \$1 Billion.

It is therefore key that this massive infrastructure of stove-pipe solutions be dismantled and replaced with database exchange environments that are not only less costly and more efficient but also that reflect enterprise view, semantics, granularity, and precision. In short, Net-Centric.

