Technical Note

2001.13

INVERSION OF CONTROL

The Impedance Mismatch in Integrated Engineering Design Systems

This technical note describes an issue in the integration of commercial off the shelf (COTS) components. This issue is a member of the *Impedance Mismatch* problems found [7] when commercial off the shelf components are assembled into systems.

This mismatch occurs when event, control sequence, or data semantics of two or more participating application domains are *mismatched*.

During the system integration process the impedance mismatch must be addressed through some means, either through an integration layer which hides the mismatch or through an integrating service, such as CORBA, which facilitates the impedance adaptation between the applications.

Separation of Concerns

The primary issue in defining an integration architecture is the ability to *separate* the concerns of each application component, while simultaneously creating an integrated system. This separation has several dimensions. Without this ability to separate the concerns, the resulting system will not only be tightly coupled and resist change and enhancement, but failures in one portion of the system will be propagated to other portions of the system, across these tightly coupled boundaries.

A Simple Manufacturing Example

A simple example of the inversion of control problem can be found when CAD, ERP, PDM and product configuration systems are integrated. Figure 2 describes a very simple set of integration components commonly found in the manufacturing domain.

- *Manufacturing* – ERP systems provide the data and work flow management for manufacturing processes. Manufacturing Bills of Material are maintained by the ERP system. These BOMs describe the *as manufactured* entities. In most cases these BOMs must have part numbers in order to be identified by manufacturing and purchasing
- *Engineering* – PDM systems provide the data and process management for *as designed* elements of the product line. Part numbers need not be assigned to *work in progress* engineering entities. Before an engineered product is released for manufacturing, some type of Bill of Material is needed. This BOM is usually generated through a configuration process.
- *Design* – CAD systems provide tools needed to assemble and analyze products in response to customer needs. These tools provide geometric as well as attribute based design. *Parametric* drawing engines are used to render different products from similar underlying geometric entities. The parametric assembly of products generate Bills of Material as well as other *derived* information from a model rather than from a list of elements in a static database.
- *Product Structure* – has been the traditional roles of Product Data Management (PDM) systems. With the introduction of *engineer to order* business models, Configurators now provide the intelligence to construct products from standard as well as engineered components. Change control is the primary role of the PDM system.

Assembling these four systems into an integrated solution exposes the inversion of control problem. Each COTS product assumes it initially operates as a stand alone entity with control, data, and semantics residing within the product. This is a natural result since each product was *purpose built* to solve a specific set of problems, targeted to specific industries. The *ownership* of data, process, and semantics is a natural outcome of the product marketing approach of each vendor.

Four Dimensions of Integrated Systems

To understand how the impedance mismatch arises in a traditional system integration, we must identify the core cause of this mismatch and the dimensions of the solution.

The problem introduced by component–based systems is that components embed assumptions about the architectural and operational context in which they operate.

← Move	Apart →	Separation Motivation
Data	Trans-port	The movement of data is seperated from the actual data itself. This allows multiple mechanisms for connecting applications domains to be used without regard to the format or form of the data.
Publish	Sub-scribe	Applications that publish events know nothing of applications subscribing to events.
Internal Event	External Event	Private events change the operational flow of an application without affecting external processes. Public events synchronize large–grained processes without impacting internal fine–grained processes.
Internal Data	External Data	Data transformations take place across application boundaries *only* when needed.
Connec-tion	Delivery	The delivery of messages and events is seperated from the management of the connection over which the delivery process takes place.

Figure 1 – Separating Concerns in Integrated Systems

These assumptions often conflict with the architecture of the integrated meta–system and with the assumptions of the components of other parts of the system. [1]

There are four dimensions to this impedance mismatch: [2]

- *Control integration mismatch* – describes how components make requests from other components. The mismatch occurs when the thread of control is maintained in each application domain, with no external means of interrupting the thread of control or causing the thread of control to be redirected by outside means. Examples of this control mismatch in the *Configurator* include:
  - The AutoCAD domain controlling the creation and updating of design objects in the configurator database. Once the design object has been created or updated, other applications must be signaled that this change has occurred. This signaling needs to be done through some means shared by all the applications participating in the integration.
  - The PDM domain controlling the product structure and revision control of this structure. Engineer to order product make use of product structure information. This information defines how individual components can be assembled

[1] "Discovering a System Modernization Decision Framework: A Case Study in Migrating to Distributed Object Technology," Evan Wallace, Paul Clements, and Kurt Wallnau. *NIST Interagency Report*, *Software Engineering Institute*. The selection of a distributed object technology may appear as a simple process. *It's what everyone is doing*. This is not the case when the business domain experience is grounded in a centralized "mainframe" culture. Many of the process, state, and data assumptions of the mainframe world are no longer valid (or valid in different ways) in the distributed object world. One simple example of this impact is the design and implementation of the inter–process communication infrastructure. Timing, communication reliability, remote object method invocation, fault detection and recovery are all system architecture issues that must be addressed in a different manner than the previous mainframe systems. It could be argued that given the fault tolerant nature of the engineering and manufacturing systems, the underlying problems are the same. However, it has been shown in the literature as well as field experience that there are a unique set of issues found in the distributed object architecture that are not found in the centralized processor architecture.

[2] The fundamental problem in the integration of heterogeneous systems is how to *normalize* the semantics of the data and process flow. The following is just a sample of the research and practices information available on the subject.

"What Do Groups Need? A Proposed Set of Generic Groupware Requirements," Munir Mandviwalla and Lorne Olfman, *ACM Transactions of Computer–Human Interaction*, 1(3), September 1994, pp. 245–268.

"Federating Heterogeneous Workflow Systems," Andreas Geppart, et al, Technical Report 98.05, Department of Computer Science, University of Zurich.

"Distributed Data Management in Workflow Environments," G. Alonso, B. Reinwald, and C. Mohan, *IBM Almaden Research Center Report*. www.almaden.ibm.com.

"An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure," Dimmitrios George Kapoulos, Mark Hornick, and Amit Sheth, *Distributed and Parallel Databases* Volume 3, 1995, pp. 119–153.

"Some Practical Advice for Dealing with Semantic Heterogeneity in Federated Database Systems," V. Ventrone, *The MITRE Corporation*, Bedford, MA and S. Heiler, *GTE Laboratories*, Waltham MA.

"Workflow based Applications," F. Leymann and D. Roller, *IBM Systems Journal*, 36(1), 1997, pp. 102

into units or other high level deliverables. Changes in the product structure must be communicated to the configuration process, but these changes must not *undo* configurations that have taken place in the past.

- *Data integration mismatch* – describes how components make data available to other components. The mismatch occurs when the semantics of the data elements are not *normalized*.
- *Process integration mismatch* – describes the end–user processes supported by a collection of components. The mismatch occurs when the workflow processes within two domains do not share the same semantics. This is actually an interoperability problem between the two process domains.
  - Each application domain has been created assuming it is the center of some process. Although each application is provided with an API, the process orientation of this API assumes the thread of control is maintained by the application.
  - The *Configurator* creates an *integrated thread of control* environment in which each application must be capable of relinquishing control while it is cooperating with other applications.
  - This transformation from control–centric to cooperation–centric takes place through the shared event signaling and control flow system.
- *Presentation integration mismatch* – describes how end–users interact with the system through the User Interface. This mismatch occurs when the user interface elements of the system contain differing semantics but similar format and syntax. The contents of tables, fields, command buttons, and other User Interface elements are not normalized with the underlying database schema.
  - There are several *primary* user interfaces in *Configurator*. AutoCAD, the PDM system, the rule definition interface, the BOM textual interface.
  - Each of these User Interfaces presents and application domain–centric view of data and process.

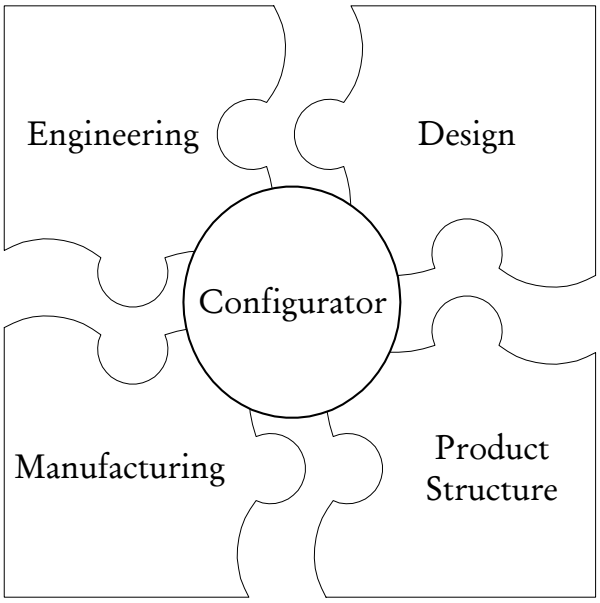

Figure 2 – Integrating Diverse System Components

In this heterogeneous COTS environment:

- Each application domain makes use of a specific implementation paradigm. The interactive process of configuring a product makes use of local and global data. Sharing this data and synchronizing the design and editing process creates opportunities for impedance mismatches. Each application assumes this private paradigm can be syndicated to other application domains. This assumption is rarely true, since the semantics of the data and control processes are unique for each application. [3]
- Each application domain assumes it is independent from other application domains. This independence includes the flow of control within the domain as well as the announcement of events and changes in state across the domain boundaries. This domain independence creates an *inversion* of control or the lack of explicit control across these application boundaries. [4] This inversion of control is a fundamental architectural flaw found in tightly coupled integration architectures, where multiple information producers and consumers are present.
- Each application domain makes use of unique file formats and communication protocols. Engineering systems, design systems, manufacturing systems, sales & marketing systems and the integration components which connect them together, all pro-

[3] "Integrating Islands of Automation," Michael Stonebraker, *eaiJournal*, September / October 1999. www.eaijournal.com.

[4] "Object–Oriented Application Frameworks," Mohamed E. Fayad and Douglas C. Schmidt, *Communications of the ACM*, 40(10), October 1997, pp. 32–38.

vide different file formats and protocols. In some cases, the format of the data is also unique to the operating system or the underlying database management system. [6]

- The event signaling, thread control, exception handling, and other low level software behaviors are difficult to control – even if the systems are intentionally designed to be compatible. [5] Each application domain assumes a unique mechanism for controlling the thread of execution. The synchronization of these execution threads across application boundaries is one of most the difficult problems in the integration of heterogeneous systems.
- Exceptions are reported within each application domain using the semantics of that application. Simple error codes and user response processes are unique to each domain. The unification of the exception semantics and the handling of the exceptions is a difficult and sometimes intractable task. [6]

In this heterogeneous integration environment, the influence of each domain – as well as many other intangible influences – creates an *impedance mismatch* between the COTS components. [7] The challenge for the system architect as well as the system integrator is to define a mechanism that integrates the disparate components while maintaining their unique functionality. [8]

In addition to the impedance mismatch between the COTS components, these components are also undergoing continuous change. [9] These changes create a moving target for the seamless integration of heterogeneous systems.

What Does This Mean in the End?

Each component in the integrated system presents a unique challenge to the system architect as well as the development staff. Starting with *enterprise application integration* platform, a loosely coupled architecture can be used to create a *federated* system of components, rather than a *tightly integrated* collection of components.

This *federation* strategy has distinct advantages over the traditional tight integration or monolithic database architectures found in competing products:

- Individual components need not know about other components in the *federation*. A shared object repository can be used to connect the business domains. Event and process synchronization is provided through a publish and subscribe mechanism.
- Content created in one domain remains in that domain until it is needed in another domain.
- An integration interface component provides a mechanism to expose the functionality of an application domain without creating a physical connection between these domains.

Business Drivers for Federated Systems

In order to address the integration (federation) business drivers for engineering and manufacturing systems, an architecture is needed that provides an economically viable product. These drivers include:

- COTS based user applications that provide the best of breed capabilities without constraining other applications.
- Data format transformation demands for internal formats that must be exchanged between the best of breed applications.
- Real–time aspects of design and configurations as raw input to the system.
- All the non–functional requirements of a 24 × 7 system.

---

[5] "Michael Stonebraker on the Importance of Data Integration," Lee Garber, *IEEE IT Pro*, May / June, 1999

[6] "Framework Integration: Problems, Causes, Solutions," Michael Mattsson, Jan Bosch, and Mohamed E. Fayad, *Communications of the ACM*, 42(10), October 1999, pp. 81–87.

[7] The term *impedance mismatch* is widely used in the database domain to describe the mismatch between SQL database technologies and Object Oriented technologies. This mismatch comes about through the query and updating processes that are distinctly different in their semantics. In addition, the concept of *architectural mismatch* is now well understood and cause of many of the problems found in the industry. To understand this issue and how it affects the design of UNA some background materials are available and should be read by anyone intending the make changes to the UNA. "Detecting Architectural Mismatches During System Composition," Cristina Gacek, Center for Software Engineering, Computer Science Department, University of Southern California, USC/CSE–97–TR–506, July 8, 1997, "Composing Heterogeneous Software Architectures," Ahmed Abd–el–Shaft Abd–Allah, PhD Thesis, University of Southern California, August 1996 and "Attribute–Based Architectural Styles," mark Klein and Rick Kazman, Software Engineering Institute, CMU/SEI–99–TR–022, October 1999.

[8] "Architectural Mismatch, or Why It's Hard to Build Systems Out Of Existing Parts," D. Garlan, R. Allen, and J. Ockerbloom, *Proceedings of ICSE '95*, IEEE Computer Society Press, April 23–30 1995, pp. 179–185.

[9] Based Software Development," Will Tracz, *Crosstalk*, January 2000, pp. 4–8. www.stsc.hill.af.mil/CrossTalk/index.asp.

Operational Drivers for Federated Systems

The reliance on distributed object technologies for the integration of COTS systems creates operational expectations not normally found in traditional engineering and manufacturing systems. These include:

- *Performance management*:
  - The collection of performance metrics from applications across large–grained boundaries.
  - Application response management for each application domain.
  - The ability to start and stop application server objects depending on the processing load.
- *Fault Management*:
  - Detection – of the fault, its source, and its cause.
  - Tracking – of faults to identify unreliable components.
  - Reporting – faults to support administration of the system.
- *Configuration management*:
  - Dynamic configuration of server and client objects to meet changing business application needs.
  - Rearrange the server objects to meet changes in load.
- *System management software*:
  - Software distribution and installation processes.
  - Enterprise operations console to manage all object and processes.
- *Application management software*:
  - Multi–level application lifecycle management – provides the means to manage fine–grained objects, server processes, the dependencies between these processes, and the introduction on non–intrusive 3rd party applications.
  - Performance monitoring across interfaces and within the application domains.
  - Event management for coordinated execution of distributed processes.

Object Technology Integration Platforms

An object can be thought of as an entity that responds to a set of *messages*. A given message causes an object to execute a given set of instructions. In reality, an *object* is a piece of code that owns things called *attributes* and provides services through *methods*. Objects have three properties that make them very useful for integrating heterogeneous systems – which is the principal problem in the publishing system application domain.

- *Encapsulation* – which allows an object to manage its own resources and limit the visibility into its behavior. An object *publishes* its interface that defines how other objects or applications can interact with it. However, the actual processing that takes place inside the object is not visible to the outside world. The object's implementation is *encapsulated* – hidden from public view.
- *Polymorphism* – which is a fancy way of saying that the same method can do different things, depending on the class that implements it. Objects in different classes can receive the same message from their clients, yet react in different ways.
- *Inheritance* – is the mechanism that allows programs to create *child* classes – known as *subclasses* or *derived* classes – from existing parent classes. Child classes inherit their parent's methods and data structures.

Many benefits are cited for object–oriented development, often to a degree that is unrealistic. [10] With the proper investment in architecture and design there are tangible benefits to the object–oriented programming model, the components that results from this model and the framework in which the components exist.

Conclusion

Without a clear strategy to deal with this integration *impedance mismatch* upfront, the gaps created will be revealed later in the integration process. These gaps are expensive to close, difficult to locate in a complex system, and reduce the reliability, performance, and robustness of the overall system.

[10] "Pitfalls of Object–Oriented Development," Bruce F. Webster, M&T Books, 1995. Although it is somewhat dated, this text is one of those *must read* works for every developer, manager, and product specialist. It describes the *myths* of object–oriented development, the consequences of these myths, and the prevention of the problems resulting from taking the myths at face value.