COOL66

STERLING SOFTWARE

E-Strategies

at CA-World

New Orleans, LA

2000

April 9-14

CBD for the Integration Project

April 9-14, 2000
New Orleans, LA

Tony R. Pierce
tpierce@ActiveDevGroup.com

Agenda

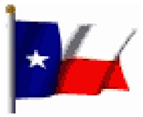
- Introduction
- The Project
- CBD: Myth & Legend versus Truth & Consequences
- Deriving Components
- COOL:Gen Models for CBD – What's Inside
- Identifying and Creating Operations
- One, Two, Three…How We Did It
- The Puzzle With No Straight Edges: Assembly
- Managing the Effects of Heavy Consumption
- Summary & Discussion…

STERLING SOFTWARE

ActiveDevelopment Group

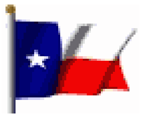
- **ADG is ActiveDevelopment Group**
  - Software Development
  - Consulting Services
  - Environment Upgrade and Support
  - Technical Education
- **Consultants with extensive COOL:Experience**
- **<allaire> ColdFusion and ASP Web development**

www.ActiveDevGroup.com



Project Overview

- **Terrific opportunity to provide a much-needed solution**
- **Integration of daily tasks for 150 users for Texas state agency (TNRCC)**
- **Conversion/Migration of 160 point-solution databases**
  - Redundant and conflicting data supported by a variety of storage mechanisms
  - Multiple technologies: FoxPro, Paradox, MS Access, …, and YouNameIt
  - Poor capabilities for support and enhancement, poor availability
- **The Primary Challenges**
  - "Trust me…"
  - Establish effective means of reporting to EPA
  - Improve the integrity of information through single-source application
  - Improve capabilities for enterprise support of the electronic tools
  - Establish standard operational procedures at an "office" level
  - Reduce reliance on "hacked together" point-solutions
  - Enforce business rules for manipulation of information



Project Technical Environment

- **Development Environment (Phases I & II)**
  - Windows NT, Oracle, Sterling COOL:Gen 4.1a, CANAM Report Composer, Crystal Reports, Sterling Client/Server Encyclopedia
- **Execution Environment (Phases I & II)**
  - 2-tier client/server
  - Windows 9X clients (GUI)
  - Sterling Client Manager
  - Sterling Transaction Enabler
  - HP-Unix server environment
  - Unix-based Oracle DBMS

Project Phases

Phase I

- •Analysis
- •Component Design
- •Prototype Application
- •Team and User Testing
- •Initial Installation

Phase II

- •Product Revisions
- •Functionality Enhancements
- •Application-generated Reports
- •Application Security
- •Further Qualification
- •Incremental Release Strategy

Phase III

- •Product Support
- •Internet Deployment
- •Approved Enhancements

Phase IV

- •Product Support
- •Incorporation into Agency ISP
- •Approved Enhancements
- •GIS Interface

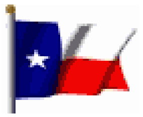
CBD: Myth and Legend
(this is what we hear)

- CBD takes longer to deliver product than the traditional methods we have used for years.
- Well, when it comes to CBD blah blah blah…
- Components are plug-n-play. Drop ‘em in and bada bing.
- Reference integrity is compromised by component development.
- Managing component object libraries is a nightmare.
- Determining the extent of consumption and the effects of change is impossible.

CBD: Truth and Consequences

(this is what we learned)

- **Our metrics are the best I've ever seen for a COOL project**
  - Command and Control
  - Emphasis on hard technical solution
  - Drop-dead pace
  - Components dramatically reduced our overall development burden
  - Developer "ownership"
- **Providers insist on "branding" CBD**
  - For marketing purposes, I think (actually, I know)
  - To be successful, need to stick to a standard
  - Need to make CBD more easily understood
- **Components are an ideal mechanism for supporting reuse**
  - But they are not "drop-ins" (have to know what you are doing)
  - Important to build technical expertise in building and managing components



CBD: Truth and Consequences

(this is what we learned)

- **Reference integrity across components is challenging**
  - But it **can** be done – about 7 different ways
  - Establish responsibility for reference integrity (which software element)
  - Plan in advance of development
- **We need tools to better manage component libraries, no doubt**
  - For our project, we devised a technique to get us by – until we can come up with something better
- **Determining the extent of operation consumption is tough**
  - Must know in order to effectively execute changes
  - We developed an electronic means of assisting in this effort
  - Configuration management became another fulltime job
  - More discussion on this important topic later in the presentation

Deriving Components

- **Constructed Three (3) Types of Components**
  1. Software Support
     - Reference, Audit, Report Database, Error Message, Note
  2. Core Business
     - Site, Compliance, Samples and Results, Federal Reporting, Document Tracking
  3. Business Support
     - Location, Contact, Affiliation, Staff
- **Project began with a logical data model derived from JAD sessions and an analysis of current systems**
- **Components were derived from the logical model based primarily on subject area definitions**
- **And we made-up some as we went along…**

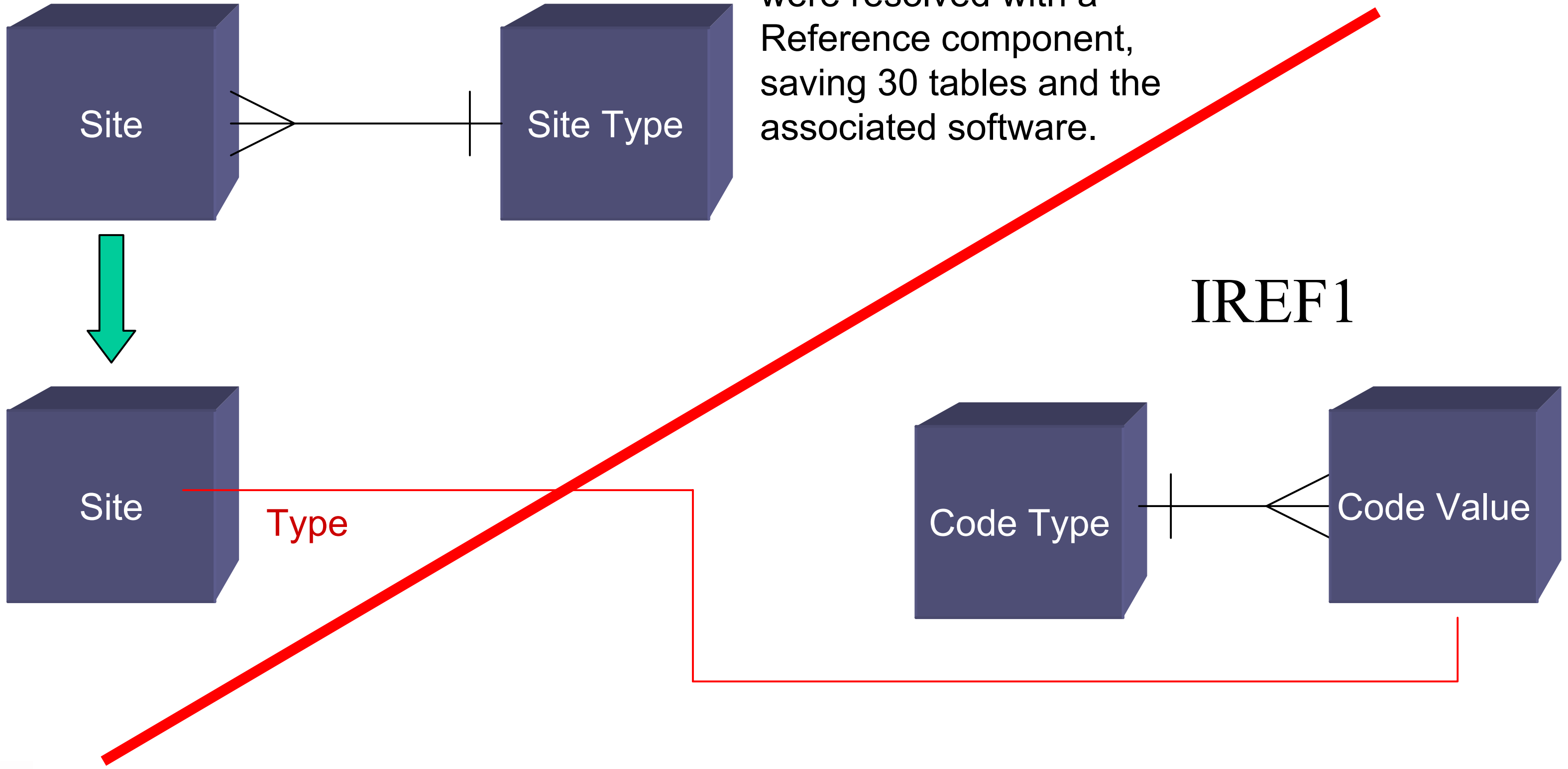
Rules for Deriving Components From Data

- **Look for candidates that can be supported by “Software Support” components.**
  - Classification entity types (Site Type, Ownership Type,…)
  - Comments, Descriptions, and Notes attributes
  - Attributes with permitted values
- **Look for candidates that can be supported by “Business Support” components.**
  - Addresses, Phone Numbers, Persons, E-mail, Companies…
- **Look for opportunities to combine functionality in “Core Business” components.**
  - Planned Events (Schedule), Correspondence (Document Tracking)
- **Minimize the number of relationships that are broken by componentization.**

Example: Data-Driven Componentization

Look for candidates that can be supported by "Software Support" components.

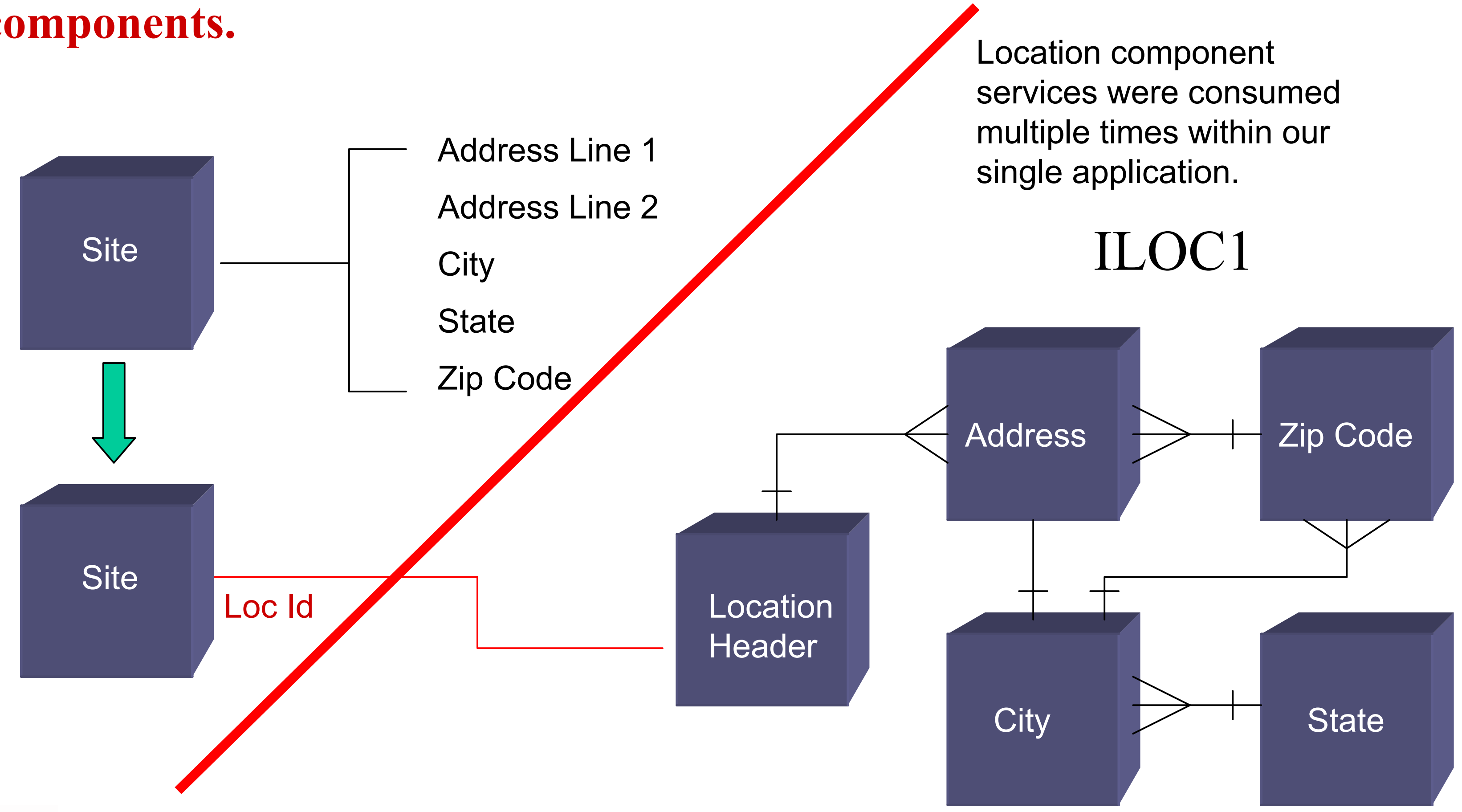
Entity Types like this were resolved with a Reference component, saving 30 tables and the associated software.



Example: Data-Driven Componentization

Look for candidates that can be supported by "Business Support" components.

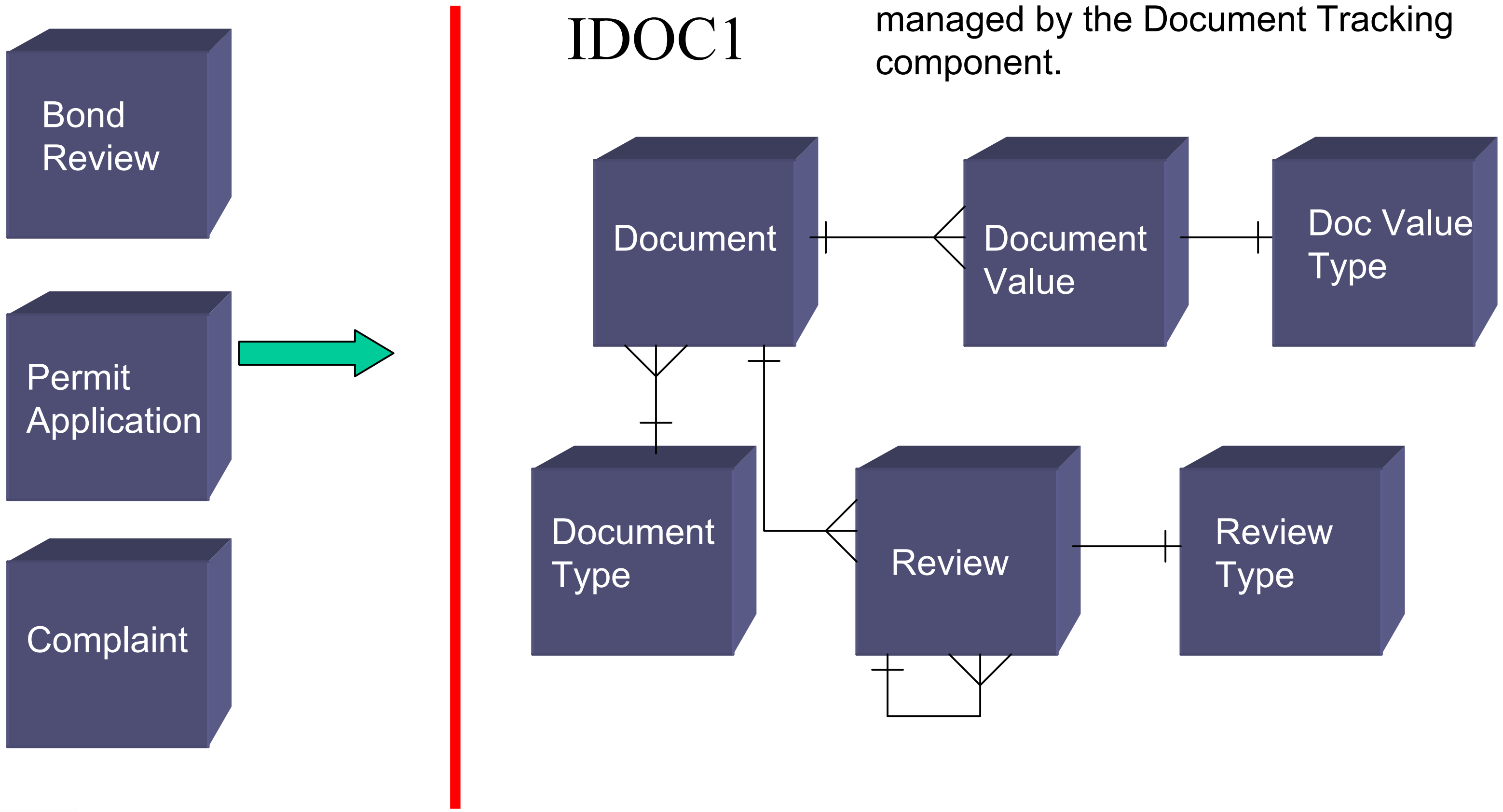
Location component services were consumed multiple times within our single application.



Example: Data-Driven Componentization

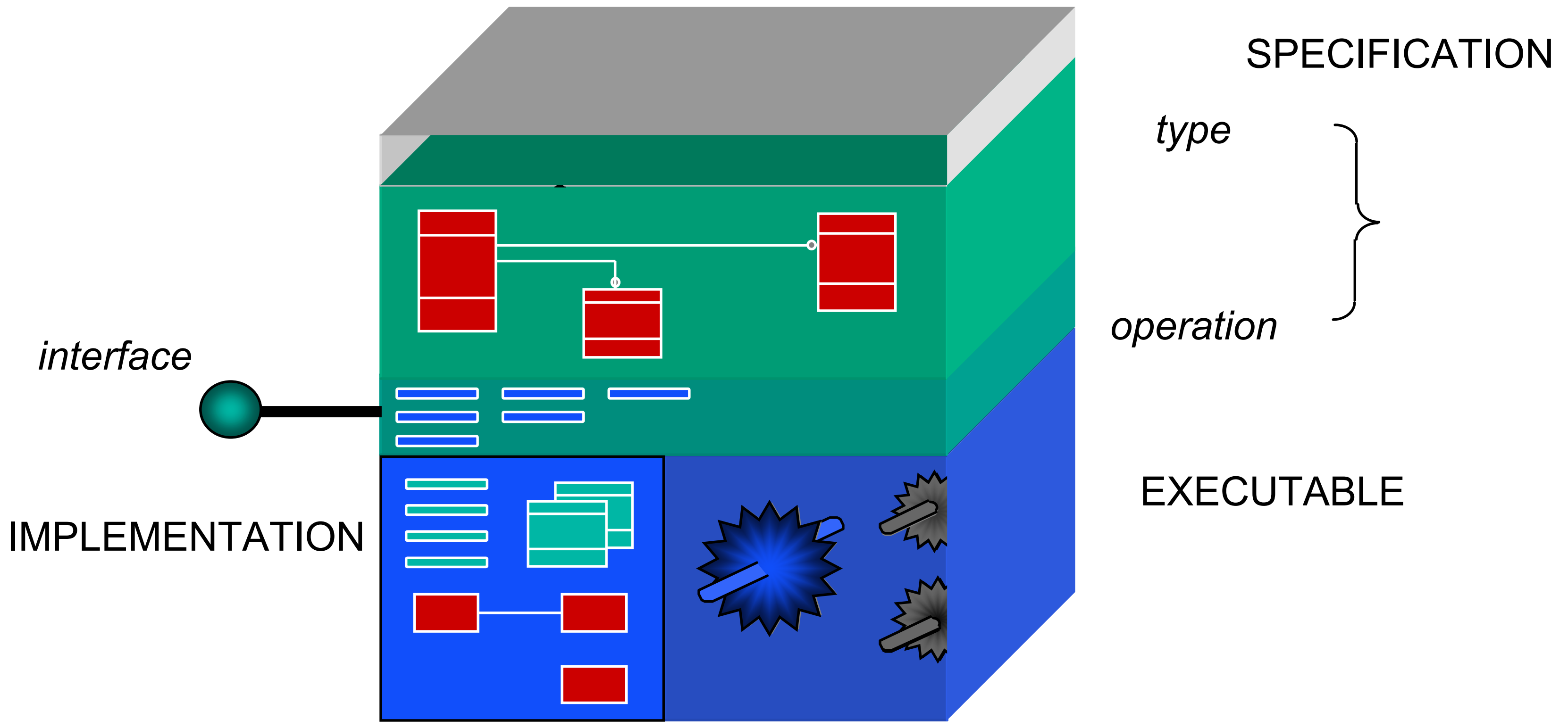
Look for opportunities to combine functionality in "Core Business" components.

Hundreds of types of correspondence are managed by the Document Tracking component.



Component Structure

Q: How to build components using COOL:Gen models?



COOL:Gen Models for CBD – What's Inside

A: By using the following models:

- **An IMPLEMENTATION model for each component**
- **A SPECIFICATION model containing the specification elements for all the components**
- **A TD model containing the implementation types and data structure for all the components**
- **An APPLICATION model containing clients, servers, and public operations of consumed components**
- **A COMMON model for generic common action blocks**
- **A REPORTS model containing software built specifically to generate reports**

Component Implementation Model

Business Systems
- Specification
- Implementation
- Test Harness
- Clients
- Servers
- For consumed ops & Common ABs

Software
- I, M, T, C Ops.
- Consumed Ops.
- Common ABs
- Clients & Servers
- Comp. “gen” svr.

Data Elements
- Specification Types
- Implementation Types & Data Structure
- Spec Types of Consumed Components

Work Sets
- Component Op Results
- Consumed Component Op Results
- Text Work, Date Work, Message Box, etc.

Exit States
- For client-to-client dialog flow & return
- For server rollback

Component Specifications Model

<u>Business Systems</u>
- Specifications

<u>Data Elements</u>
- Specification Types

<u>Software</u>
- Public "I" Ops.

<u>Work Sets</u>
- Component Op Results
- Text Work, Date Work, Message Box, etc.

Technical Design Model

Data Elements
- Implementation Types
- Data Structure

Application Model

Business Systems
- Specification(s)
- Clients
- Servers

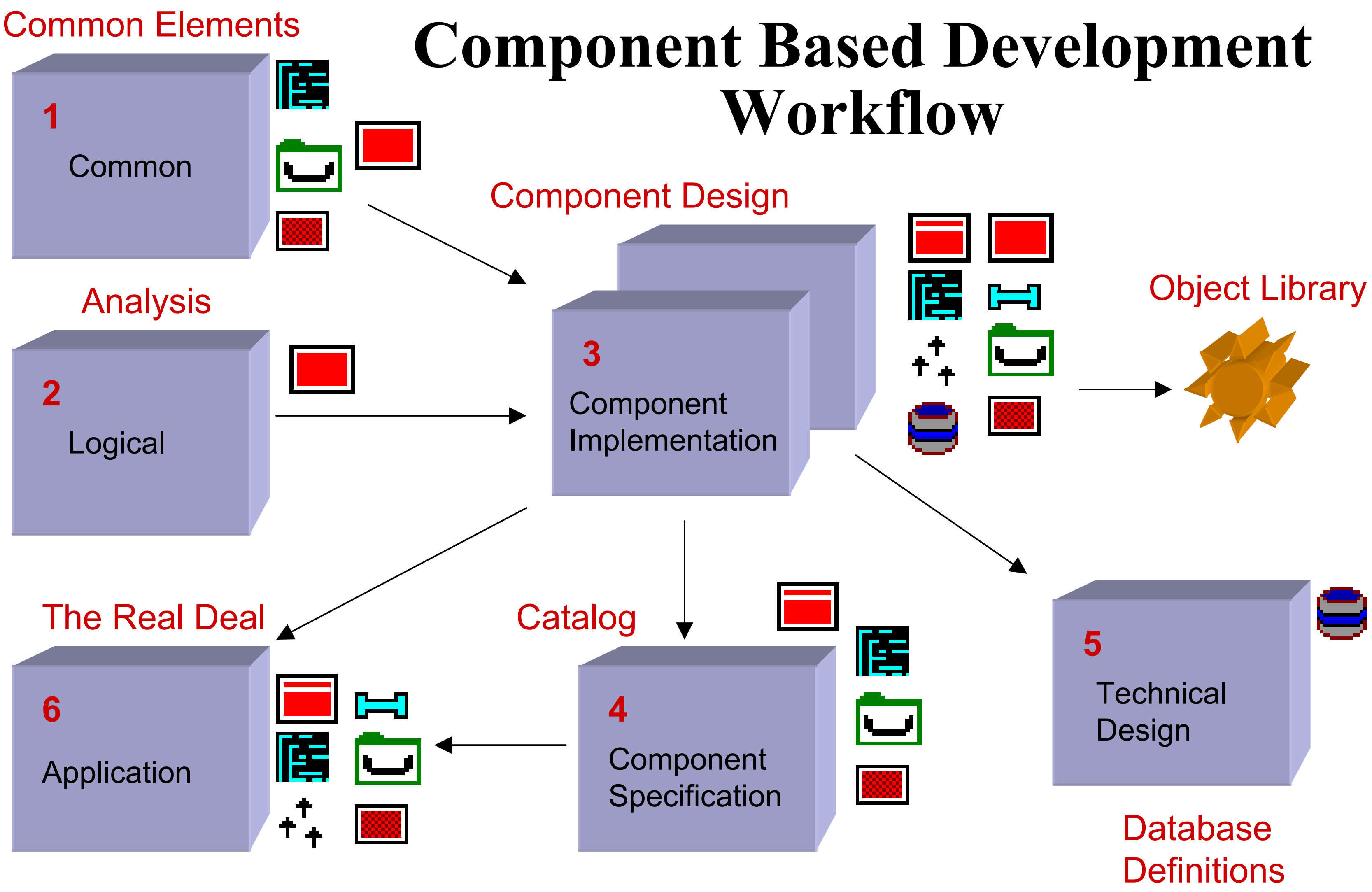
Software
- Public "I" Ops.
- Common ABs
- Clients & Servers

Data Elements
- Specification Types

Work Sets
- Component Op Results
- Text Work, Date Work, Message Box, etc.

Exit States
- For client-to-client dialog flow & return
- For server rollback

Component Based Development Workflow

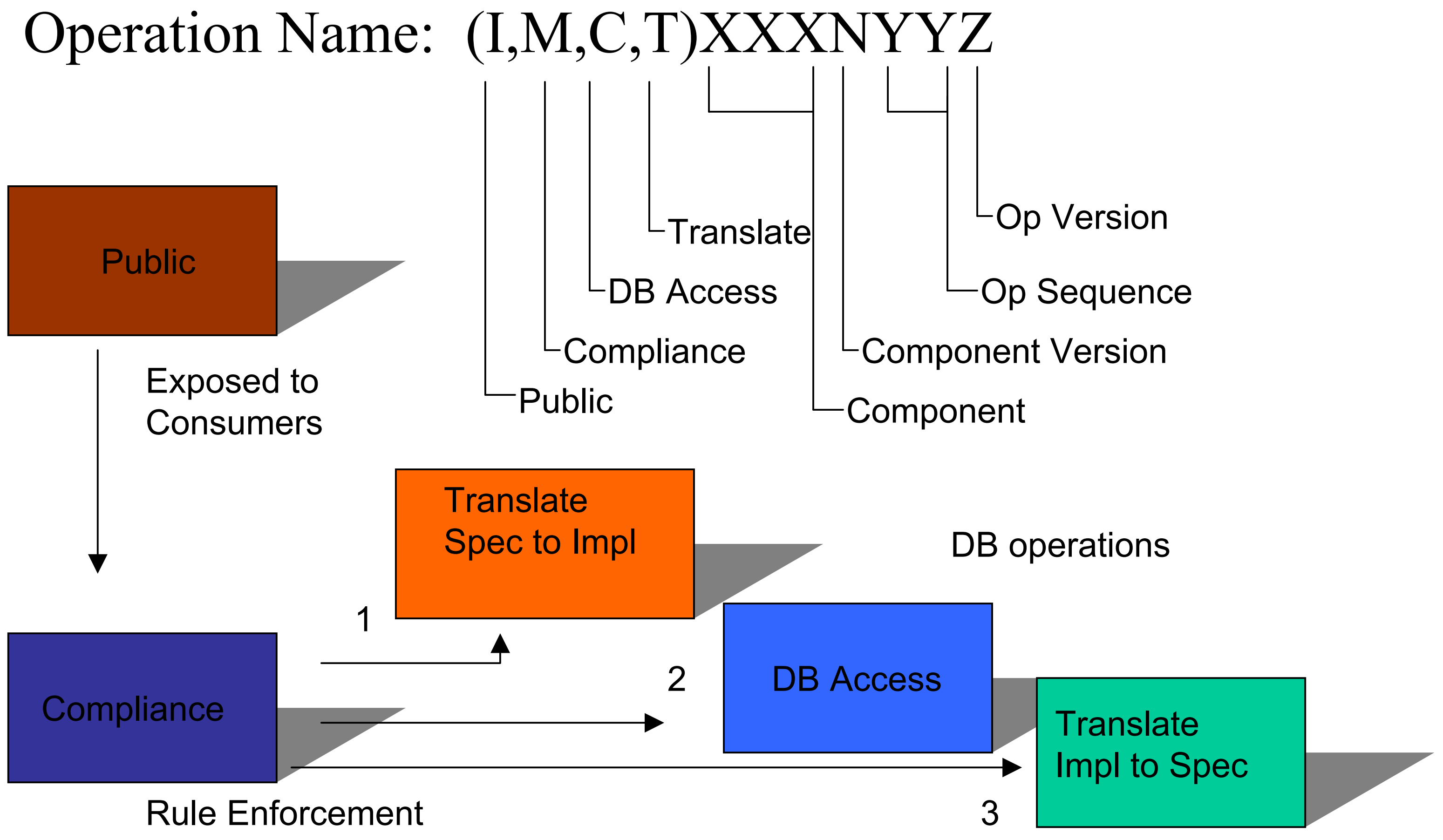


Identifying and Creating Operations

- **What can you do to a piece of information (besides ignore it)?**
  1. Create an instance
  2. Modify attributes (and foreign keys)
  3. Read an instance to examine its attributes or verify its existence (by instance identifier and by name, usually)
  4. Delete an instance (either "hard" or "soft" delete)
  5. Retrieve a sorted group of occurrences (for lists, usually)
  6. Special operations on a group of occurrences (determining average value of an attribute over multiple instances, etc.)

For each specification type, operation types 1-5 above can almost always be easily constructed. We began constructing these operation types as soon as the components were cut-out and the specification types were "stable."

The Anatomy of Component Operations



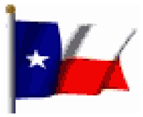
Operation Types

- **Public Operation**
  - Exposed to consumers
  - Contains specific pre- and post-condition statements of behavior
  - Uses only specification types and worksets in its views
  - Exports a standard workset to describe the results of processing
- **Internal (Compliance) Operation**
  - Enforces business rules and data integrity
  - Checks all pre-conditions except those for which database access is required
- **Internal (Translation) Operations**
  - T1 translates specification type values to the corresponding implementation type values. One T1 operation per component (usually).
  - T2 translates implementation type values to the corresponding specification type values. One T2 operation per component (usually).
- **Internal (Database Access) Operation**
  - Checks pre-conditions which require database access
  - Performs database operations (select, insert, update, delete, other)

One, Two, Three…How We Did It

Preparing the Organization

- **Prepared the organization for its first CBD project by providing a "CBD Class" to infrastructure and development managers**
  - Does everyone understand what we are doing?
  - If not, will they agree to our technique anyway?
- **Prepared step-by-step checklists for developers, along with a mechanism for estimating level-of-effort**
  - No one enjoys the opportunity of making it up as they go along.
  - Required results
- **Performed a test to validate the implementation of a component and consuming application into the agency's runtime environment**
  - Can we get it installed, and will it run?



One, Two, Three…How We Did It
Preparing the Development Team

- **Prepared IMPLEMENTATION models prior to releasing to the developer(s)**
  - Business Systems
  - Specification and Implementation Types
  - Common Action Blocks
  - Worksets
- **Provided client and server procedure templates to specify standard presentation and behavior**
- **Provided examples of each operation type (I,M,C,T) to illustrate their interaction (calling structure and view-handling)**
- **Reviewed the CBD methodology and "our way" with developers**
- **Provided "CBD 96 V2" document as a general guideline**

One, Two, Three…How We Did It
Satisfying the Quality Initiative

- Revised the Unit Test Plan format
- Negotiated compromises on presentation and structure standards to adapt them to CBD constructs and “our way”
- Designed Component Specification Document format
- Created online help
- Created design specifications
- Required the development of test harnesses to qualify operations

The Puzzle With No Straight Edges: Assembly

1. **Create and then migrate public operation stubs**
   - to component spec model
   - to consumer models
   - to the application model
2. **Migrate implementation types and data structure to TD model**
3. **Migrate servers to the application model**
4. **Migrate clients to the application model**
5. **Generate component objects and create component libraries**

The Puzzle With No Straight Edges: Assembly

6. Create application library from component library objects
7. Generate and install RI triggers
8. Generate server remote installation files from application model
9. Generate and install clients from application model
10. Install servers, deploy client executables

Some of the Results…

COMPONENT CONSUMPTION MATRIX

Consumed by

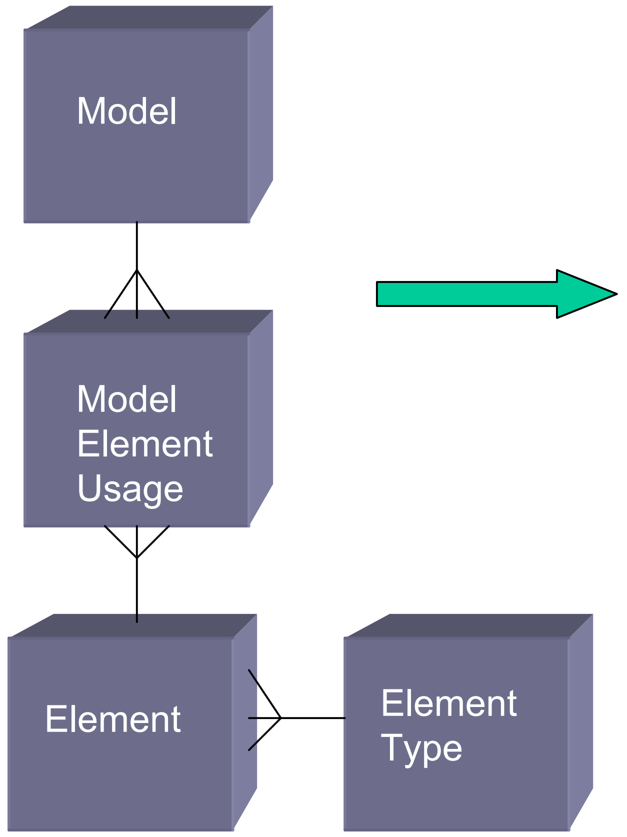
	AFL	AUD	CMP	CON	DOC	JCL	LOC	MSG	NOT	REF	RPT	SCH	SEC	SIT	SRS	STF	
AFL		X		X			X						X	X			5
AUD													X				1
CMP		X			X				X	X		X	X	X	X		8
CON	X	X											X				3
DOC		X							X	X			X	X		X	6
JCL																	0
LOC		X		X									X				3
MSG		X											X				2
NOT		X											X				2
REF		X											X				2
RPT		X								X			X				3
SCH		X								X			X				3
SEC																	0
SIT	X	X	X	X	X		X			X		X	X		X	X	10
SRS	X	X	X	X			X		X	X			X	X			9
STF	X	X		X			X			X			X				6
	4	13	2	5	2	0	4	0	3	7	0	2	14	4	2	2	

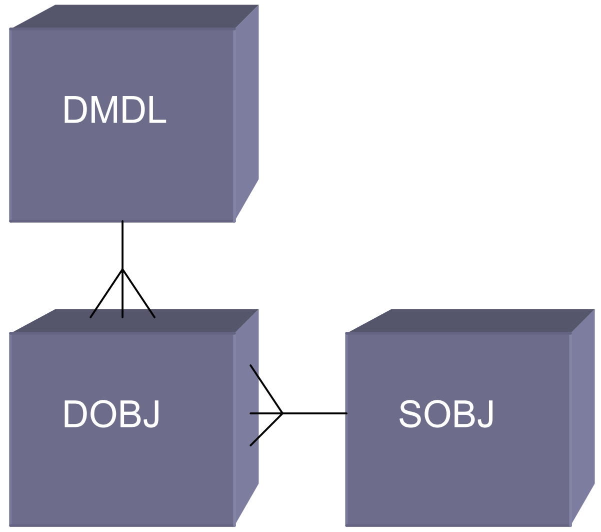
Consumes

Managing the Effects of Heavy Consumption

- **Two (2) Important Considerations**
  - Where is an element consumed?
  - What actions are required to distribute the effect of a change to an element?
- **Our Solution**
  - Created software to identify references to objects across component and application model boundaries
  - Created software to log and track change requests, the affected elements, and the implementation actions required
  - We could not manage heavy cross-consumption and rapid changes without some type of automated assistance

Finding Consumed Elements

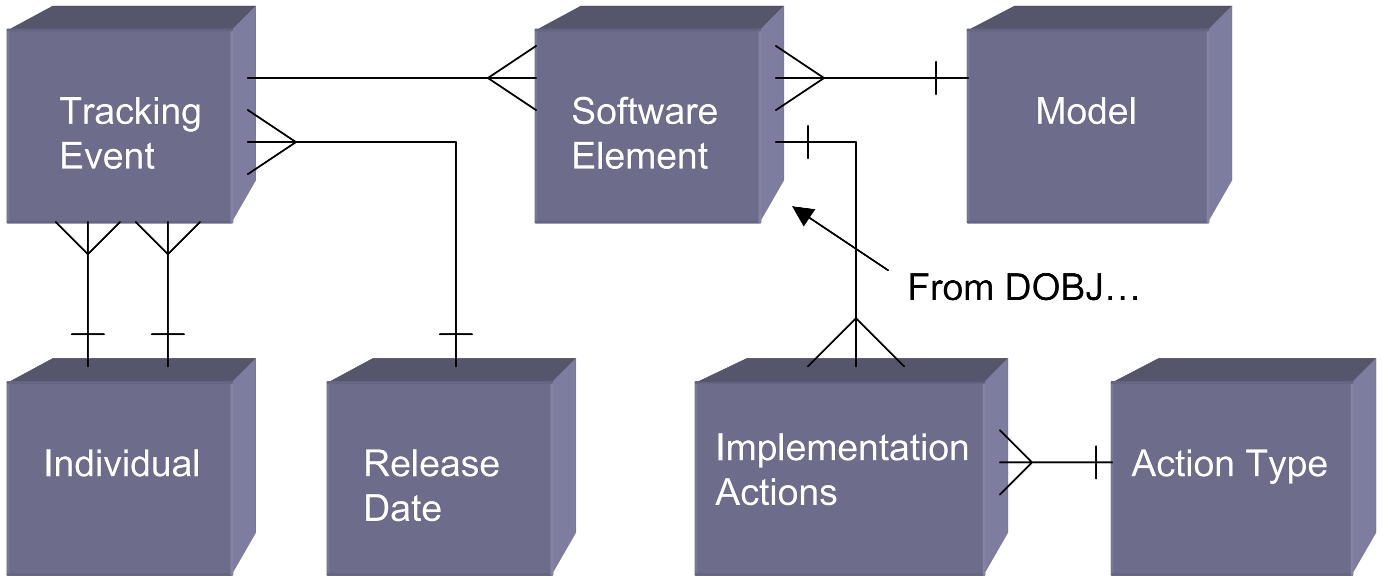




Based on Original Object Id and Schema Release

here object, object, object...

Tracking Changes and Implementation Actions



Managing Problems, problems, problems...

STERLING SOFTWARE

Summary and Discussion

- CBD is ideal for integration projects
  - Supports integration of diverse-yet-related functionality
- COOL:Gen is ideal for component-building (especially due to the benefits of the encyclopedia)
- CBD? I would definitely do it again – with a few modifications
  - Clearly prescribe the techniques to the development staff
  - Create an overall roadmap beforehand
  - Need to know exactly what/when/how
- Would like the process to be more "automatic"
  - Select from component "bookshelf" rather than build
- Need to speak the same component language across projects
  - Operation, Spec Type, Impl Type,…
- Need to improve configuration management
  - With better tools
  - With better techniques