# Defining Software Component Specifications: An Open Approach

Kenneth Klein
Computer Sciences Corporation




**NDIA Systems Engineering Conference**
**October 22-26, 2007**

EXPERIENCE. RESULTS.



# A Couple Definitions

- Open
  - Based on widely excepted and supported standards
  - Defines key interfaces using these standards
  - Not proprietary

- Software Component
  - A modular part of a software design that hides its implementation behind a set of external interfaces.
  - Within a system, components satisfying the same interfaces may be substituted freely.

- That's what the terms mean in the context of these slides….



## The Problem

- Given an “as-built” component-based Department of Defense (DoD) software system
  - Code written in Java
  - Interface-based component services
- Needed an approach to documenting each component as a as a set of well-defined interfaces
  - Required to meet DoD “openness” standards
  - Critical for making components extendible and reusable

The problem has a problem…



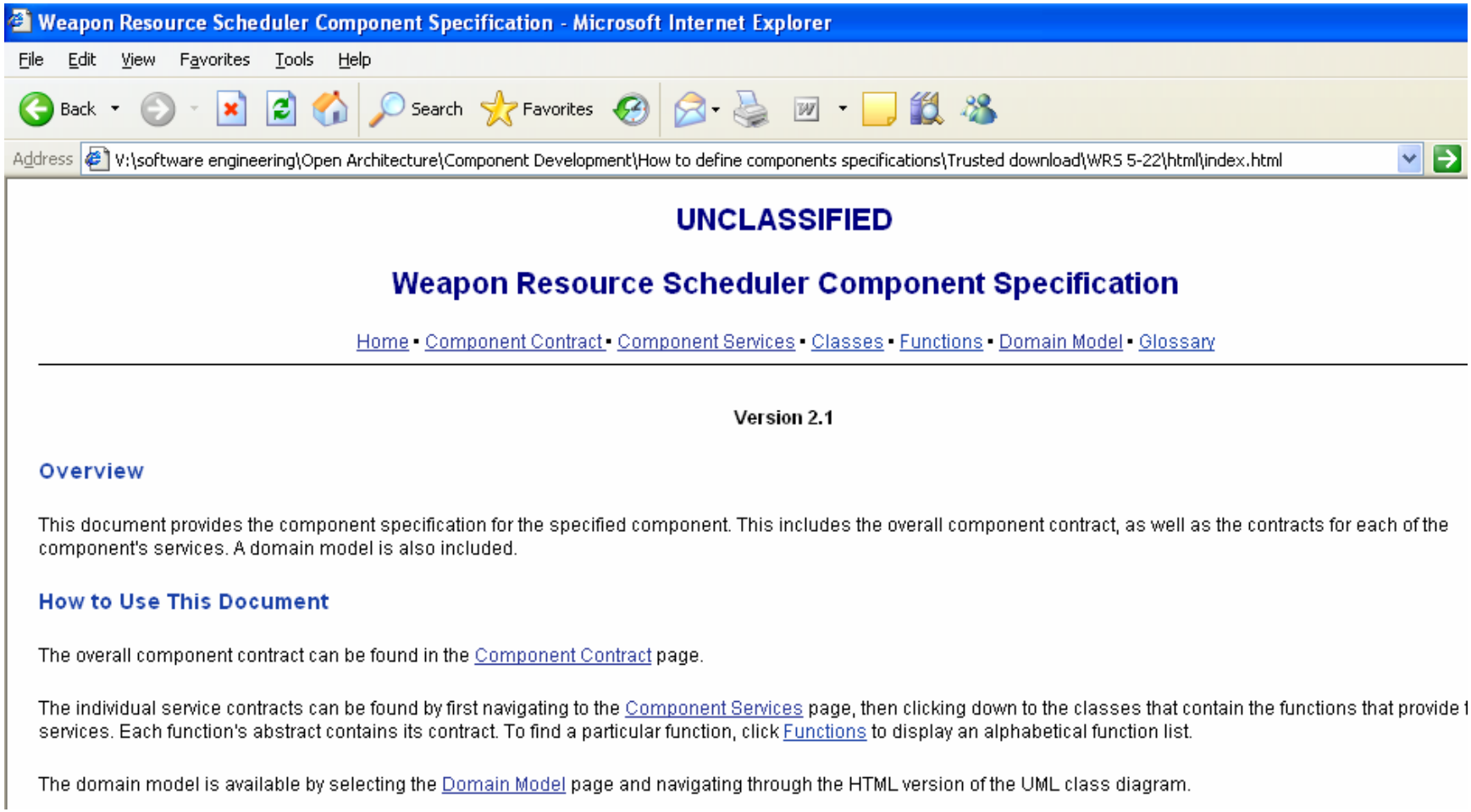
## *Well-Defined* is not Well-Defined

- A lot of literature available on defining:
  - Information exchange standards, e.g., CORBA, JMS, DDS
  - Specific implementations of these standards
  - Component frameworks, e.g., SOA, EJB
  - Quality of Service requirements
- Not so much out there on defining a service's functional behavior

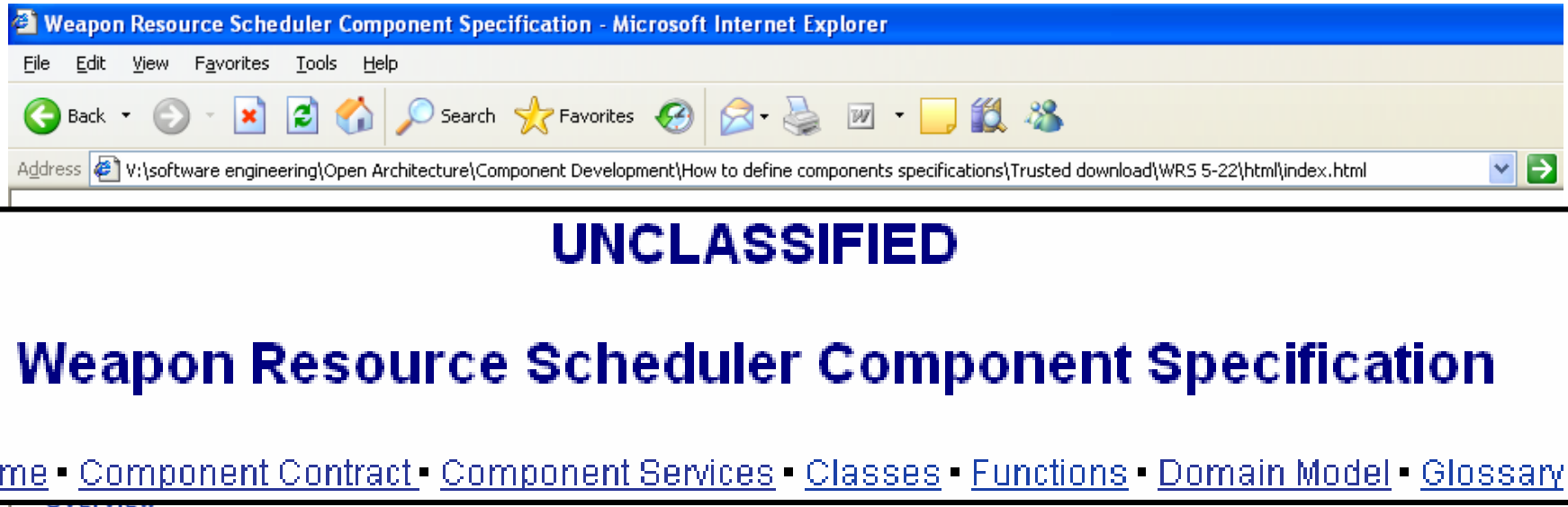
# The Solution

- Define the component and its services using:
  - Lightweight UML domain modeling
  - Design by Contract (DbC) principles
- Tools used
  - A UML modeling tool that can generate HTML output
  - Doxygen
    - Open source C++/Java documentation generation tool
      - Similar to Javadoc
        - » Recognizes Javadoc comment delimiters
    - Reads source code, generates HTML
    - www.doxygen.org
  - A web browser

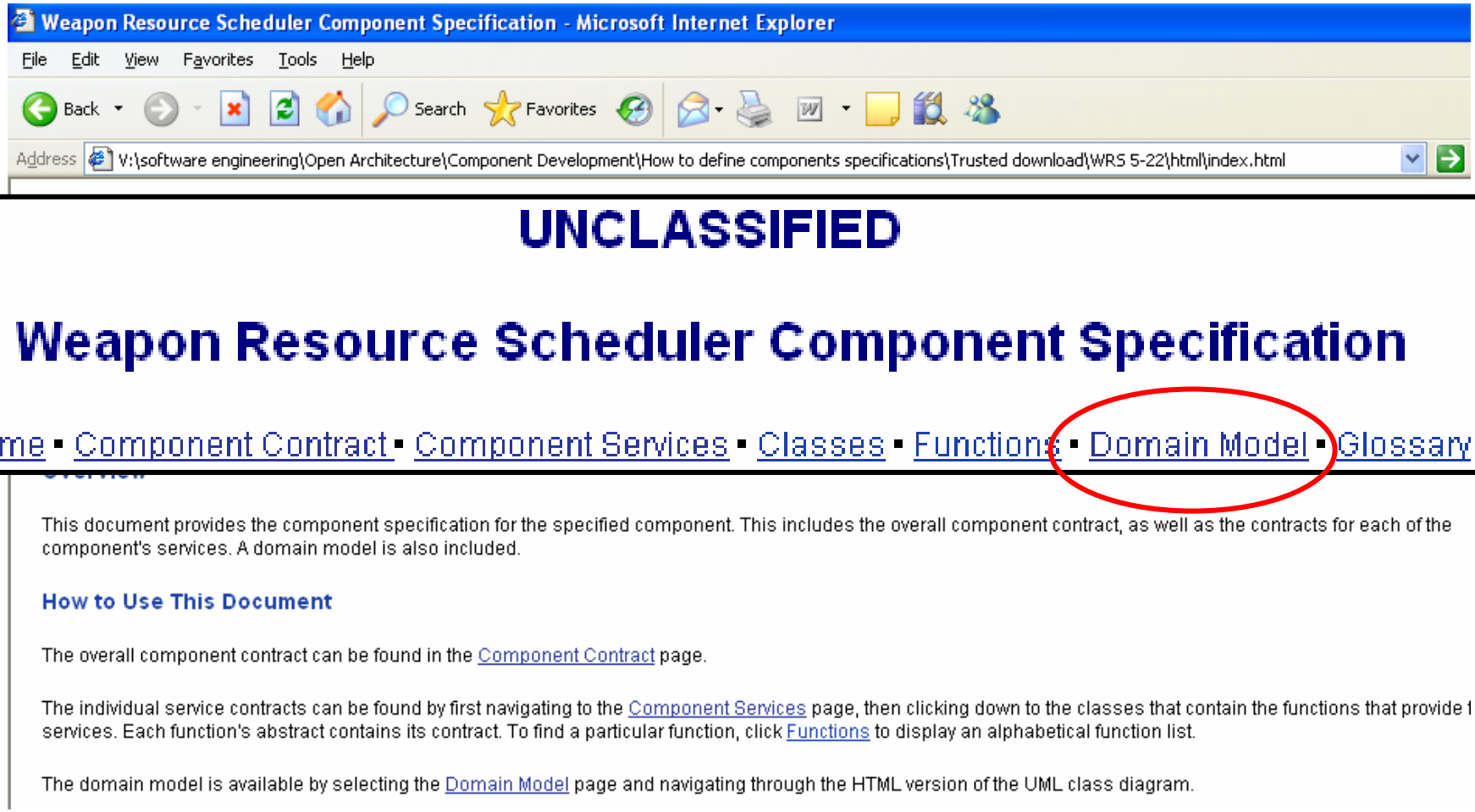
# Navigating the Spec



# Navigating the Spec



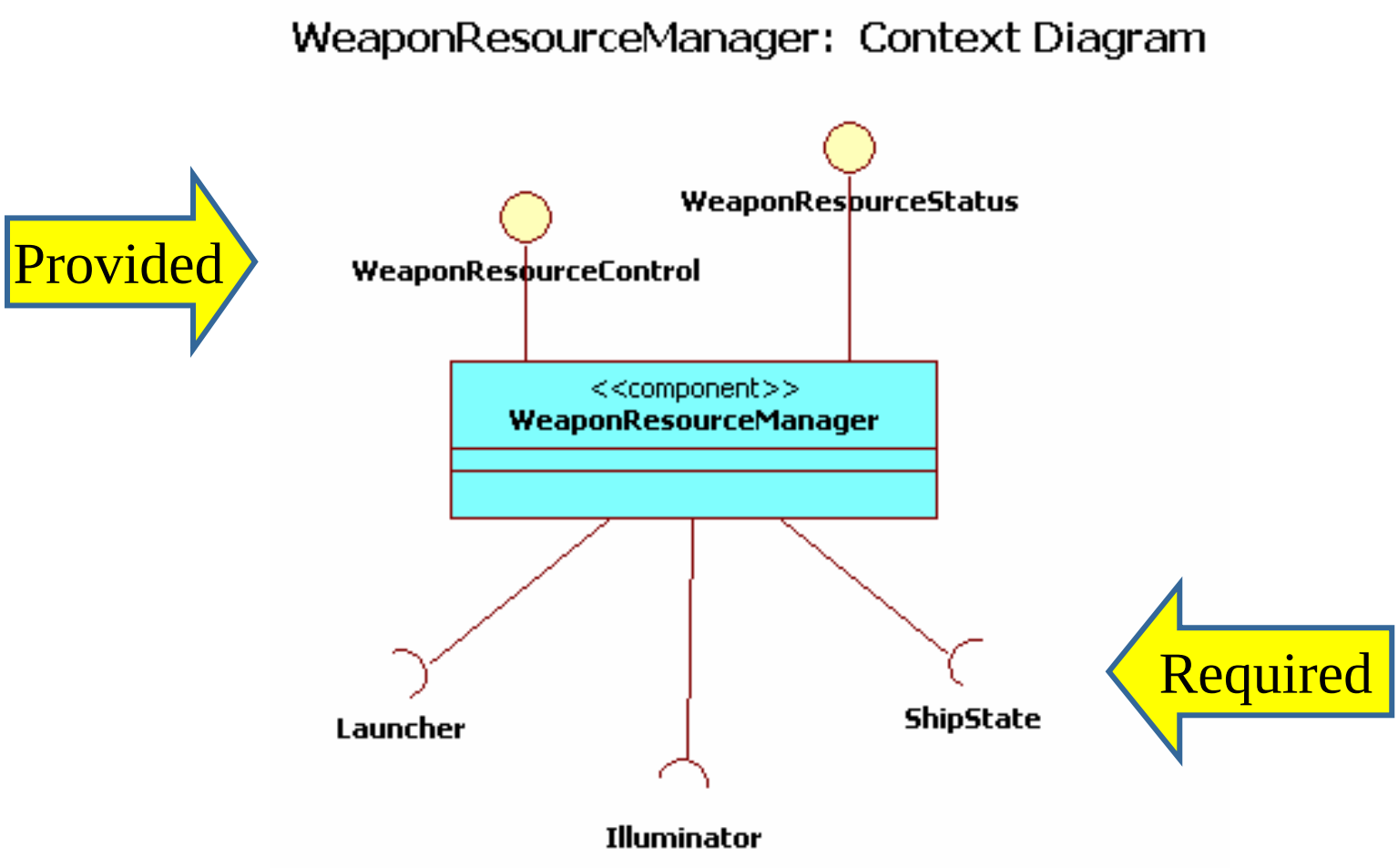
# Domain Model: The Context Diagram



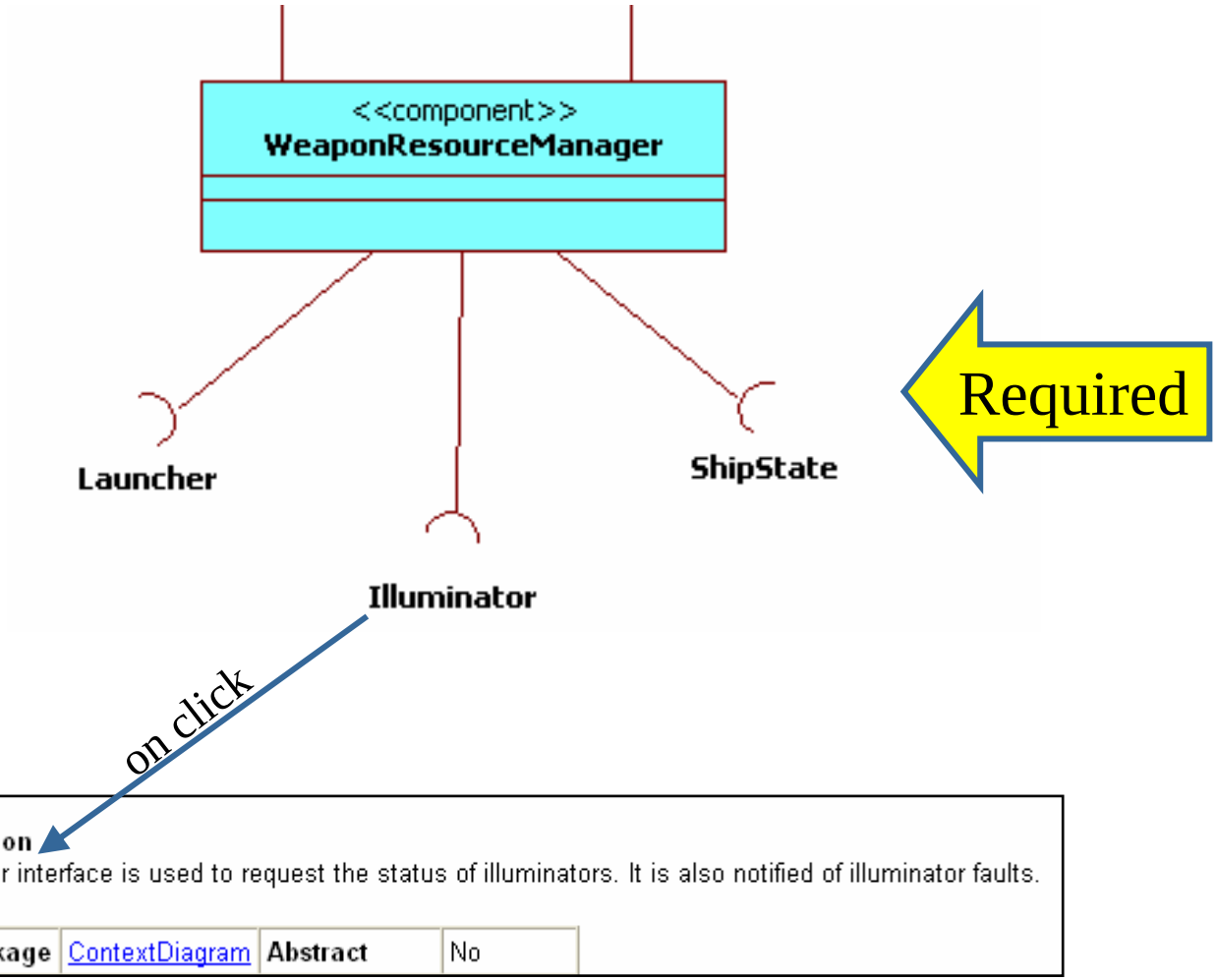
# Context Diagram

- Shows component's *provided* and *required* interfaces
  - Provided interface declares services that this component offers to external components
  - Required interface declares services that this component requires from external components
- Describes required interfaces in context of this component
  - Each component may describe the same required interface differently based on the component's needs
  - E.g., given an Illuminator interface, one client may *require it to check Illuminator equipment status*; another client may *require it to Illuminate a target*

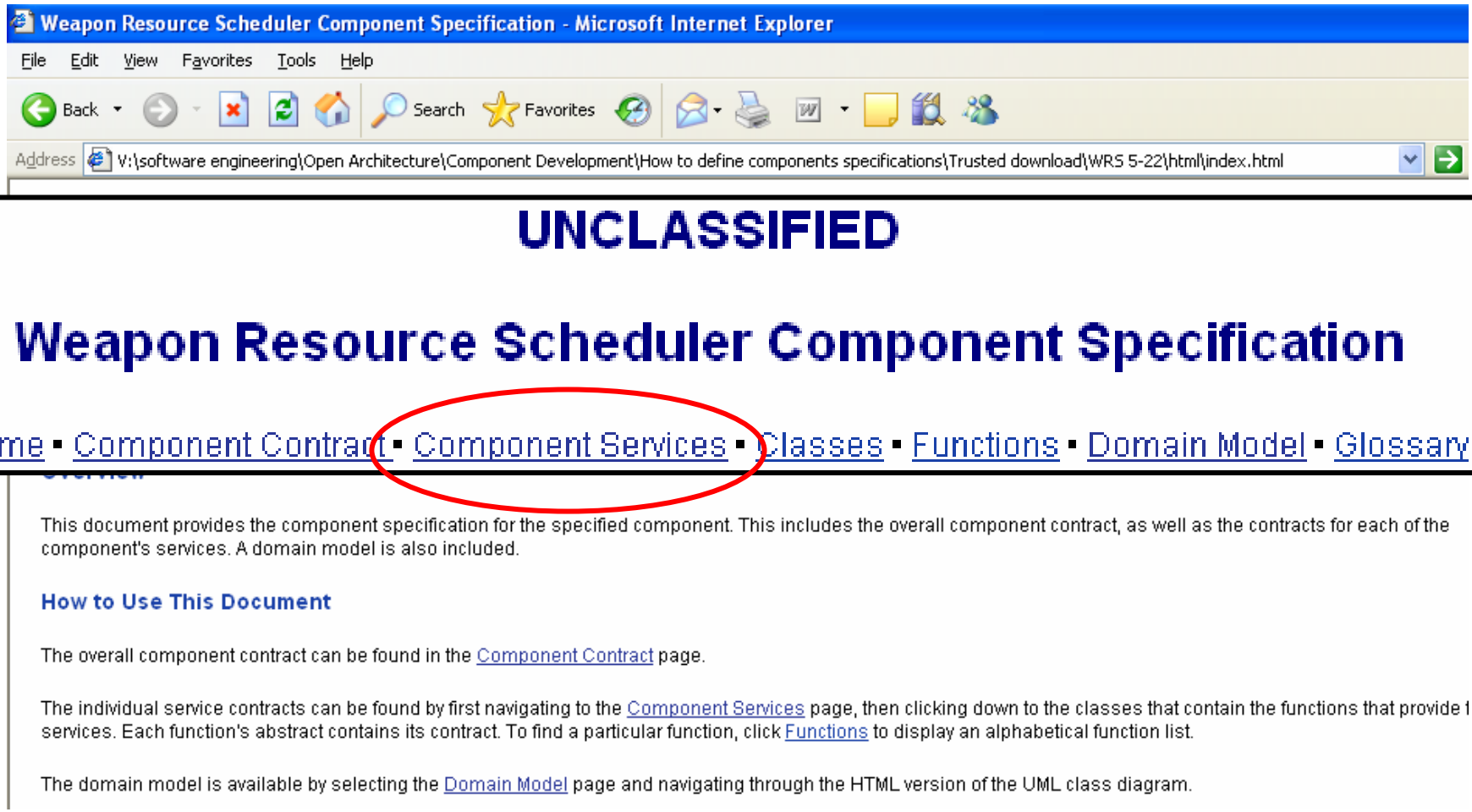
# Context Diagram Example



# Context Diagram Example (cont.)



# Component Services Example



## Component Services

- What is expected of the client?
- What does the service do?
- State all this with as few implementation details as possible
  - Most implementation choices should not impact the specification
  - More likely specification will remain unclassified
- **Design by Contract provides a solution**

# What is Design by Contract (DbC)?

- Defines the *contract* between the *interface* and its *clients*
- Preconditions
  - States that must be true when service is invoked
- Postconditions
  - If service is invoked when preconditions are true, postconditions describe guaranteed outcome
    - e.g., state changes, messages sent
- Invariants
  - Attribute constraints that must always be true:
    - After component instantiation
    - Before/after each service invocation
    - e.g., An Engagement must have exactly one Target
- Exceptions
  - Describes what happens when preconditions or invariants are violated or postconditions cannot be met
  - Behavior can be “undefined”

# Why DbC?

- Well documented, mature paradigm
  - Term coined by Bertrand Meyer in 1997
  - Since then, large volume of literature written on the topic
    - See resource list
- Decoupled from implementation details
  - Guidelines for “just enough” information
  - Implementation can change without impacting contract
- Encourages discussions that may otherwise never occur
  - Provides common vocabulary for complex concepts
  - Exceptions often discovered when writing contracts

# Why DbC? (cont.)

- Facilitates Liskov Substitutability Principle (LSP)
  - Service implementations/extensions *must not add preconditions or remove postconditions*
- Supports:
  - Maintainability/Extendibility/Reusability

# DbC Note: Preconditions and Callbacks

- Callback services should not change the state that triggered the callback
  - Remaining observers will receive incorrect notifications
    - Subject component has a list of color observers
    - Subject reports “I just turned red”
    - One of the observers changes subject to blue
    - The remaining observers will incorrectly be notified that subject is red
- Mitigation
  - Subject component keeps track of whether a callback is in progress
  - Any offered service that could change an observed state *has a precondition* that notifications are *not* in progress
    - Above observer’s attempt to make subject blue would be rejected

# DbC Note: Maintaining the Invariants

- Exceptional service termination must restore component invariants
  - Otherwise, component is not stable, so its services' behavior is undefined
  - May be criteria for invoking recovery path
- Concurrency should only be allowed for services that can guarantee that preemption can only occur while the component invariants are in place
  - Mitigated by many concurrency oriented architecture and design patterns
    - See *Pattern-Oriented Software Architecture Vol. 2: Patterns for Concurrent and Networked Objects*

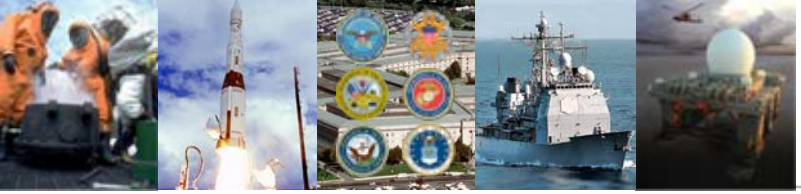
# Component Service Contracts

- Method signature
  - Captured as-is from source code
- Preconditions/Postconditions/Exceptions
- Query or Command
  - Does the service change the parameters' or the component's state?
- Parameters
  - Constraints
    - E.g., valid ranges, precision, units
  - Is ownership transferred?
    - If "no," client must be notified of any state changes
  - Type Definitions
    - Imported as-is from source code
    - Linked via hypertext



# Component Service Contracts (cont.)

- Quality of Service
  - Performance, throughput, blocking, availability
- Is concurrency allowed?





# Service Contracts as Comments in the Code

- From *The Mythical Man Month* (M3), pg. 169 [Fred Brooks, 1995] (first edition published in 1975)
  - We typically attempt to maintain a machine-readable form of a program and *an independent* set of human-readable documentation, consistent of prose and flow charts.
  - The results in fact confirm our teachings about *the folly of separate files*. Program documentation is notoriously poor, and *its maintenance is worse*.
  - **The solution, I think, is to merge the files, to incorporate the documentation in the source program.**

- This is what Doxygen does…



# Component Services Example: Java Code

```
/**
 * DESCRIPTION:
 * <p>
 * This method will distribute the request to the WeaponResourceManager.
 * <p>
 * @param[in] request  The request being sent to the WeaponResourceManager.
 *       -# Valid ranges
 *               - Not null
 * @par Query or Command:
 *       Command
 * @pre
 *       -# None.
 * @post
 *       -# If the request is an Alpha Request, it was added to the
 *          Illuminator Schedule.

...
*/
public void setRequest( RequestIF request );
```



# Component Services Example: Doxygen Output

**void WeaponResourceManager::setRequest ( RequestIF *request* )**

DESCRIPTION:

This method will distribute the request to the **WeaponResourceManager**.

**Parameters:**

`[in]` *request* The request being sent to the **WeaponResourceManager**.

1. Valid ranges
   - Not null
2. Ownership transferred (Y/N)? Y

**Query or Command:**

Command

**Precondition:**

1. None.

**Postcondition:**

1. If the request is an Alpha Request, it was added to the Illuminator Schedule. This processing includes the use of the Illuminator interface to acquire equipment status.
2. If the request is a Beta Request, it was added to the Launcher Schedule. This processing includes the use of the Launcher interface to acquire equipment status.

**Exceptions:**

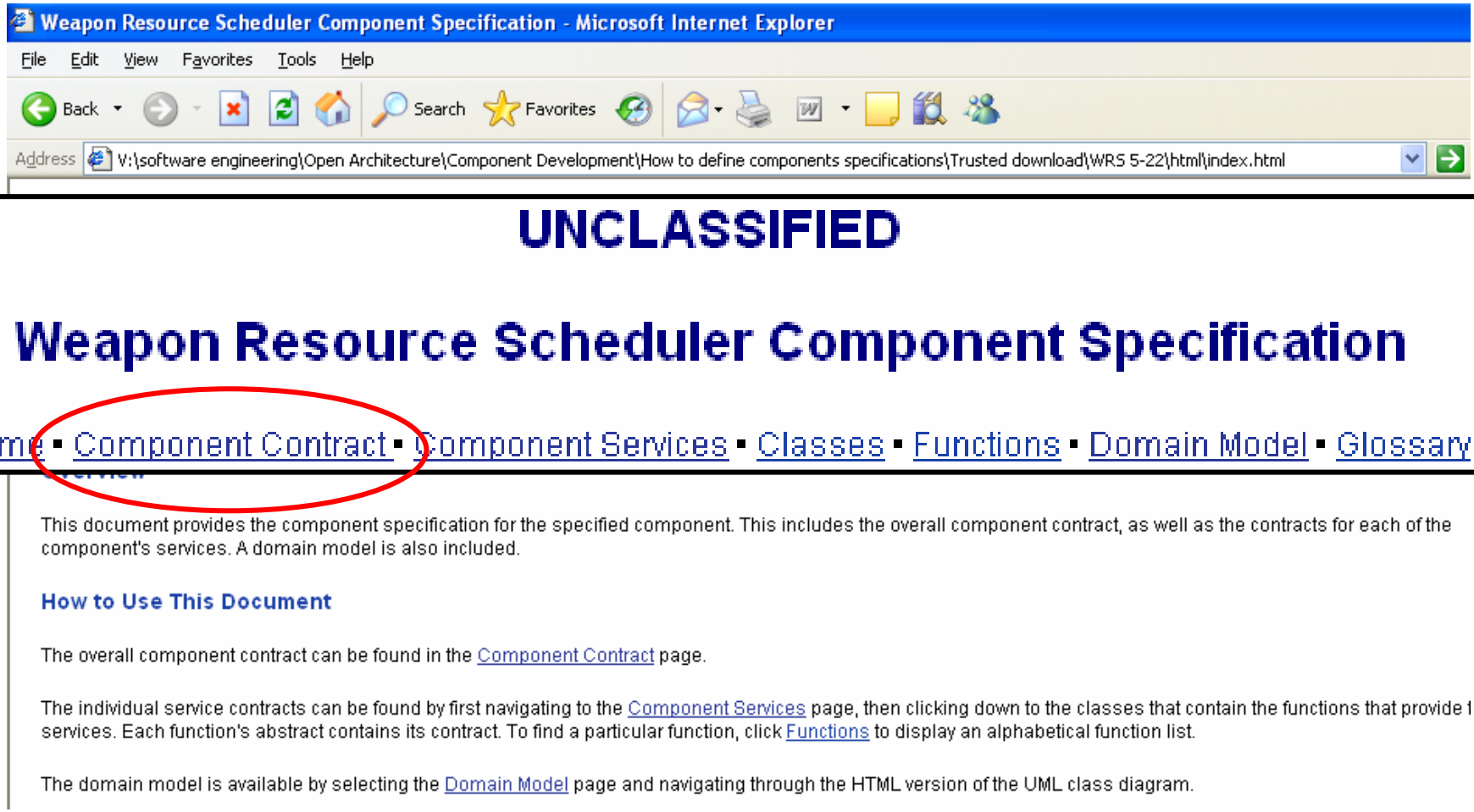
*NullRequest* The request is null. Behavior undefined.

**Blocking (Y/N):**

N

Extracted from code comment block

# Component Contract Example





## Component Contracts

- Preconditions/Postconditions
  - Before and after component startup, respectively
- Invariants
  - Applicable to component as a whole
    - Each is a pre & post condition for every service
- Exceptions
  - If pre/post conditions or invariants violated
- “Full load” memory requirements
- Proven hardware platform and OS support
- Communication standards and implementations
  - E.g., JMS/Websphere, DDS/NDDS 4.0, CORBA/ACE TAO
- Other Protocols/Standards
  - POSIX, SNMP, .NET, etc.



## Component Contracts (cont.)

- Programming Languages
- Configuration file dependencies
- Availability requirements, e.g., MTBF



# Component Contract Example

## WeaponResourceManager Component Contract

**Precondition:**

The AlphaProperties configuration file must exist.

The initial state must be provided.

**Postcondition:**

The WeaponResourceManager was started and the initial state was set using the provided parameter.

The WeaponResourceManager applied the states found in the AlphaProperties file.

Communications were established with the Illuminator, Ship and Launcher components.

**Invariant:**

Communication must be maintained with the Illuminator, Ship and Launcher components.

**Exceptions:**

| | |
|---|---|
| *MissingPropertiesFile* | The AlphaProperties file does not exist. Error is logged and program exits. |
| *InvalidProperties* | Properties in the AlphaProperties file were missing or invalid. The properties are set to their default values. |
| *RequiredComponentsUnavailable* | Unable to communicate with the Illuminator, Ship or Launcher. Behavior undefined. |

**Standard Used for Invoking Services**

JMS/Tibco 4.0.0

**Language Dependencies**

Java 5.0

More below…



# Contract Ambiguity Problem

- From *M3* pp. 63-64
  - Human language is not naturally a precision instrument for [specification] definitions.
  - Formal definitions are precise. What they lack is comprehensibility.
  - **I think we will see future specifications to consist of both a formal definition and a prose definition.**

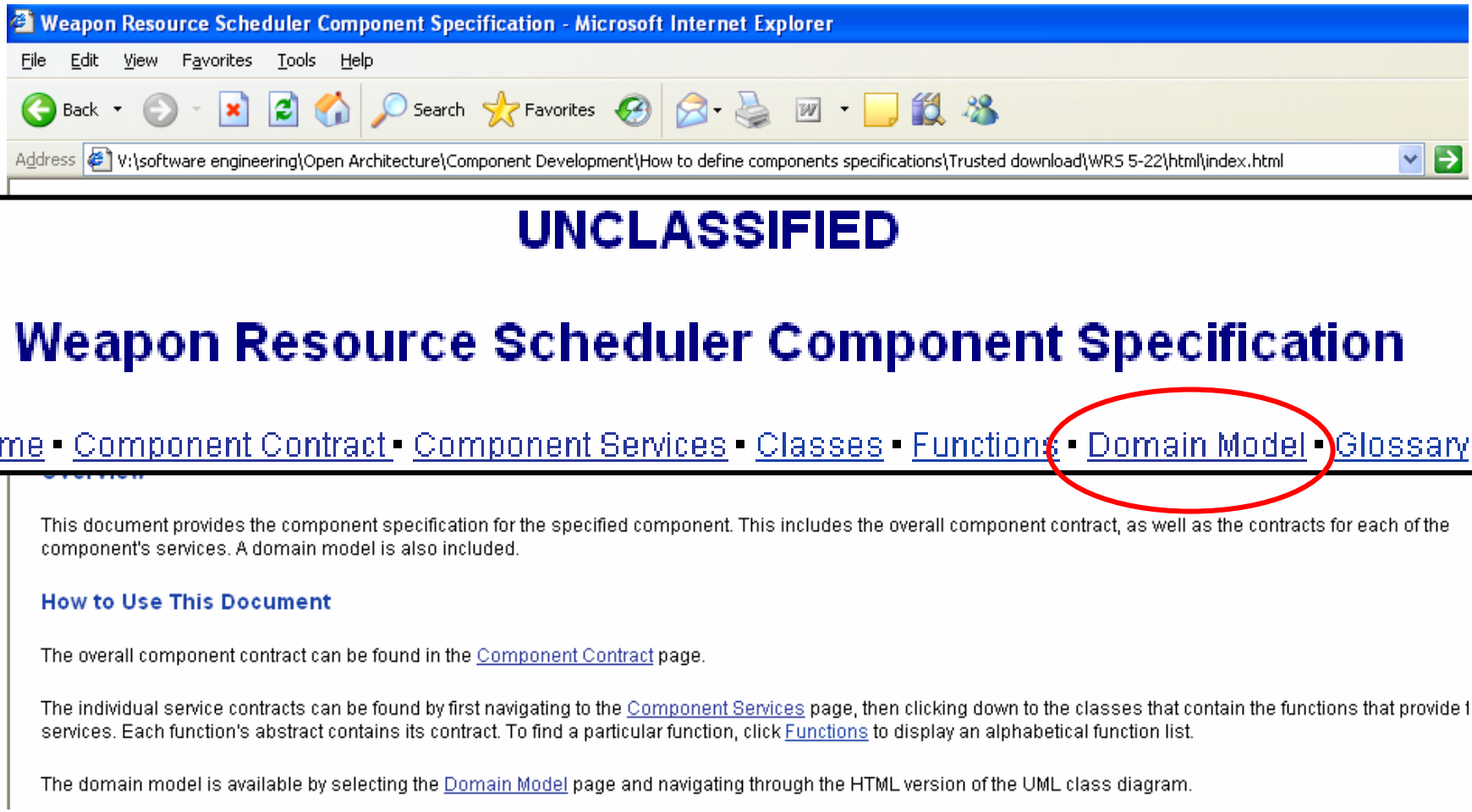
- This is what the Domain Model does…



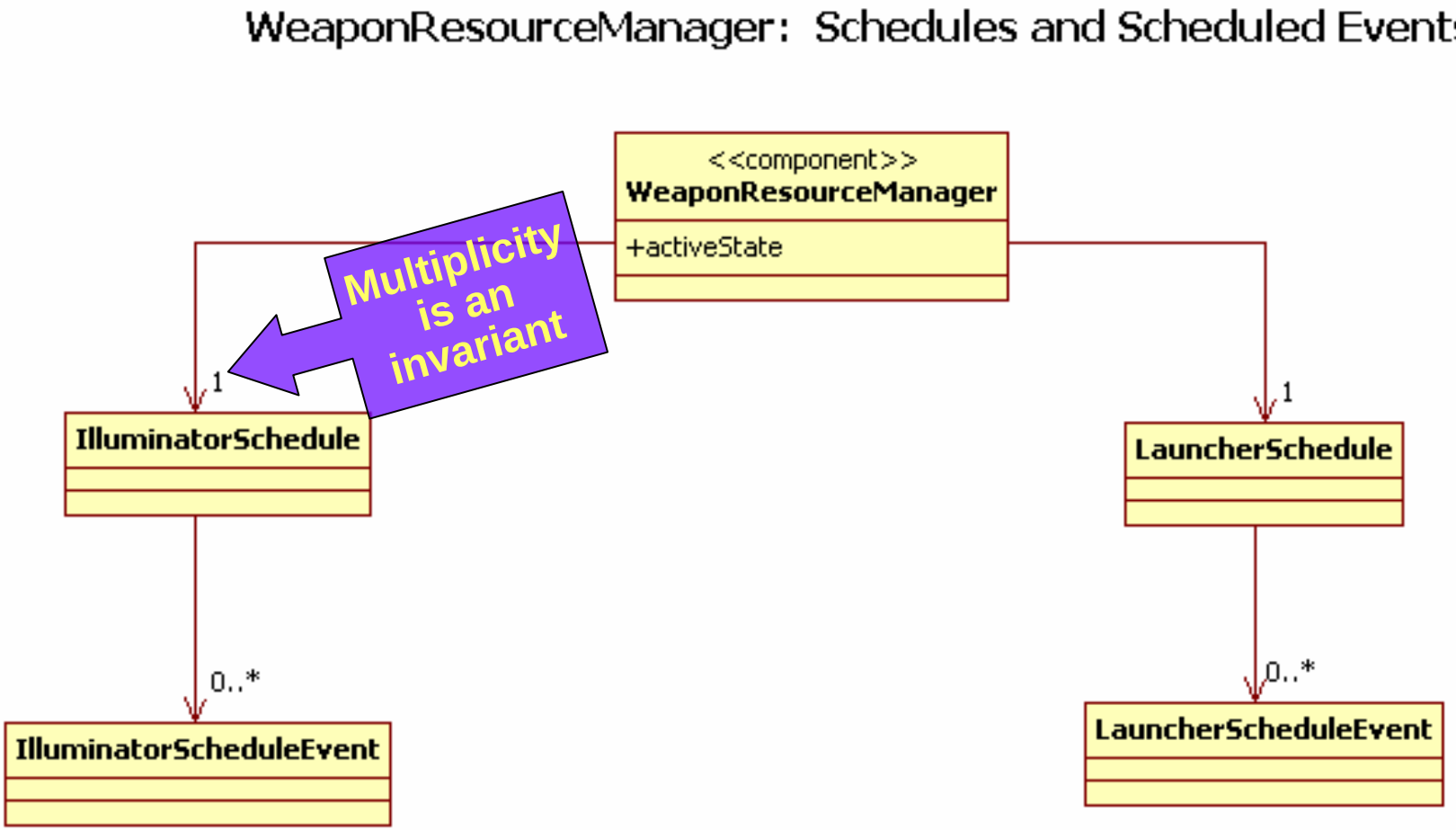
# Contract Ambiguity Solution: The Domain Model

- Provides formality of UML
  - Each domain class is clearly defined in the model
- Contracts reference domain classes in plain English
- What is a domain class?
  - Real-situation *notional* class in a domain, e.g., Launcher Schedule, Target, etc.
  - They are not actual software implementation classes
- Why not implementation classes?
  - *M3* says (pg. 175): "If one uses only a *highest-level structure graph*, it might safely be kept as a separate document, for *it is not subject to frequent change*."
  - Notional domain classes are stable, because they are decoupled from implementation details

# Domain Model Example



# Domain Model Example

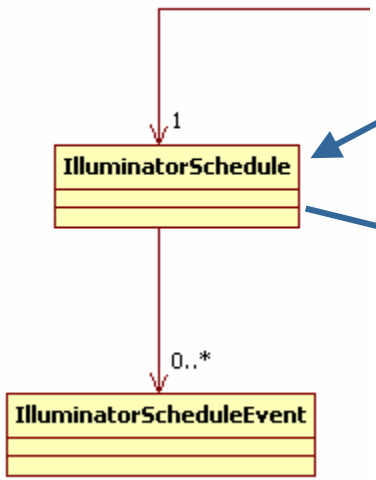


# How Domain Model Relates to Contracts

Contract

**Postcondition:**

1. If the request is an Alpha Request, it was added to the Illuminator Schedule. ' to acquire equipment status.
2. If the request is a Beta Request, it was added to the Launcher Schedule. Thi acquire equipment status.




on click

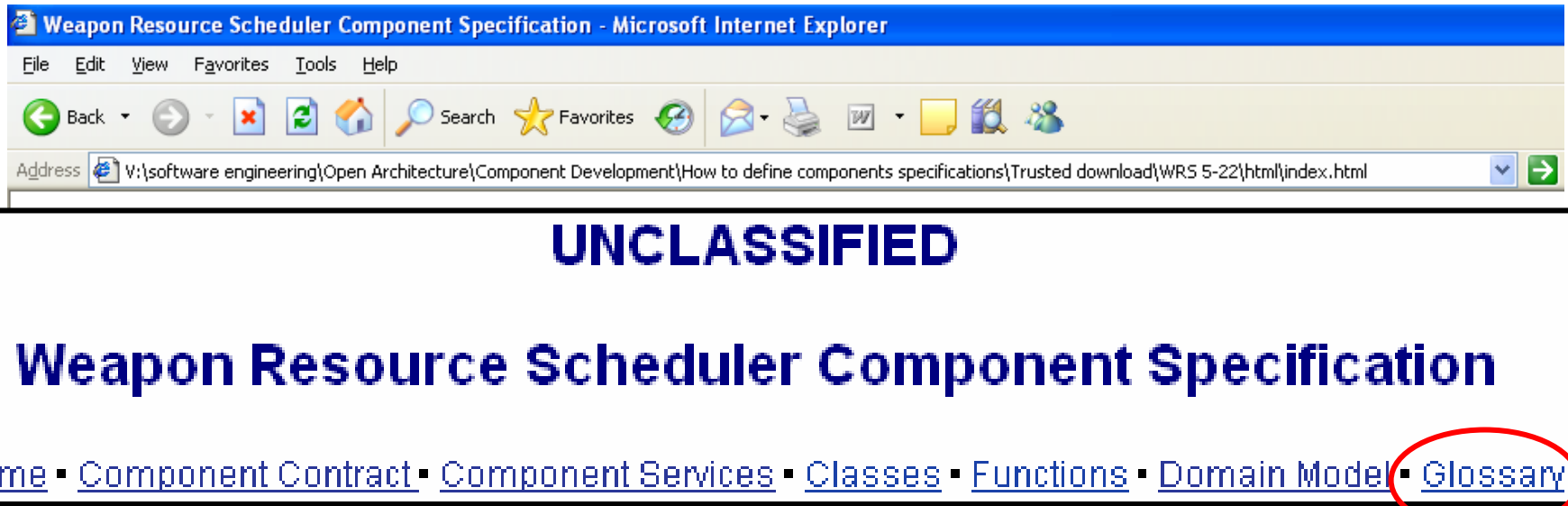
**Class IlluminatorSchedule {Analysis}**

**Documentation**
The IlluminatorSchedule contains all of the current illuminator reservations.

| Parent Package | DomainObjects | Abstract | No |
|---|---|---|---|



# Glossary


Weapon Resource Scheduler Component Specification - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address V:\software engineering\Open Architecture\Component Development\How to define components specifications\Trusted download\WRS 5-22\html\index.html
UNCLASSIFIED
Weapon Resource Scheduler Component Specification
Home ▪ Component Contract ▪ Component Services ▪ Classes ▪ Functions ▪ Domain Model ▪ Glossary
This document provides the component specification for the specified component. This includes the overall component contract, as well as the contracts for each of the component's services. A domain model is also included.
How to Use This Document
The overall component contract can be found in the Component Contract page.
The individual service contracts can be found by first navigating to the Component Services page, then clicking down to the classes that contain the functions that provide t
services. Each function's abstract contains its contract. To find a particular function, click Functions to display an alphabetical function list.
The domain model is available by selecting the Domain Model page and navigating through the HTML version of the UML class diagram.



# Glossary Excerpt

## Component Contracts

Component contracts apply to the component as a whole. They should not be confused with **Component Service Contracts** which apply to particular services.

**Precondition:**

System and/or environment states required in order to successfully instantiate the component, e.g., config files must exist.

**Postcondition:**

Component states that exist upon completion of component instantiation and initialization. For OA, this includes completion of the component's start method. Examples of component level postconditions include:

- Indicating if the component was placed in active or standby state
- Listing which config files were read
- Stating that communications with other specific components were initialized
- Listing which subscriptions and publications were defined

**Invariant:**

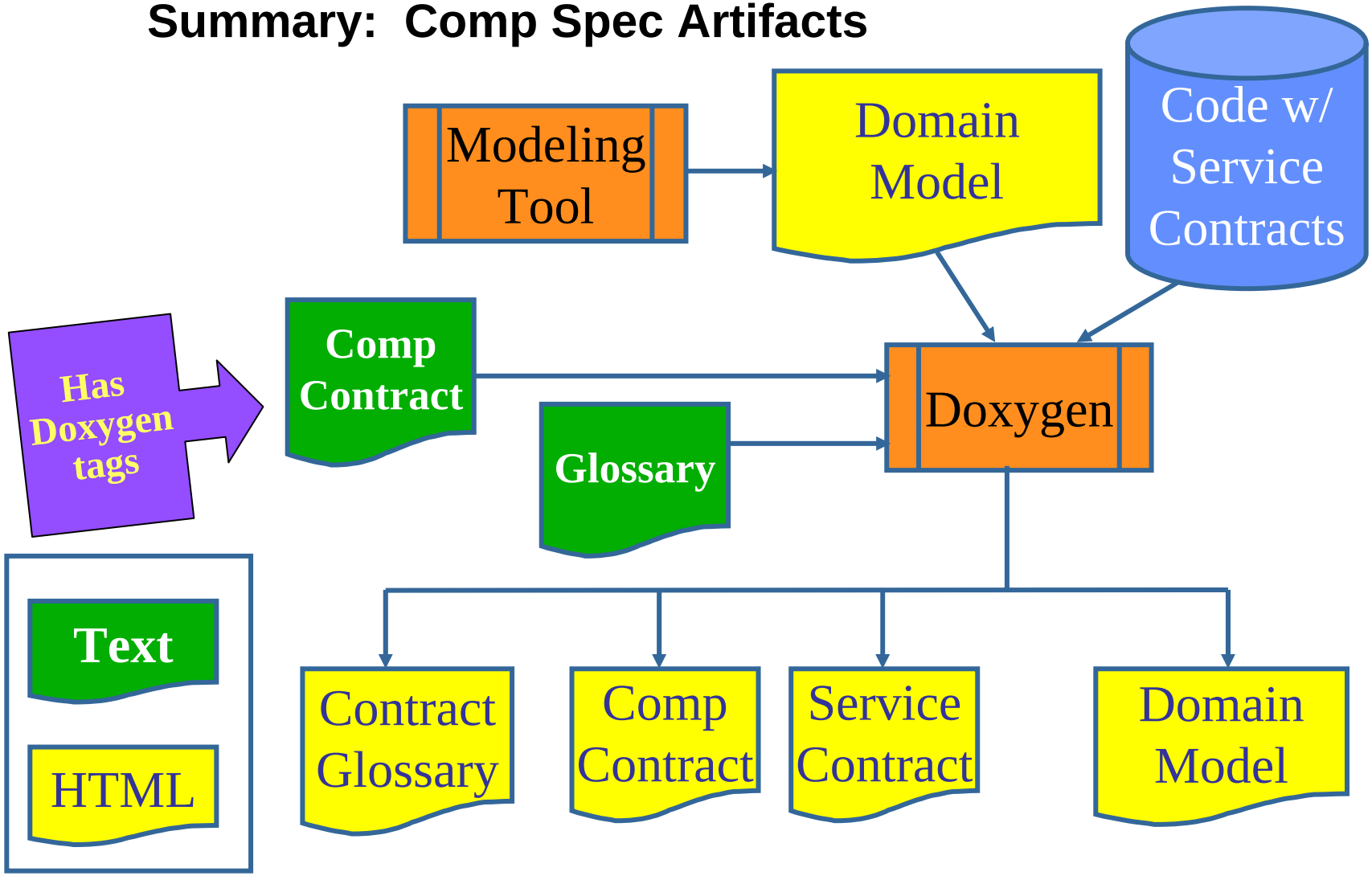
Component states that must be true after component instantiation/initialization **and** at the start and completion of any offered service. Multiplicity invariants are specified in the Domain Model (e.g., an Engagement may have between 0 and X weapons), so they should not be specified here. Note that when a service terminates due to exceptional behavior, part of the component's exception handling should insure that the invariant states are still true. If they are not true, the component should restore the invariant states. If this is not done, the component will be in an unstable state.

**Exceptions:**

*NameOfException*

- What happens if component initialized when preconditions are not met.
- What happens if preconditions are met, but postconditions cannot be met.
- What happens if an invariant fails to be maintained.
- Notes
  - An exception is only handled if explicitly stated, otherwise component behavior is undefined.
  - "Exception" means behavior in the event of contract violation. It is not meant to imply that a C++

# Summary: Comp Spec Artifacts





## How This Technique Addresses Openness

- Provides *well-defined* interfaces using open paradigms
  - DbC and UML Domain Modeling
- Generated using open tools
- Output is readable in any HTML browser



## What the Component Spec Provides

- Software Architect
  - Defines a component's role in overall architecture
  - Facilitates component reuse
- Software Developer
  - Defines implementation constraints
  - Describes exceptional behavior
- System Engineer
  - Facilitates understanding of the component's role in fulfilling requirements
- Component Test Engineers
  - Provides basis for writing component level tests

Lets the stakeholders know the rules



## Level of Effort for Sample Component

- Task requires domain knowledge
  - *Does not need to be expert*, but does need access to an expert
- Documented 40 services
  - 28 trivial, e.g., getters/setters
  - 12 non-trivial
- 3.5 staff weeks

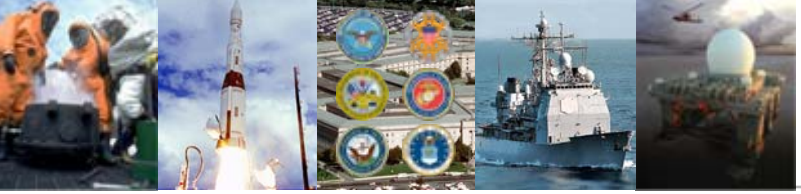
Writing specs takes time



# For More Information

**Kenneth Klein**
**kklein1@csc.com**
**856-252-2359**

**Joanis Ploumitsakos**
**jploumit@csc.com**
**856-252-2091**





# Backup

## DbC Resources


- Larman, C., *Appying UML and Patterns: Introduction to OOA/D & Iterative Development, 3rd ed.* 2005: Prentice Hall.
- Szyperski, C., *Component Software: Beyond Object-Oriented Programming 2nd ed.* 2002: ACM Press.
- Mitchell, R. and McKim, J. *Design by Contract, by Example.* 2002: Addison-Wesley, Inc.
- Cheesman, J. and Daniels, J. *UML Components: A Simple Process for Specifying Component-Based Software.* 2001: Addison-Wesley, Inc.
- Hunt A. & Thomas D., *The Pragmatic Programmer*. 2000: Addison Wesley.
- Meyer, B. *Object-oriented Software Construction, 2nd ed.* 1997: Prentice Hall.
- http://archive.eiffel.com/doc/manuals/technology/contract/page.html
- http://ootips.org/lsp.html



# Comp Spec Development Process