

GSAW 99

Interface Management Issues in Component-based Architectures

Phillip Schmidt

The Aerospace Corporation

El Segundo, CA 90245-4691

Phillip.P.Schmidt@aero.org



**THE AEROSPACE
CORPORATION**

Agenda

- **Some Trends**
- **Architectural Challenges**
- **Assumptions/Definitions**
- **Representation/Policy Objectives**
- **Approach to Architectural Representation**
- **Interface Management Strategies/Policies**
- **Summary**

Some Trends

- **Component-based systems as way to support reuse and evolution**
- **Independently developed components**
- **Interface definition without source code or internal design**
- **Evolution of new capabilities via new interfaces**
 - Component change management services (e.g. Context state management, deployment management, reconfiguration management)
 - Interface discovery, trader services, dual interfaces
 - Transaction services and fault tolerant computing
 - Component-level access control
 - Customizable activation services
 - Scriptable components

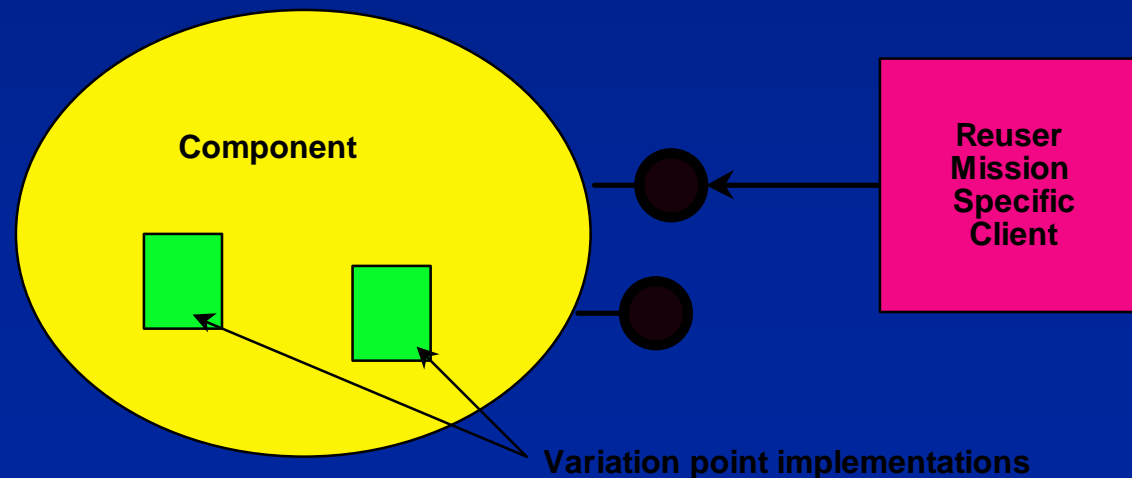
Architectural Challenges

- Design reference architectural **representations/abstractions** that recognize components and support interface/implementation separation, and component evolution
- Be able to **map** architectural (logical) representations to realized (deployed) components and their interfaces
- Define interface management **policies** that recognize interface/component evolution/deployment
- **Maturity of modeling tools**
 - multi-perspective
 - descriptive/prescriptive



Assumptions

- Reusers will build (**mission-specific**) code to interfaces provided by a component-based architecture
- Reusers may **augment/provide implementations** to interfaces the component framework defines



Definitions

- **An interface** is an abstraction which has a permanent, unique name and represents a collection of operations that support a service to a client.
- **The interface abstraction is separate from its implementation**
 - No attributes
 - Can have inheritance relationships with other interfaces
- **A (logical) component** is an abstraction which represents a functionally related collection of services. The component provides and uses these services via interfaces.
- **Object** encapsulates state and behavior; unit of instantiation. Has unique identify (e.g., object reference)
- **Component category** identifies a related collection of components
 - A domain component typically belongs to one component category.
- **An interface category** is a collection of related interfaces.
 - A component's interfaces could belong to multiple interface categories.
 - A category could specify optional vs. required interface provisioning requirements for compliant components
- **Interface management** pertains to the evolution of component interfaces over time.

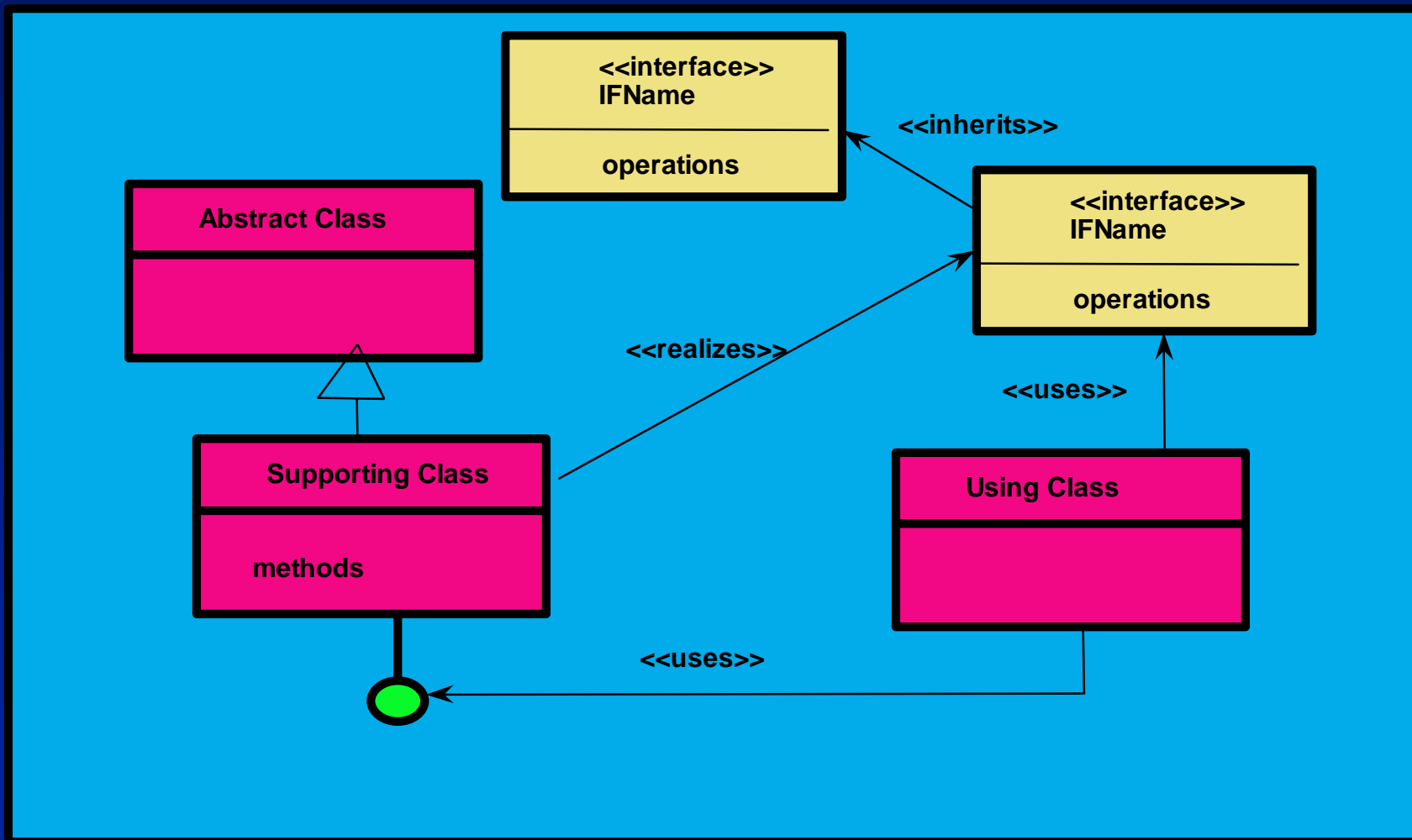


Representation/Policy Objectives

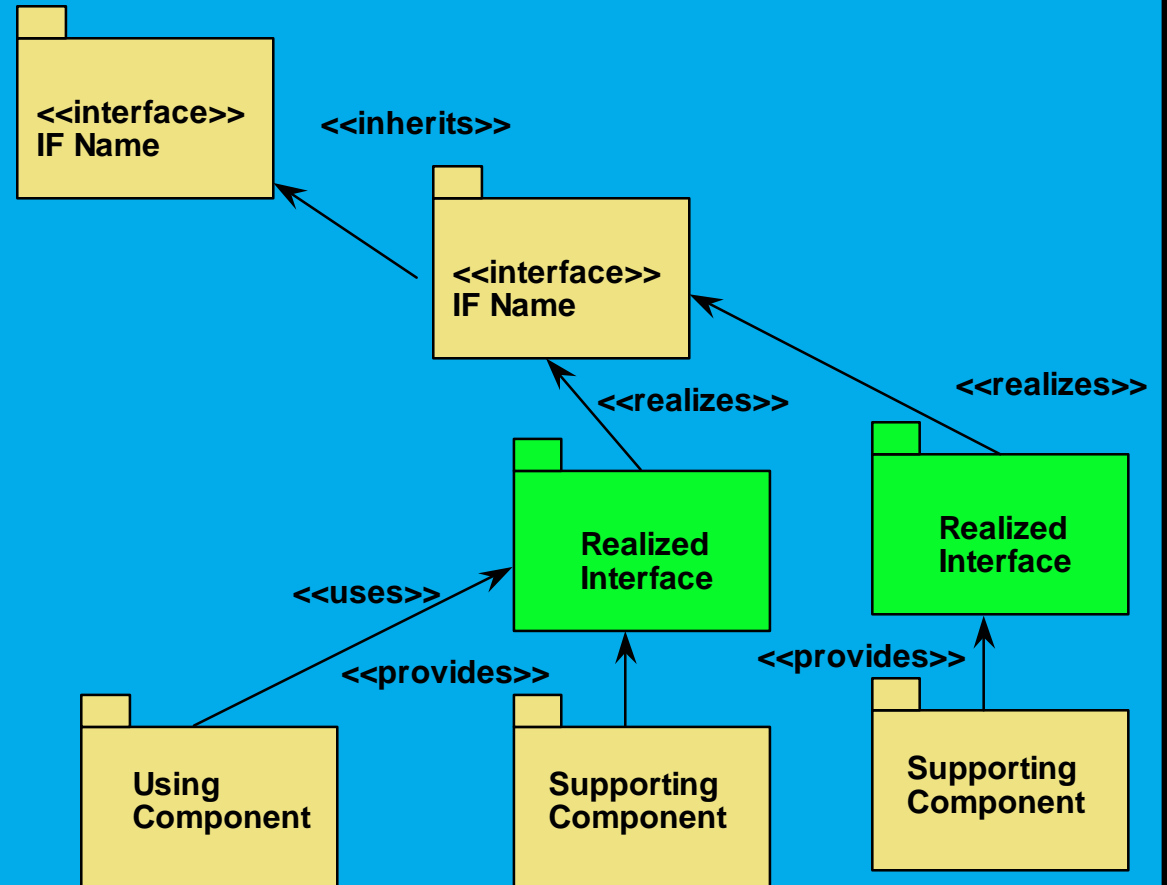
- **Want reference architecture representation**
 - Interface and implementation separation
 - Avoid specifying design classes too soon
- **Want realizable architectures**
 - Interfaces do get realized
 - Components do get deployed
 - Determine which interfaces are provided as well as used
- **Want to solve the Component Deployment Problem--both in our logical representation models as well as in our Sustainment process.**
 - Reuser code using an earlier version does not break when new CCT components are deployed.
 - All reusers are not forced to simultaneous, lock-step, evolution
 - There is a **versioning** / component bundling issue. Configuration management alone is not sufficient.
- ***Approach: Define layered abstractions that capture interfaces, realizations of those interfaces, and components***



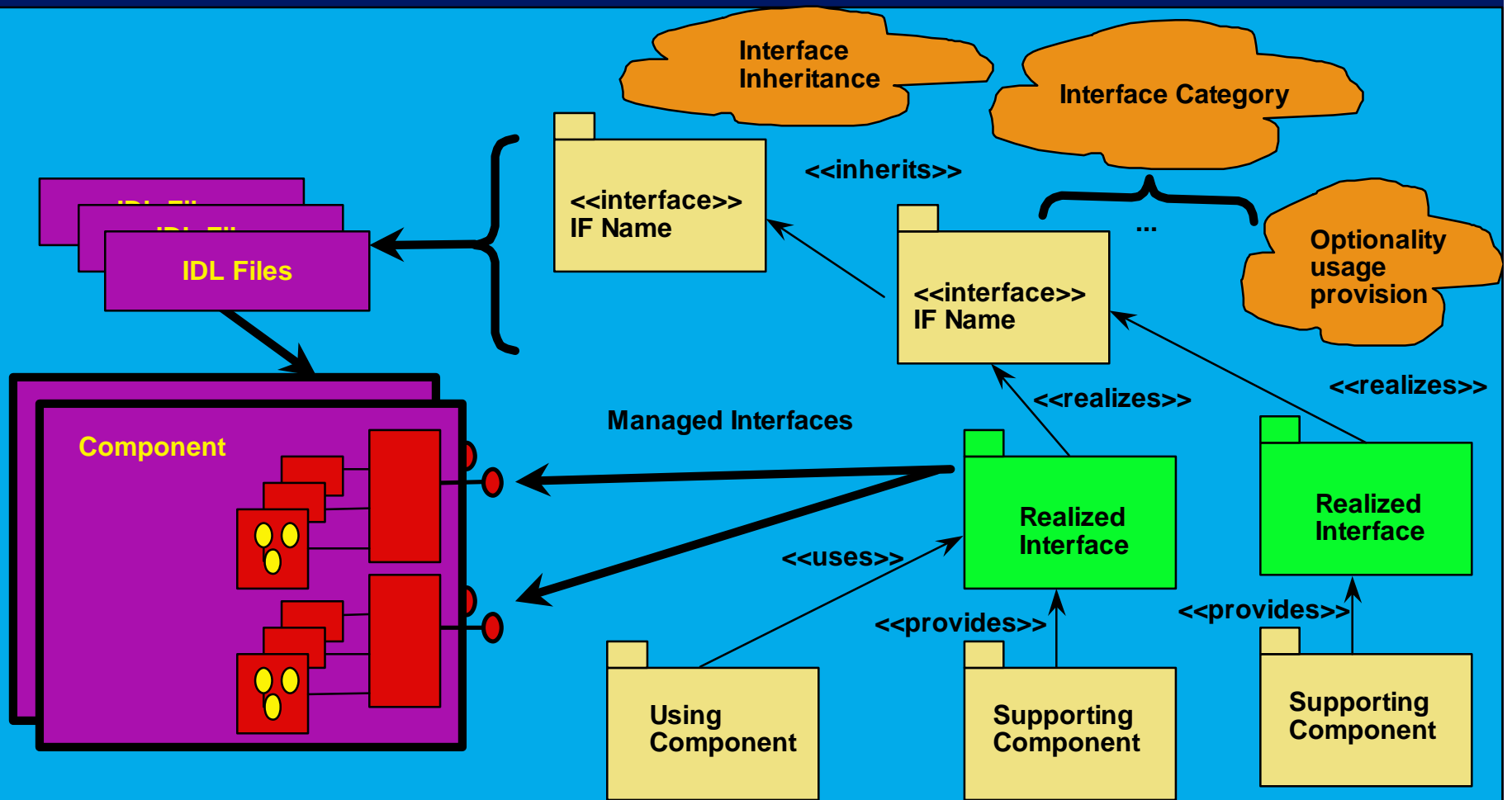
Basic UML Representation



Component-based Abstractions



Architectural Mappings



Some Benefits

- **Supports general evolution strategy: New interfaces can be defined and added to components.**
 - New component additions (e.g. via independent development) accommodated as realized interfaces/components
 - Evolution of interfaces managed architecturally traceable to deployable components
 - Logical components traceable to deployable components as well.
- **Separation of interface from implementation preserved**
 - Interface versions
 - Implementation versions
 - Architectural representation need not change for simple implementation fixes
- **Managed architectural interfaces are specified by IDL**
- **Architectural direction can be provided through the management of interface categories.**
 - Domain-specific ontologies of interface categories
 - Interface categories are more general than component categories

Some Interface Management Policies

- **Interfaces are formally proposed within an architectural context**
 - New problem solution or alternative
 - Can be offered as a possible replacement
- **Interfaces are contractually specified**
 - Syntactic interface specification
 - *Semantic functional description*
- **Interfaces evolve and their architectural evolution is “owned by” the keepers of the reference architecture**
 - Evolution of IDL files essentially define Interface families
- **Interfaces are immutable after publication**
 - Interface publication creates a **contract** between reuser and owner
 - Interface names are globally unique and never reused.
- **Interface growth is managed:**
 - Natural: **effectively die** off when they are no longer used or replaced by something better (yeah--right)
 - Expiry: No more DPRs after a certain date
 - Technical review upon their proposal
 - *Deprecation (e.g. we goofed!)*
- **A technical review board is established to guide architectural evolution and future interoperability**
 - *Interface category management can structure interfaces across components*
 - *Support for advanced services (e.g. interface discovery, scriptability, etc.)*



Summary

- **Interface management needs to be addressed to solve the component-deployment problem**
- **Introduced some architectural abstractions to assist representation**
- **Highlighted how mappings to realized component deployment could be managed**
- **Suggested terminology, strategies, and policies to address interface management issues in component-based evolution**

