



**Openwings  
Architecture Description Specification  
Ver 1.0 Final**

COPYRIGHT ©2000-2003 General Dynamics Decision Systems, INC.  
ALL RIGHTS RESERVED  
This document is subject to "Terms of Use" as described at <http://www.openwings.org>.

## PREFACE

About the Open Specification Process

A brief overview of the community process for developing Openwings Specifications is discussed below. This process was developed by General Dynamics Decision Systems and is similar to the community process for developing Java specifications.

A Process Management Organization (PMO) oversees development and maintenance of all Openwings Specifications. Any company, organization or individual can join the Specification development team; team members are called Participants. The latest Openwings Specifications will be maintained on a Public Web Site.

Participants can submit requests to the PMO to modify existing or develop new Specifications. If the PMO accepts the request, the PMO will then begin the process of forming an Expert Team and will appoint a Specification Lead.

The Specification Lead will direct the Expert Team in developing the new Specification. Once the team agrees on a draft of the Specification, it will be posted on the Public Web Site for public review. A member of the Expert Team will also begin development of a Reference Implementation and Compatibility Test Suite for the new Specification.

Based on public comments, a final release of the Specification will be produced by the Expert Team. Concurrently, the Reference Implementation and Compatibility Test Suite will be completed. Once the Final Public Release of the Specification is completed and posted on the Public Web Site, the Expert Team will disband.

This process is intended to provide rich consensus based Specifications.

Contributors

We would like to recognize and thank the following people for their contributions to this document.

Author(s):  
Guy Bieber

Expert Team Members:  
Stuart Lewin, BAE Systems North America  
Ramesh Nagappan, Sun Microsystems  
Ayal Spitz, Mitre Corp  
Dr. Dave Usechak, Ocean Systems Engineering Corporation

Other Contributors:  
Jeffrey Carpenter, John Grosberg, Pat Vessels

---

<b>1.</b>	<b>INTRODUCTION .....</b>	<b>1</b>
1.1	PURPOSE .....	1
1.2	SCOPE .....	1
1.3	DEFINITIONS .....	1
1.4	OVERVIEW .....	2
1.4.1	<i>Openwings Overview</i> .....	2
1.4.2	<i>Document Overview</i> .....	2
<b>2.</b>	<b>GOALS AND REQUIREMENTS .....</b>	<b>4</b>
<b>3.</b>	<b>ARCHITECTURE.....</b>	<b>5</b>
3.1	MODELING LANGUAGES.....	5
3.1.1	<i>Unified Modeling Language</i> .....	5
3.1.2	<i>Interface Definition Language</i> .....	7
3.1.3	<i>Architecture Description Language</i> .....	7
3.2	ROLES .....	10
<b>4.</b>	<b>COMPLIANCE.....</b>	<b>11</b>
<b>5.</b>	<b>REFERENCES AND FURTHER READING.....</b>	<b>12</b>

FIGURE 1:	RELATIONSHIPS BETWEEN ADL, UML, AND IDL .....	3
FIGURE 2:	GOALS / REQUIREMENTS TABLE.....	4
FIGURE 3:	UML DIAGRAMS.....	6
FIGURE 4:	ACME ADL BASICS.....	8
FIGURE 5:	IMAGE SERVER ACME ADL EXAMPLE .....	9
FIGURE 6:	IMAGE SERVER EXAMPLE INTERFACES.....	9
FIGURE 7:	INTERFACE GROUPING.....	10
FIGURE 8:	COMPLIANCE TABLE .....	11

## 1. Introduction

### 1.1 Purpose

The purpose of this document is to describe the Openwings architectural component model for plug and play software / hardware components. This document introduces the terms and notation used in the architecture.

### 1.2 Scope

This document is intended to provide a framework for component models and a formal description language. It is not meant to replace existing component models such as CORBA and EJB. It does, however, abstract these models. As with all of the Openwings APIs this specification uses existing commercial standards.

### 1.3 Definitions

The following definitions provide a context for this specification:

Term	Definition
Architectural Definition Language (ADL)	A language for defining how systems are constructed. Systems are composed of hardware and software components with formally defined interfaces and connections.
Interface Definition Language (IDL)	Any language that defines interfaces independent of their implementations.
Unified Modeling Language (UML)	A graphical notation for object models. It has been recently extended to cover more systems related products such as deployment and component diagrams.
System	A system is a collection of interacting components. Some architectures refer to collections of systems as families and other architectures refer to systems that compose larger systems as elements. We will use systems to refer to all of these descriptions.
Component	<p>The SEI working definition of a component is:</p> <ul style="list-style-type: none"> <li>• A binary implementation of functionality</li> <li>• Contractually specified</li> <li>• A unit of independent deployment</li> <li>• Subject to 3<sup>rd</sup>-party composition</li> <li>• Conformant with a component model</li> </ul> <p>We add to this that a component provides / uses services through contractually specified interfaces. Components are typically described as processes, but could be entirely independent threads running in one process or Java Virtual Machine. Components may be nested much like systems. Additionally, hardware components may be represented by software components. A system can be composed of a single component typically visualized as a box.</p>
Connector	Connectors represent interactions between components. A connector is a transport implementation for connecting component ports together. Examples of connectors are CORBA, RPC, RMI, EJB, DCOM, sockets, etc. Typically visualized as a line between ports.
Service	A service is a capability provided by components via well-defined interfaces or collections of interfaces.
Space	A space is a collection of components working together as a single system.

Interface	An interface is an implementation independent description of a service. It provides specific syntactical definition as well as a semantic behavioral specification. Examples of interfaces are Java interface specifications or CORBA IDL interface specifications.
Port	A port provides an attachment point with a component and its environment. Ports provide the proxy code to attach interface binding to connectors. Proxy code takes transport specific interfaces and translates them to a transport independent interface binding.
Binding	A binding is a mapping of one language to another. For example, CORBA defines bindings from CORBA IDL to several languages (C++, ADA, C, Smalltalk, etc.)
Role	In an interface between two components, the role describes who does what, e.g. client and server or seller and buyer.
Properties	Properties are attribute / value pairs (also known as key / value pairs) that are associated with an object or thing.
Attachments	The points at which things are put together

## 1.4 Overview

### 1.4.1 Openwings Overview

Openwings™ is an open community, non-proprietary effort to define specifications for self-forming, self-healing systems. Motorola (now General Dynamics Decision Systems) and Sun Microsystems established the Openwings™ consortium in June of 1999. Since then, over 100 companies have registered to help mature its development, and more companies are registering daily.

The core Openwings™ framework is designed to incorporate existing commercial standards and is intended for use in both commercial and military environments. Openwings™ is being developed using a community development process modeled after the very successful Java Community Process. Anyone may join the Openwings™ community, participate on expert teams, or use the resulting specifications free of charge by merely signing up at the community web site (<http://www.openwings.org>).

The Openwings™ Architecture provides a framework for building plug-and-play, service-oriented, network-centric, self-forming, self-healing systems that are independent of middleware, databases, platforms, and deployment contexts. Openwings™ has a special focus on issues of availability, security, and interoperability. Openwings™ is the embodiment of a new movement in the software engineering community towards a paradigm known as Service-Oriented Programming (SOP). For an introduction to SOP, refer to <http://www.openwings.org/download/specs/ServiceOrientedIntroduction.pdf>.

### 1.4.2 Document Overview

This document describes how software architectures are to be defined. Component models are the next evolution of objects. A component is really a collection of objects that implement interfaces. A key requirement is that the interfaces be defined independently of the implementation. This independence allows the software to be used based on services it provides, instead of what it is. Hence, specifics of any vendor's

implementation are hidden from the application developer. All the application developer needs to know is the interface, not the construction. As seen in the Java language, the inclusion of interfaces in an object-oriented language has created a significant new component market. It in fact is the basis for all Java specifications and the bean framework.

This document defines a way to produce software component architecture descriptions that are transport and deployment environment independent using existing APIs and protocols.

The following diagram visualizes the relationships between interfaces, objects, components and the modeling languages they are described with.

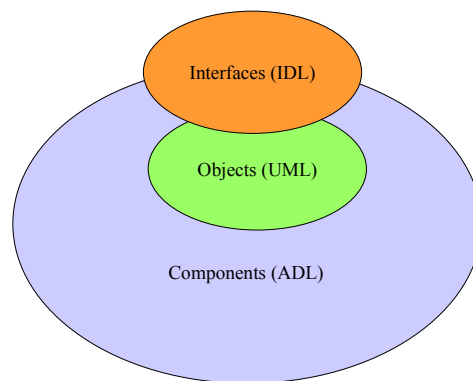


Figure 1: Relationships between ADL, UML, and IDL

This diagram shows that components contain objects, which implement interfaces. At the component level these interfaces are published, i.e. the implementation is hidden.

This document is similar to the format of other Openwings specifications:

- Goals and Requirements – Describes requirements of the specification.
- Architecture – Describes the various architectural aspects of the specification.
- Compliance – This section describes what is required to comply with the specification.

## 2. Goals and Requirements

The following table shows the goals and requirements of this specification. Each requirement references one or more of the top-level requirements found in the Openwings Overview.

#	Goals / Requirements	Ref
1.	Shall provide a formal architecture language that maps directly to code.	7
2.	The architecture language shall integrate easily with UML and IDL.	7
3.	The architecture language shall support synchronous / RPC like protocols, including CORBA, DCOM, RMI, and EJB.	6
4.	The architecture language shall support asynchronous / messaging like protocols, including JMS, CMS, and ModIOS Event Server.	6
5.	The architecture language shall support stream like protocols such as RTF for video and voice.	6
6.	The architecture language shall provide interfaces independent of implementations at the component level (much like Java interfaces), such that components may implement one or more independent interfaces.	7

Figure 2: Goals / Requirements Table

### 3. Architecture

#### 3.1 Modeling Languages

##### 3.1.1 Unified Modeling Language

UML was originally used to graphically describe object models. Recently, it has been extended to model systems and components. Currently UML is made up of the following diagrams / sub-models:

Name	Description	Example
Use Cases	Used to describe user interactions with a system. The intent of these diagrams is to focus on how people will use the system.	
Class Diagram	This diagram is used to show the traditional object model. It shows aggregation, inheritance, associations, and interfaces. Most tools now support round trip engineering between a class diagram and source code.	
Interaction Diagrams	Sequence and Collaboration diagrams are used to show time interaction between objects.	
Package Diagrams	Package diagrams provide a mechanism for grouping classes. This is modeled after the Java package concept.	

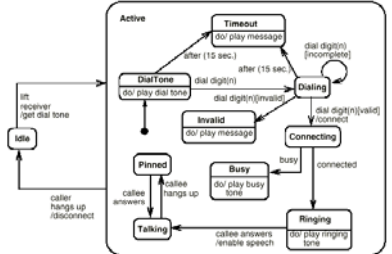
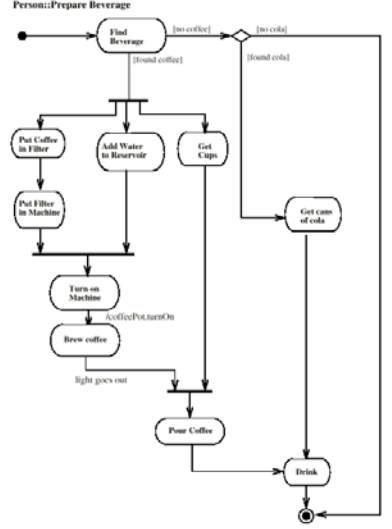
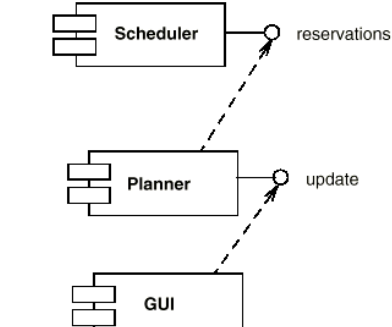
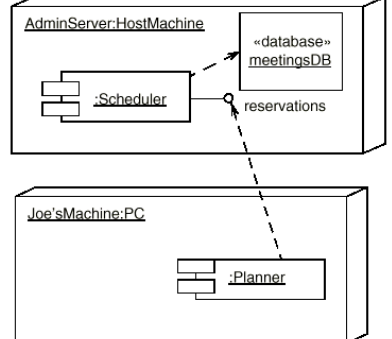
<p>State Diagrams</p>	<p>State diagrams are used to model the internal state behavior of objects.</p>	 <p>Figure 3-49 State Diagram</p>
<p>Activity Diagram</p>	<p>Activity diagrams are used to model task workflow and synchronization.</p>	 <p>Figure 3-50 Activity Diagram</p>
<p>Component Diagram</p>	<p>Component diagrams are used to model processes and their published interfaces. Note the lollipop notation for interfaces.</p>	
<p>Deployment Diagram</p>	<p>Deployment diagrams are used to map software components (processes) to hardware.</p>	

Figure 3: UML Diagrams

UML is a graphical notation, which currently has no formal Backus-Naur Form (BNF). Besides this, the majority of UML models do not have mappings to programming

languages. Only recently has it been possible to do round trip engineering between class diagrams and source code. This inability to map all of the models to code causes a disconnect between the conceptual behavior and actual implementation. This is often evident when looking at design products that seemingly have nothing to do with their implementation.

UML is beginning to get stronger at the architectural level, but again lacks formalism. It is expected that Openwings will make heavy use of UML diagrams. These diagrams all have direct mappings to implementations. In addition, activity diagrams, interaction diagrams, and state diagrams will be used as necessary. Interface Definition Languages will be used to define component interfaces. For describing components themselves, the more formal Architecture Description Languages will be used.

### **3.1.2 Interface Definition Language**

UML has the concept of an interface in its class model. This implementation independent concept of an interface comes from Java. In Java these are used both for internal interfaces and external interfaces. Traditional Interface Definition Languages such as CORBA IDL or Microsoft IDL focus on inter-process interfaces. The Openwings Interface Definition Specification covers this topic in more detail. Suffice it to say here, that components will have interfaces defined in some Interface Definition Language.

### **3.1.3 Architecture Description Language**

The DARPA research community has been doing a lot of work on modeling software architectures. Several different architectural languages have come out of this effort. A formal meta-language was later defined to transfer these component models from one custom language to another. The result of this effort is the ACME ADL out of Carnegie Mellon University. The following diagram shows the basics of the notation and language.

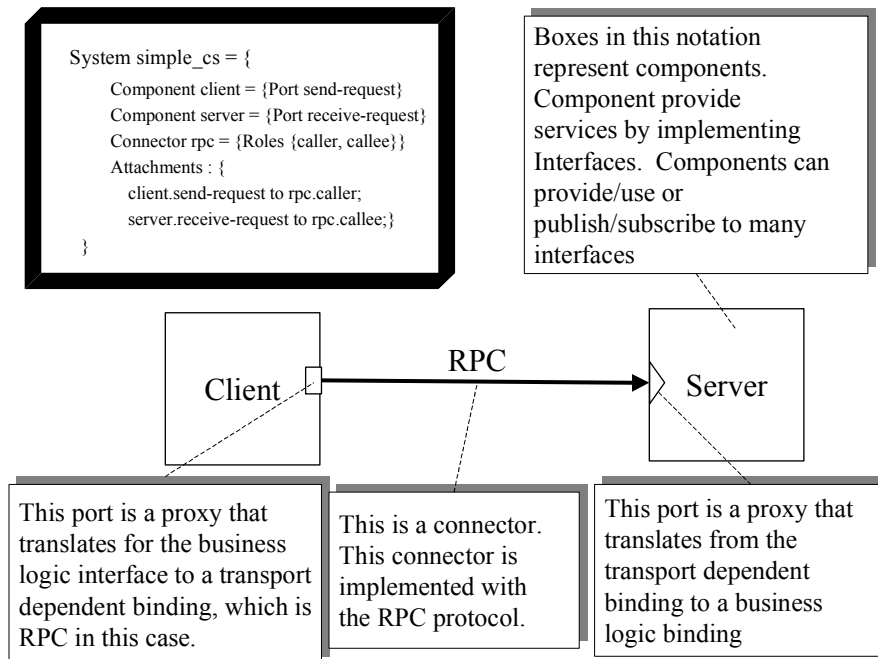


Figure 4: ACME ADL Basics

The ACME ADL provides a formal language for these diagrams that can be easily used to design or represent software components at runtime. This model provides the basic constructs for transport independent communication between components, by providing the concept of connectors and ports. Ports can be implemented as proxies that are discoverable at runtime. Connectors can also be discovered at runtime. Hence applications are interoperable with legacy applications using any protocol / transport and new applications can easily change to more efficient transport layers without code changes. Here is an example of an image server in ACME ADL (note the ACME diagram is overlaid on a UML deployment diagram):

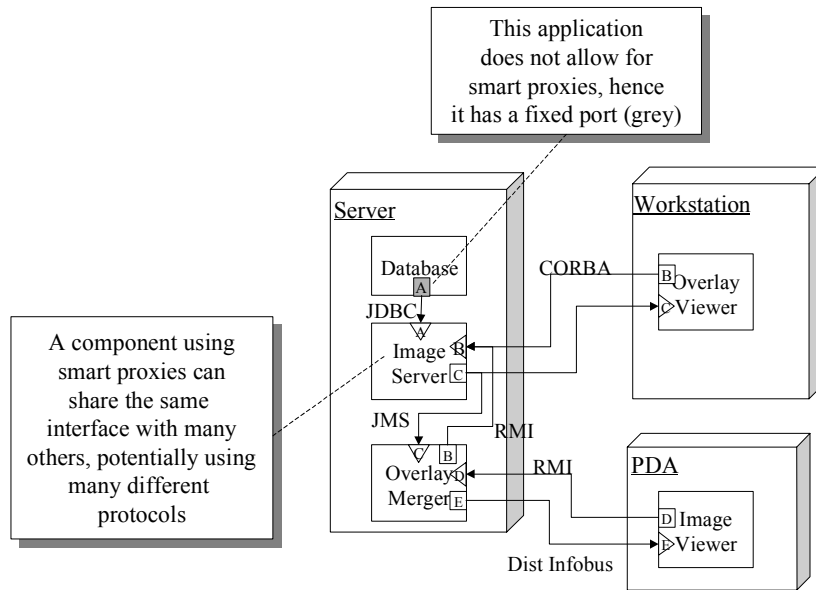


Figure 5: Image Server ACME ADL Example

In this example a database stores indexes to images stored in flat files. The Image Server abstracts away the persistence mechanism and serves the images to consumers. The Overlay viewer may actually request multiple images from the image server, merge them intelligently, and display them. The Overlay Merger provides a service to do a similar thing for ultra thin clients such as a PDA. In these examples there are several interfaces (in a real example the labels would translate to URL style names):

Interfaces	Description
A	A standard JDBC interface
B	An interface for requesting images at a fixed resolution and size.
C	An interface for notifying image consumers of new images they have requested.
D	An interface for requesting a series of images merged at a fixed resolution, size, and format.
E	An interface for notifying overlay consumers of new overlays they have requested.

Figure 6: Image Server Example Interfaces

There are a few extensions we propose to the notation:

- Ports have labels used to associate ports to interfaces.
- Ports can be grayed out to designate a fixed port (transport layer is fixed at compile time).
- Individual unidirectional interfaces can be combined to make bi-directional, multi-protocol interfaces (see the notation in the diagram below).
- A lollipop notation will be used to show bus style, one to many, interfaces (see the UML component diagram for an example).

All of these extensions do not affect the core ACME ADL language; they just extend the notation.

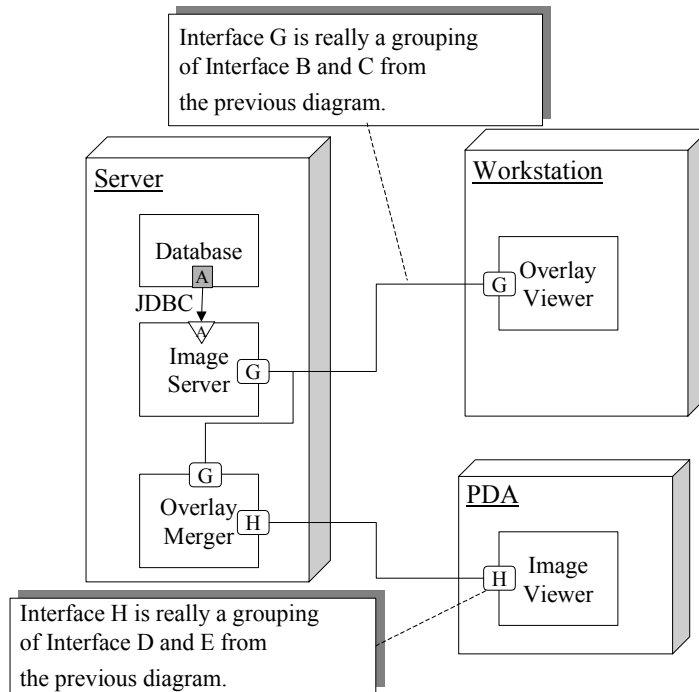


Figure 7: Interface Grouping

Please refer to the ACME ADL specification for the formal BNF form of the language and further examples.

### 3.2 Roles

The following key roles have been defined for Openwings.

Role	Question
Systems Engineer	Does the design work?
System Integrator	Does the implementation integrate easily?
Tester	Does the system meet requirements?
Deployer	How to configure the system?
Maintenance	What's not working and why?
Operator	It just works!!!
Software Developer	How do I build this software component?
Logistics support	How do I service the customer most efficiently?
Hardware Developer	How do I build this hardware component?

#### 4. Compliance

Here is the compliance checklist for this specification.

#	ITEM
1	The UML use case, class diagram and deployment diagrams shall be used. The UML interaction diagram, package diagram, state diagram, and activity diagram shall be used as needed.
2	Interfaces shall be defined independent of transport layers (refer to the Openwings Interface Specification and Openwings Component Specification).
3	Software Components and their connections will be modeled with ACME ADL.

Figure 8: Compliance Table

## 5. References and Further Reading

1. Openwings Architecture Whitepaper,  
<http://www.openwings.org/download/specs/openwingswp.pdf>
2. Openwings Interface Specification,  
[http://www.openwings.org/download/specs/openwings\\_interface.pdf](http://www.openwings.org/download/specs/openwings_interface.pdf)
3. <http://www.omg.org>, OMG Web Site
4. <http://www.rational.com/uml/index.jtml>, Rational UML Web Site
5. [http://www.cs.cmu.edu/~acme/acme\\_documentation.html](http://www.cs.cmu.edu/~acme/acme_documentation.html), CMU ACME ADL Documentation Web Site