MODELING OPERATIONAL TASK ASSIGNMENT IN AIR FORCE WING COMMAND AND CONTROL

THESIS

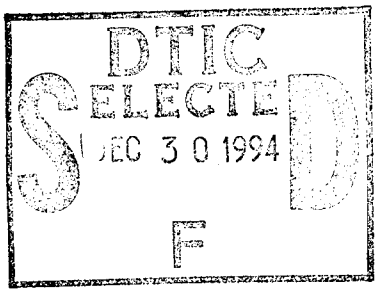
Robert J. Hunt
Captain, USAF

AFIT/GCS/ENG/94D-09

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GCS/ENG/94D-09



MODELING OPERATIONAL TASK ASSIGNMENT IN AIR FORCE WING COMMAND AND CONTROL

THESIS

Robert J. Hunt
Captain, USAF

AFIT/GCS/ENG/94D-09

Approved for public release; distribution unlimited

AFIT/GCS/ENG/94D-09

MODELING
OPERATIONAL TASK ASSIGNMENT
IN AIR FORCE WING COMMAND AND CONTROL

THESIS

Presented to the Faculty of the Graduate School of Engineering

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the

Requirements for the Degree of

Master of Science

Robert J. Hunt, B.A.

Captain, USAF

December 1994

Table of Contents

List of Figures

Acknowledgements

I wish to thank my research advisor, Dr Thomas Hartrum, for his expert advice and assistance. His easygoing manner and helpful pointers were much appreciated. I cannot imagine a better research advisor here at AFIT. I also wish to thank my readers, Maj Paul Bailor and Maj David Luginbuhl for their insightful (if not inciteful) comments and suggestions.

I wish to thank my fellow software engineering classmates for making the lab an interesting place in which to work. I especially wish to thank my research partner, Mike Sarchet. His energy, sense of humor, and putting "skills" helped make this effort the success that it was. I'm not sure anyone could make this process "fun", but he made this process as fun as it could possibly be.

I thank my parents, Bob and Dianne Hunt, for helping us during some medical emergencies. Without their assistance, I may never have completed this effort. Finally, I must thank my wife, Monique, for supporting me in this work during some difficult times. I only delivered a thesis. She brought Jamie into this world and showed me what is really important in life. I especially thank my children, William, Rachel, and Jamie, for understanding that I couldn't always be there to play, but for loving me anyway and always being there at the door to greet me. This thesis could not have been successfully completed without your support.

Robert J. Hunt

Abstract

This research investigated the feasibility of applying software engineering technology to the Air Force wing command and control (C2) domain. As part of this research, domain analysis and object-oriented techniques were investigated and a specific approach was chosen to analyze the domain. Analysis of the domain resulted in an object-oriented domain model that captured the key objects, operations, and associations, and behavior of wing C2. The domain model was used to design and implement a prototype software tool that enables wing decision makers to assign resources to mission tasks and to make assessments about automation's impact on wing C2 operations.

MODELING
OPERATIONAL TASK ASSIGNMENT
IN AIR FORCE WING COMMAND AND CONTROL

I. Introduction

1.1 Background

A combat commander's ability to effectively command and control his forces often means the difference between victory and defeat on the battlefield. Nowhere was this lesson more clear than in operation DESERT STORM. The Coalition forces, knowing the importance of command and control (C2) to combat operations, began the air war by targeting high priority Iraqi C2 nodes, and within a matter of hours Saddam Hussein had lost the ability to effectively command and control his forces. This decisive blow by the Coalition forces early in the campaign prevented Saddam Hussein and the Iraqi military from ever gaining the upper hand.

Today, battlefield commanders rely heavily on computer-based systems to maintain positive command and control of their combat forces. For example, the AWACS system provides commanders with a composite picture of the air battle, while Joint-STARS provides commanders with a picture of the ground battle. The situational awareness provided by these systems gives commanders the ability to control the air space and direct air strikes in near realtime.

One reason commanders have come to depend on C2 systems is because capability and performance have increased dramatically in recent years. C2 systems can now take multiple sources of data and correlate the information into a concise picture of the battle. Directly responsible for these robust systems are advances in computer technology like processing capability, networking capability, and user interfaces. However, as C2 systems have become more specialized and complex so has the effort required to develop new C2 systems and maintain existing systems. This, in turn, has led to increased development and maintenance costs. Also, battlefield commanders now have the responsibility to interpret and act upon the vast amount of information being produced by these modern C2 systems. Increasing costs and system complexity have led C2 experts to the realization that insufficient attention has been focused on how to integrate these highly specialized systems into a single battle management system. Without a mechanism to understand how a wing conducts its mission, it is difficult (if not impossible) to perform this kind of mission analysis or integrate new C2 systems into an overall battle management architecture.

1.2 Problem

Currently, Air Force wing and headquarters decision makers do not have a way to systematically represent how a wing goes about performing its mission. This deficiency means key personnel do not have the management and analysis tools necessary to assess the impact of C2 automation on wing operations or understand what requirements are best satisfied with computer-based C2 systems. Further, system developers and maintainers also do not have a knowledge base to use in the specification and design of new computer-based C2 systems or in modification of existing systems.

One way to capture the knowledge about key objects, operations, and relationships relevant to wing C2 is through domain analysis. Performing a domain analysis on the Air Force wing C2 domain would result in the production of a domain model that consists of taxonomic information to describe the domain structure and rule bases to capture domain knowledge. The domain model could then be used by wing and headquarters personnel to analyze the effects of C2 automation on unit readiness and effectiveness and best decide how to allocate scarce resources. The domain model would also help key decision makers in understanding what aspects of the mission are most suitable for automation. Furthermore, system developers and maintainers would have a definitive repository or knowledge base to establish the design of new C2 systems or justify the modification of existing systems. Thus, a domain model of wing C2 operations would serve as a management tool for wing and headquarters decision-makers while also serving as a reusable requirements repository for system developers and maintainers (see Figure 1).

1.2.1 Problem Statement. **Demonstrate the feasibility of applying software engineering technology to Air Force wing command and control.**

1.2.2 Research Objectives.

- Demonstrate how domain analysis techniques can be used to develop engineering models of Air Force wing C2 operations.
- Show how engineering models of Air Force wing C2 operations can be used to develop a software tool that determines the effects of automation.

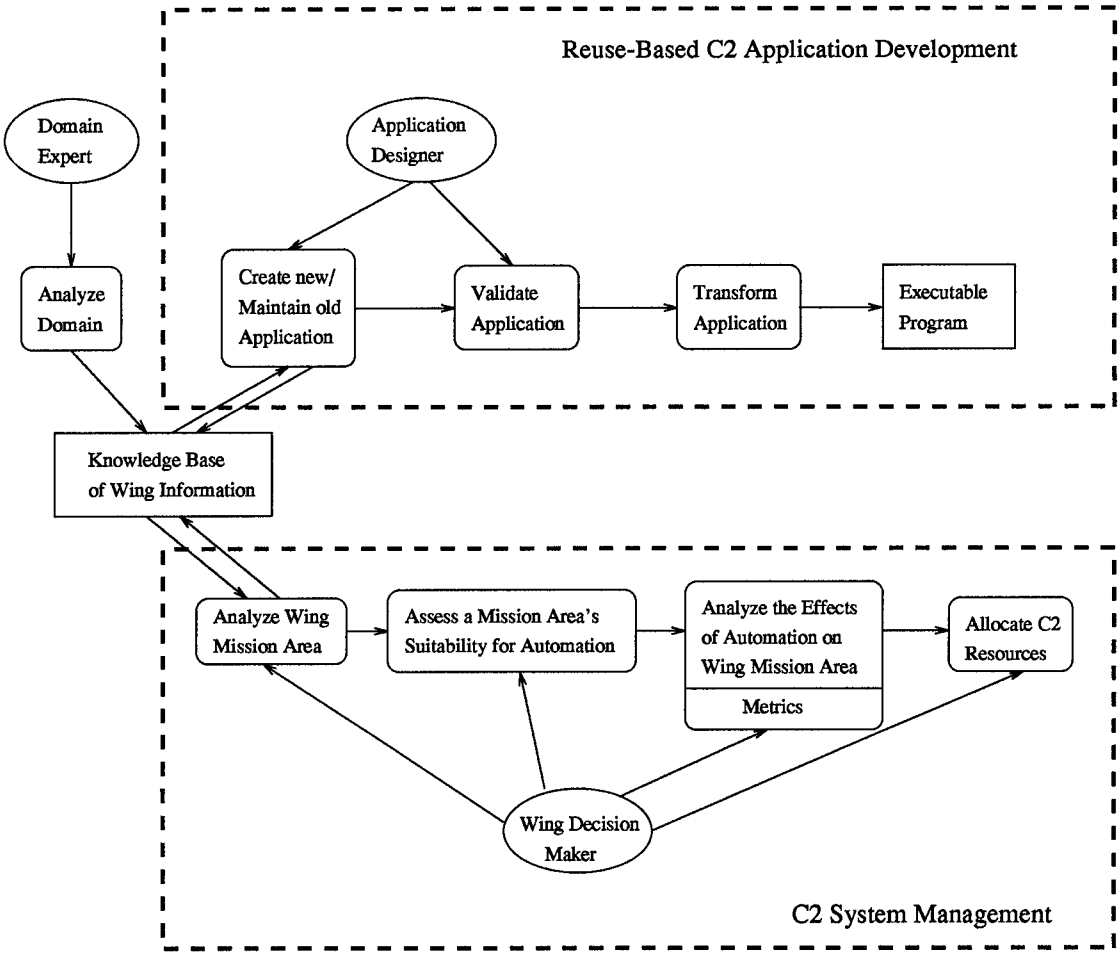


Figure 1. Formalized Model Development Process (21:1–2)

1.3 Scope

The domain analysis portion of this thesis effort focused on Air Tasking Order (ATO) processing, scheduling, and execution. As such, this research focused on the details of the Operations Group while the supporting groups were modeled more abstractly. Figure 2 shows the organization of a typical Air Force wing and the scope of this research effort.

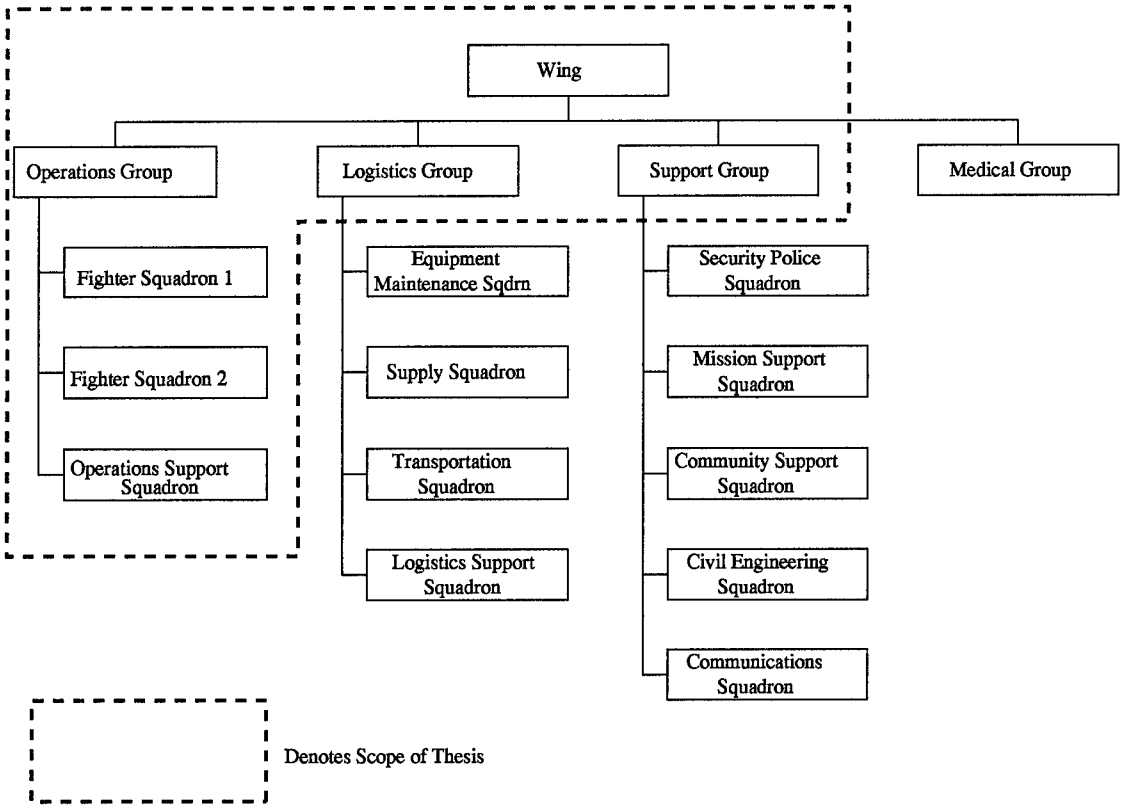


Figure 2. Typical Wing Organizational Chart (7:8)

1.4 Assumptions

Prior to conducting this research effort it was necessary to make a few assumptions. First, it was assumed that domain experts would be available to support the domain analysis process and verify domain model results. It was also assumed that the research

sponsor would provide access to valid and up-to-date information on past, present, and future C2 domain analysis activities.

1.5 Approach

To meet the challenges and research objectives described in Section 1.2, the following tasks were accomplished:

- *Gain an Understanding of Domain Analysis* - A survey was conducted of current software engineering literature, focusing primarily on domain analysis and object-oriented analysis/design.
- *Analyze Previous C2 Work* - Past, present, and future domain analysis work in the C2 domain was examined to determine applicability to this research project and avoid any duplication of effort.
- *Define a Domain Analysis Process* - Based on knowledge gained from the previous two tasks, a specific domain analysis process was defined for this research effort.
- *Perform Domain Analysis* - This task involved performing a domain analysis of Air Force wing C2 operations and producing a corresponding domain model. Sponsor C2 experts were consulted to verify the correctness of the domain analysis results and domain model.
- *Instantiate a Domain Model* - The domain model was used as a knowledge base to build a prototype software tool.
- *Prototype Testing* - The final phase of the research was to test the prototype software tool for correct behavior.

1.6 Sequence of Presentation

The remainder of the thesis is organized as follows: Chapter II reviews current software engineering literature. Chapter III describes the domain analysis process used to analyze the Air Force wing C2 domain and presents the domain model that resulted from the domain analysis. Chapter IV contains the analysis and design of the prototype tool.

Chapter V presents the testing and validation of the prototype tool. Finally, Chapter VI presents research conclusions and recommendations for future research.

II. Literature Review

2.1 Introduction

This literature review is a survey of current software engineering literature and research in areas of software engineering technology that can be applied to the Air Force C2 domain. In particular, Section 2.2 describes domain analysis and serves as the basis for defining the domain analysis approach used in this research project. Section 2.3 presents a brief overview of the object-oriented approach, which provided the framework for developing an object model of an Air Force wing.

2.2 Domain Analysis

Domain analysis is an important step in system development. In recent years, domain analysis has received considerable attention from the software engineering community in the form of research, conferences, workshops, and technical papers. One reason for the focus on domain analysis is the increasing reliance on software by today's computer systems and the fact that software design, development, and maintenance is becoming a more complex job. This need has led researchers to find better ways to engineer modern software systems.

2.2.1 Definition of Domain Analysis. Examining the work of researchers produces a variety of definitions for domain analysis. Neighbors first introduced the concept in 1980 as "the activity of identifying objects and operations of a class of similar systems in a particular problem domain" (12). Similarly, Ogush and Prieto-Díaz define domain analysis as "the process by which information used in software systems is identified, captured and organized with the purpose of making it reusable when creating new systems"

(13, 15). Further, Lowry describes domain analysis as "a form of knowledge acquisition in which concepts and goals of an application domain are analyzed and then formalized in an application oriented language suitable for expressing software specifications" (8:648).

The common theme of these definitions is that domain analysis is a way to capture the knowledge about key objects, operations, and relationships relevant to a particular domain. Most often, this knowledge is maintained in a domain model which is an abstraction and encapsulation of domain information into a predefined knowledge structure.

2.2.2 Approaches to Domain Analysis. With definitions of domain analysis in hand, the next step is to understand the various approaches to applying domain analysis. Prieto-Díaz, in his paper *Domain Analysis for Reusability*, examined different domain analysis approaches. The results of his analysis are shown in Figure 3.

Raytheon	CAMP	McCain	Arango
Identify commom functions Group by classes Organize into library Analyze business system structures Identify common structures Abstract structures Build objects	Identify similar systems Decompose functionally Abstract functionally Define interfaces Encapsulate results	Define reusable entities Define resuable abstractions Classify abstractions Encapsulate results	Bound domain Collect examples Identify abstractions Formalize abstractions Classify abstractions Encapsulate results

Figure 3. Domain Analysis Methods Compared by Prieto-Díaz (22:8)

Using common elements from these various approaches Prieto-Díaz developed the following domain analysis process, which is also represented in Figure 4:

1. Pre-Domain Analysis
   (a) Define the Domain
   (b) Scope the Domain

(c) Identify Sources of Domain Knowledge

(d) Define Domain Analysis Goals and Guidelines

2. Domain Analysis

(a) Identify Objects and Operations

(b) Abstract the Objects and Operations

(c) Classify the Abstracted Objects and Operations

3. Post-Domain Analysis

(a) Encapsulate the Classified Objects and Operations

(b) Produce Reusability Guidelines (16)

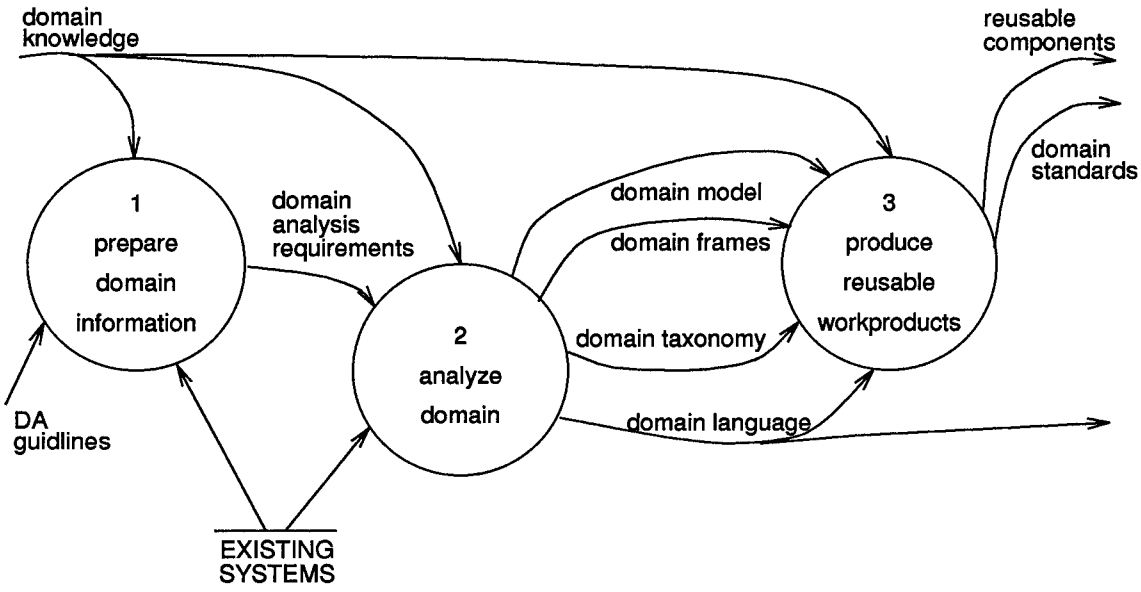


Figure 4. Domain Analysis Approach Proposed Prieto-Díaz (16:67)

The central activity of Prieto-Díaz's approach to domain analysis (Analyze Domain) is shown in Figure 5. In his approach, the domain analyst — with the assistance of domain experts — identifies and documents key pieces of domain knowledge. Sources of domain knowledge, for instance, include technical experts, documentation, requirements, and similar systems. The domain analyst organizes the collected domain knowledge into a usable form like a domain model, consisting of a domain taxonomy and a domain language that can be used as a specification language to construct new systems (16).

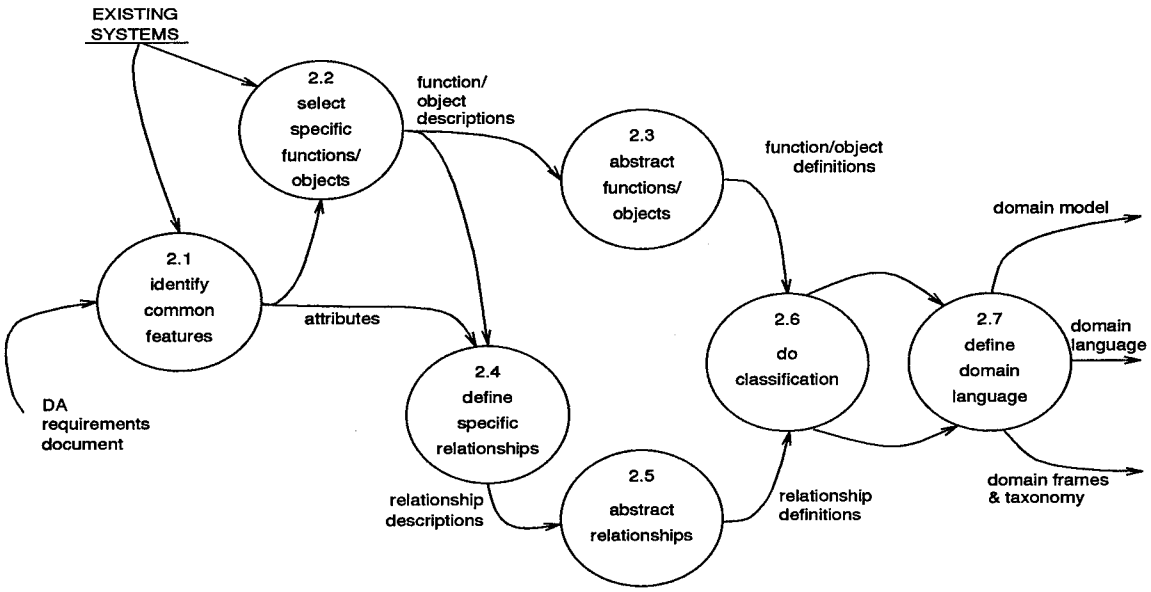


Figure 5. Prieto-Díaz's Concept to Analyze Domains (16:67)

McCain's approach to domain analysis consists of two stages. The first stage is an *application domain analysis*, which identifies the components to be implemented and their associated classification. This stage consists of the following tasks: define reusable entities, define reusable abstractions, and perform classification of reusable abstractions. The second stage is a *component domain analysis*, which is used to develop the component implementation requirements. This stage consists of the following tasks: define abstract interface specification, perform constraint analysis to minimize abstraction constraints, define applicable algorithms, and define customization requirements (9). McCain's approach is different from other approaches presented here because of his explicit inclusion of the component analysis stage (21).

Simos of the Domain Analysis Working Group proposes the following nine step domain analysis process:

1. Define domain analysis
2. Identify and scope the domain

3. Select a representative set of systems to study
4. Gather inputs for the domain analysis
5. Perform feature analysis at the requirements level
6. Analyze separability, selectability, and trade-offs of features
7. Select an implementation technology
8. Implement and validate products in phases
9. Deliver products of domain analysis (20)

Prieto-Díaz, at the same working group, proposed a somewhat different approach:

1. Acquire knowledge (define scope)
2. Conduct high-level functional analysis (top down)
3. Identify objects and operators (bottom up)
4. Define domain models (map objectives into architecture) (20)

The Working Group compared these two approaches with Wirfs-Brock's object-oriented design approach and Motta's knowledge elicitation process and found that all processes end up with a design model and are for the most part similar (20).

Iscoe, with a different approach to domain analysis, focuses on the product of domain analysis as opposed to the actual approach or its inputs. He suggests the problem is "to create a model for domain knowledge that is general enough to be instantiated in several domains" (5:299). His approach involves developing levels of "meta-models" that a domain analyst can use to capture the information of a particular application domain (5). Iscoe's approach to domain analysis is known as domain modeling (21).

2.3 Object-Oriented Approach and Modeling

An object-oriented modeling approach is based on the notion of an object. Objects consist of data structures (i.e., attributes) and behaviors (i.e., methods) in a single entity.

Object-oriented technology differs from conventional programming approaches where data structures and behaviors are only loosely connected. The main benefits of object-oriented analysis and design are that they promote software reuse, reduce the potential for errors, and simplify software maintenance (18). According to Pressman

> The unique nature of object-oriented design (OOD) lies in its ability to build upon three important software design concepts: abstraction, information hiding, and modularity. All design methods strive for software that exhibits these fundamental characteristics but only OOD provides a mechanism that enables the designer to achieve all three without complexity or compromise (14:334).

Other benefits of an object-oriented approach are the ability to better model a particular domain and to create models that include both physical and behavioral characteristics (2).

2.3.1 Definition of Terms. The following is a list of definitions for the major themes in object-oriented modeling.

- Abstraction – focusing on essential aspects of an object while disregarding levels of detail not pertinent to the particular problem. It also means focusing on what an object is and what it does without making premature design decisions (18:7).
- Encapsulation – Separating the external aspects of an object from internal implementation details of the object. This allows implementations of objects to be changed without affecting the applications that use it. Encapsulation is also referred to as information hiding (18:7).
- Object Model – Describes the static structure of objects and their associations/relationships with other objects. Object models are usually represented in a graphical form like an object diagram (18:6).
- Object Diagram – a formal graphical notation for modeling objects, object classes, and their relationships to one another (18:23).
- Object Class – A description of a group of objects with similar properties (attributes), common behavior (operations), common relationships to other objects, and common semantics (18:22).
- Instance – A single instantiation of an object. Each instance has its own attribute values and relationships to instances of other objects (1).

- Attribute – A data value (not an object) held by the objects in a class (e.g., *name, age,* and *weight* are attributes of *Person* objects, and *color, weight,* and *model-year* are attributes of *Car* objects) (18:23).
- Association – A physical or conceptual relationship between object classes. For example, a person *works-for* a company (18:27).
- Multiplicity – Specifies how many instances of one class may relate to each instance of another class in an association (See Figure 6) (18:30).
- Aggregation – The "*part-whole*" or "*a-part-of*" relationship in which objects representing components of something are associated with an object representing the entire assembly (18:36). This relationship is also referred to as an "*is-composed-of*" relationship (4).
- Inheritance – An abstraction for sharing similarities among classes while preserving their differences. It is the relationship between a class and one or more further refined versions of it. The class being refined is called the *superclass* and each refined version is called a *subclass.* Inheritance is sometimes called the "*is-a*" relationship because each instance of a subclass is an instance of the superclass as well. The most important use of inheritance is the conceptual simplification that comes from reducing the number of independent features in a system (18:38–43). For example, *Airplane* is the superclass of *F-16* and *B-52.* Each subclass *inherits* the features (attributes, operations, and relationships) of its superclass. For example, *B-52* inherits attributes *manufacturer, weight,* and *wing-span* from *Airplane* as well as the operations *takeoff* and *land* (1).

2.3.2 Object Model Notation. The object diagram notation selected for this thesis is Rumbaugh's Object Modeling Technique (OMT) (18). Figure 6 provides the subset of OMT notation used to develop the Air Force wing C2 object diagrams. Note that an object is represented as a rectangle containing the class name and optionally the attributes and/or operations defined for the class.

2.4 Summary

There is no single best approach to domain analysis or object-oriented modeling; each approach has its own merits. What is important is that the approaches to domain analysis and object-oriented modeling presented in this chapter provided the necessary

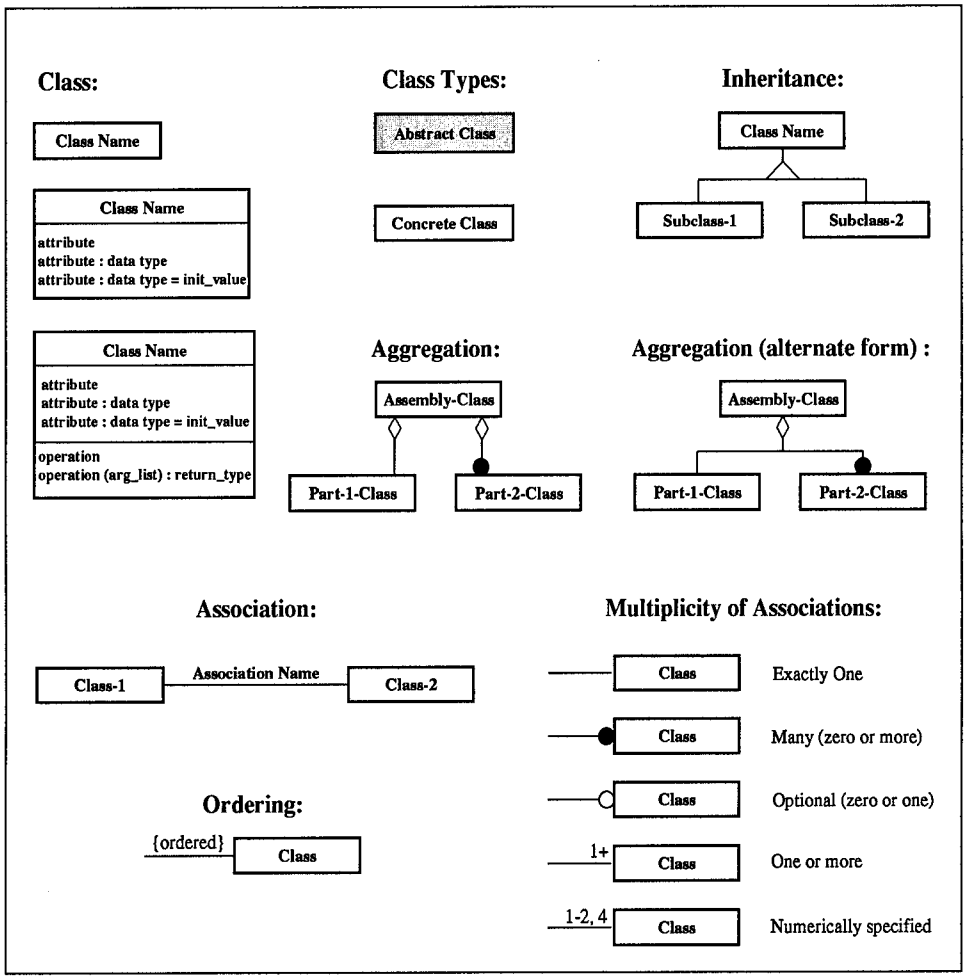


Figure 6. Object Modeling Notation (1:2–6)

foundation to begin the requirements analysis (i.e., domain analysis) of Air Force wing C2. This analysis is the topic of the next chapter.

III. Domain Analysis

3.1 Introduction

Chapter II characterized domain analysis as a way to capture the knowledge about key objects, operations, and relationships relevant to a particular domain. This chapter describes the process followed to conduct a domain analysis of Air Force wing C2, resulting in a domain model of Air Force wing C2 operations. As stated in Chapter II, there is no definitive approach to conducting domain analysis. In fact, even the same analysts, such as Prieto-Díaz, sometimes have different approaches. The approach chosen for this research effort combines techniques from the various approaches studied.

3.2 Domain Analysis of Air Force Wing C2 Operations

The approach used in this research combines ideas from Prieto-Díaz, Tracz, and others. The specific steps in this process are:

1. Define the Domain
2. Scope the Domain
3. Identify Sources of Domain Knowledge
4. Obtain Domain Knowledge
5. Choose Model Representation
6. Develop Domain Model

3.2.1 Define the Domain. The first step in performing domain analysis is defining and naming the domain of interest. Defining the domain can be accomplished by analyzing project requirements. One must understand the project goal before one can define the

domain of interest. It is important not to be too restrictive in defining the domain of interest. The next step will deal with scoping the domain defined here (16:67).

The domain of interest for this project has been defined by research requirements from the sponsor; the domain of interest is Air Force C2 activities. A typical Air Force wing is depicted in Figure 2. A top-level view of wing operations is depicted in Figure 7. These diagrams describe the initial domain of interest.

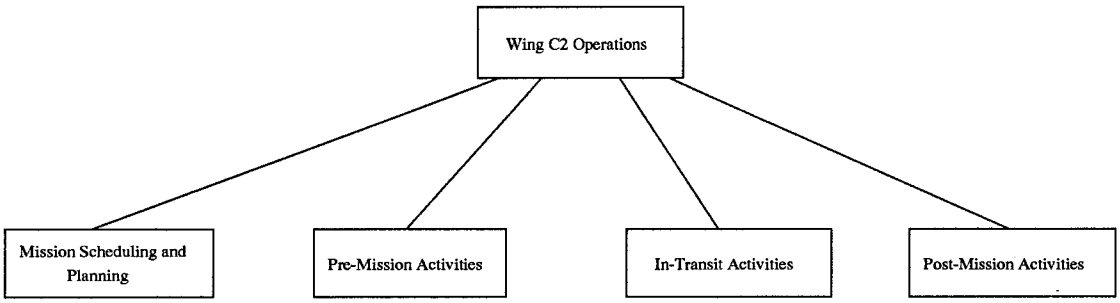


Figure 7. Breakdown of Wing Operations for ATO Processing

3.2.2 Scope the Domain. A domain can be defined at a great level of detail or it can be more abstract. In addition, it is often unclear where one domain ends and another begins. During this step, the domain analyst must place specific bounds on the domain of interest. The goals of domain analysis may place constraints on the domain and help the analyst find the proper scope of the domain (22:21).

For this research, the initial domain of wing C2 has been narrowed in several areas. The primary interest is in how the Air Tasking Order (ATO) affects the wing mission and what effects automation tools have on mission execution. Wings are typically tasked to do several missions, but this research is not concerned with those missions not related to the ATO. Also, as previously stated in Section 1.3 and illustrated by Figure 2, the domain

scope has been narrowed to just the Operations Group level, since the major portion of the wing mission tasked via the ATO is accomplished at this level. The other portions of a wing will either be modeled at a more abstract level or not at all.

Another scoping issue was that this research was to model the performance issues of the various tasks that comprise the processing of an ATO. Modeling or simulating the actual tasks in processing an ATO was outside the scope of this research.

3.2.3 Identify Sources of Domain Knowledge. The domain analyst is not always an expert in the domain at hand. In such a case, the analyst must find sources from which to obtain the knowledge necessary to reason about the subject domain. These resources may include domain experts, domain documentation, or previous domain research (16:67).

For this research, assistance has been provided by domain experts from HQ PACAF, the sponsor, as well as from the 89th Fighter Squadron (AFRES) at Wright-Patterson AFB, OH. Valuable domain information has also been obtained from many pieces of documentation, including (10), (11), and (17).

3.2.4 Obtain Domain Knowledge. Once sources of domain knowledge have been identified, one must start the process of collecting and analyzing domain knowledge. Whether from interviewing domain experts or studying domain documentation, the analyst must record the key information obtained about the domain and organize this information in a meaningful, logical way (22:23).

For this research, initial domain knowledge was obtained by studying the research proposal, which was provided by the research sponsor. Correspondence and meetings

with the sponsor helped clarify issues. After receiving and studying many pieces of C4I documentation, meetings were held with another domain expert. This was an iterative process and continued until adequate knowledge of the domain was obtained.

3.2.5 Choose Model Representation. The domain analysis process will result in a domain model. Therefore, it is important to choose a model representation up front. Modeling techniques range from informal to formal representations and can be object or functionally oriented. Models can be merely paper studies or can be fully automated.

An object-oriented approach for modeling the wing C2 domain was used for this research. The approach selected was based on Rumbaugh's OMT (18) and is briefly described in Section 2.3. The domain model developed from this analysis contains two components: the object model and the functional model. The development of a separate dynamic model as part of the overall domain model was not necessary because the event and state information pertinent to the wing C2 domain was incorporated into the functional model.

The object model is used to represent the static structure of the domain. In the C2 domain, for example, the object model describes how the wing and its components are organized as well as identifying the associations or relationships between these entities. The functional model, on the other hand, describes the processes, operations, or functions the objects from the object diagram perform. In the case of the Air Force C2 domain, the functional model shows the various steps in processing an ATO and the inputs, outputs, and data stores relevant to each task. From the functional model of the C2 domain an analyst can also understand the precedences and dependencies among the various tasks, providing insight into the issue of sequential vs. concurrent tasks.

3.2.6 Develop the Domain Model. As described in the previous section, a two part domain model was developed for this research. The following sections discuss the object model development and functional model development in greater detail.

3.2.6.1 Object Model. In object model development, the analyst looks for specific entities in the scoped domain and defines them as classes of objects. **Aircraft** is an example of an object class. The use of the term object is often ambiguous in the literature. For this research, unless otherwise noted, the term object will mean object class.

The analyst must give a description of the object, as well as define the attributes an object possesses. For example, the object **Aircraft** has attributes such as **Type**, **Tail Number**, and **Fuel Capacity**. The analyst must also determine the interrelationships or associations between objects. For example, **Maintains** is an association between a **Maintenance Squadron** class and an **Aircraft** class. The analyst must also define operations that are applied to or performed by the objects defined in the previous step. For example, **Refuel** is an operation for the **Aircraft** object. With knowledge of the objects, attributes, associations, and operations, the analyst can develop the object model.

For this research, object model development began with analysis of (10), (11), (6), (17), and (7). In particular, process diagrams and context diagrams were reviewed, noting all inputs, outputs, and controls. Additionally, the data dictionaries in (10) and (11) helped to further refine the attributes of the initial set of objects. Studying the documentation and interviewing the domain experts helped define the relationships or associations between various objects. Operations for each of the objects identified in the previous steps were defined by studying (10), (11), and (6), and the functions listed in (6) were mapped to the

objects performing the function. Similarly, processes from the context diagrams of (10) and (11) were mapped to specific objects.

The preliminary object model was drafted based on the knowledge and understanding of the objects, attributes, associations, and operations. Refinement of the object model was an iterative process, as with most analysis and design projects. Feedback from the research sponsor and domain experts was key to finalizing the object model.

Figures 8, 9, 10, and 11 are the results of the object model development and are discussed below. The reader is referred to the Data Dictionary in Appendix A for more detailed definitions of the objects shown in these figures.

Figure 8 shows the major object classes of the C2 operations domain — **C2 Task** and **Wing/Wing Operations Center (WOC)** — and how they fit together. The **ATO Task** and **Non-ATO Task** are specializations of the **C2 Task**. The reason for this distinction is because this research is only concerned with the tasks required to process an ATO (see Section 3.2.2). The **Wing/Wing Operations Center (WOC)** is composed of a **Battle Staff, Mission Planning Cell (MPC)**, and **SRC Cell**, which are composed of **Wing Personnel**. The **tasked** association indicates that a **Wing/Wing Operations Center (WOC)** can be assigned multiple **C2 Tasks** (i.e., ATO). The attributes of the **tasked** association, Time To Complete and Overall Accuracy, are defined as follows:

1. Time To Complete - A measure of how long it takes to complete processing of an ATO.
2. Overall Accuracy - A measure of how error-free the products of the ATO are.

As its name indicates, the association **assigned** shows that **Wing Personnel** can be assigned multiple **C2 Tasks**. The attributes of the association **assigned** — Time and Accuracy and — are defined as follows:

1. Time - A measure of how long it takes to complete a task.
2. Accuracy - A measure of how error-free the product of a task is.

The attributes for the **tasked** and **assigned** associations were selected after detailed analysis of the C2 domain and represent the key measures for quantifying **C2 Task** performance. The values of these attributes are derived from the characteristics of their associative objects.

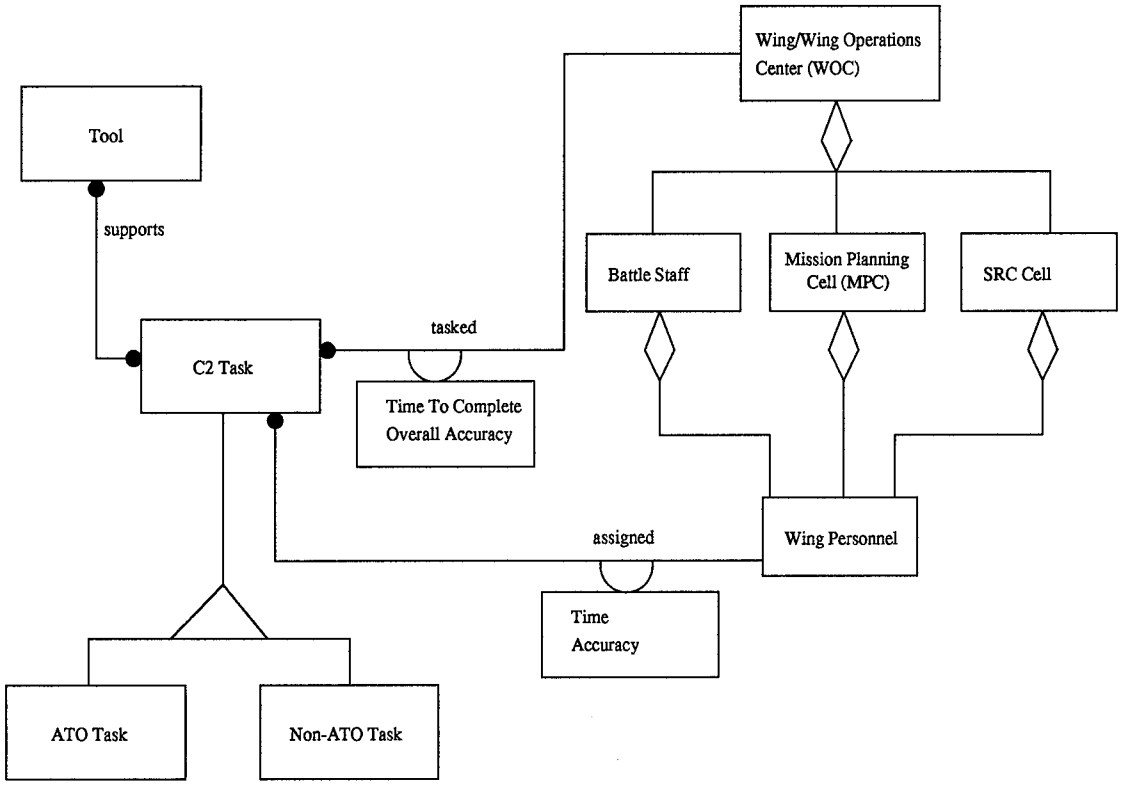


Figure 8. Top-Level Perspective of Object Model

Figure 9 shows the major sub-tasks that are specializations of an **ATO Task** and the **Mission Scheduling and Planning** task. Along with the hierarchy of tasks are the common attributes for all ATO tasks. The attributes were defined by abstracting out the key characteristics of the **ATO Task** class. These attributes are at the heart of understanding the details of the **ATO Task** class.

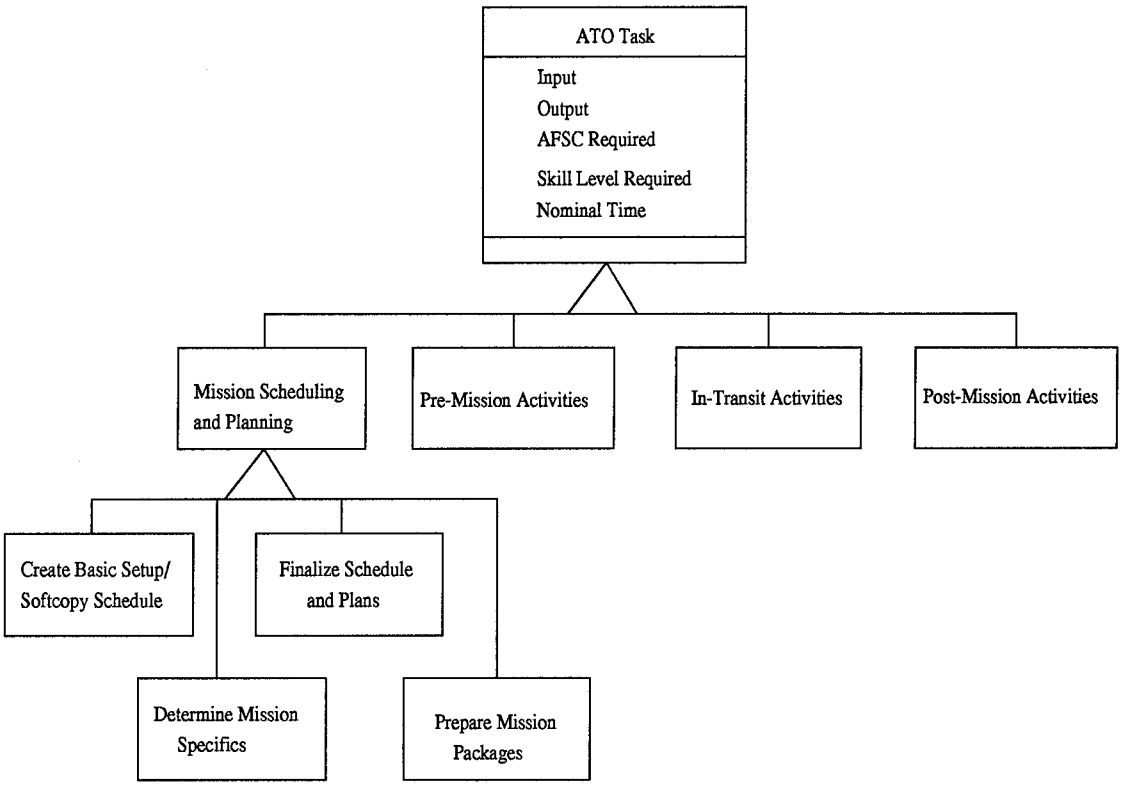


Figure 9. Task Portion of Object Model

Figures 10 and 11 are both object models of an Air Force wing but are developed from different points of view. First, Figure 10 shows the wing from an organizational perspective, more like an organizational chart, that is typical of peacetime operations. It shows that the wing is composed of groups, which are in turn composed of squadrons. Figure 11, on the other hand, shows how the wing is organized for its war-fighting mis-

sion. The people from the various organizational components join forces to staff a WOC — the hub for conducting all wing-level airborne operations. For example, Figure 10 indicates that **Weather**, **Weapons**, and **Intelligence** personnel are assigned to the **Support Squadron** of the **Operations Group** for peacetime activities. However, when the wing prepares for combat operations, these people become part of the **Mission Planning Cell** and focus on supporting wing-level activities as opposed to squadron-level activities.

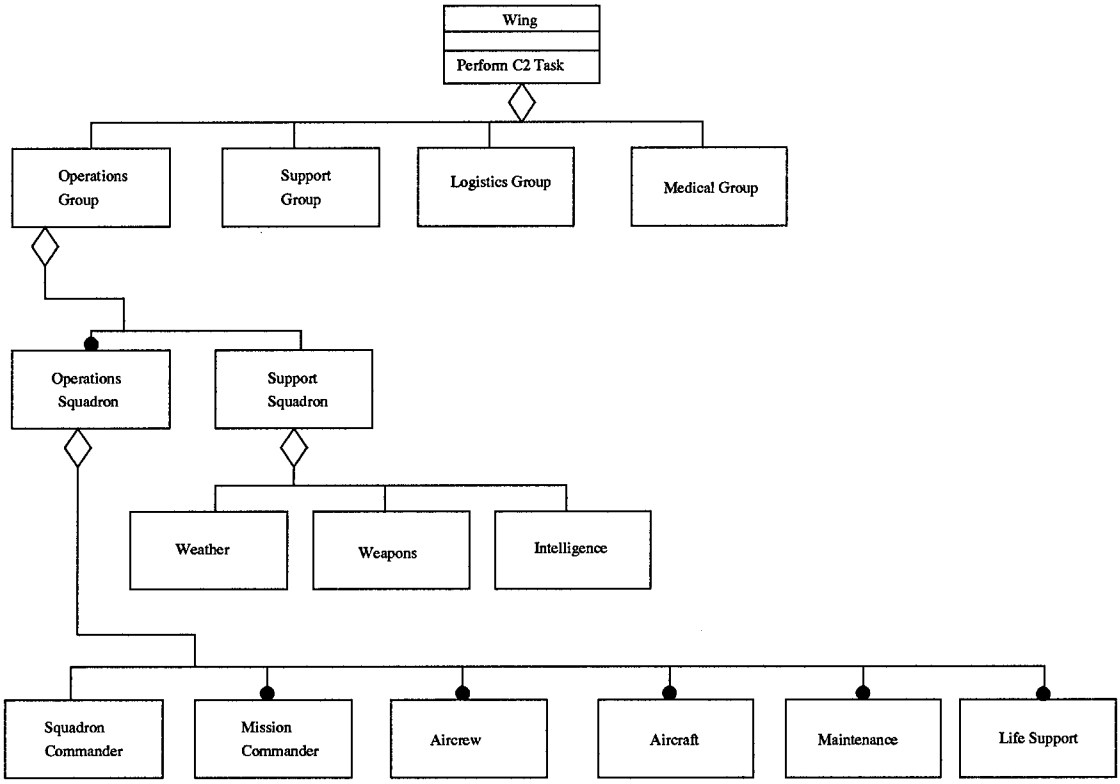


Figure 10. Wing Organization — Peacetime Perspective

3.2.6.2 Functional Model. This phase of the domain analysis process was devoted to documenting the behavior associated with Air Force wing C2 operations. Specifically, this research focused on the tasks associated with how a wing processes the ATO and who performs those tasks. The object model is an important piece of the domain

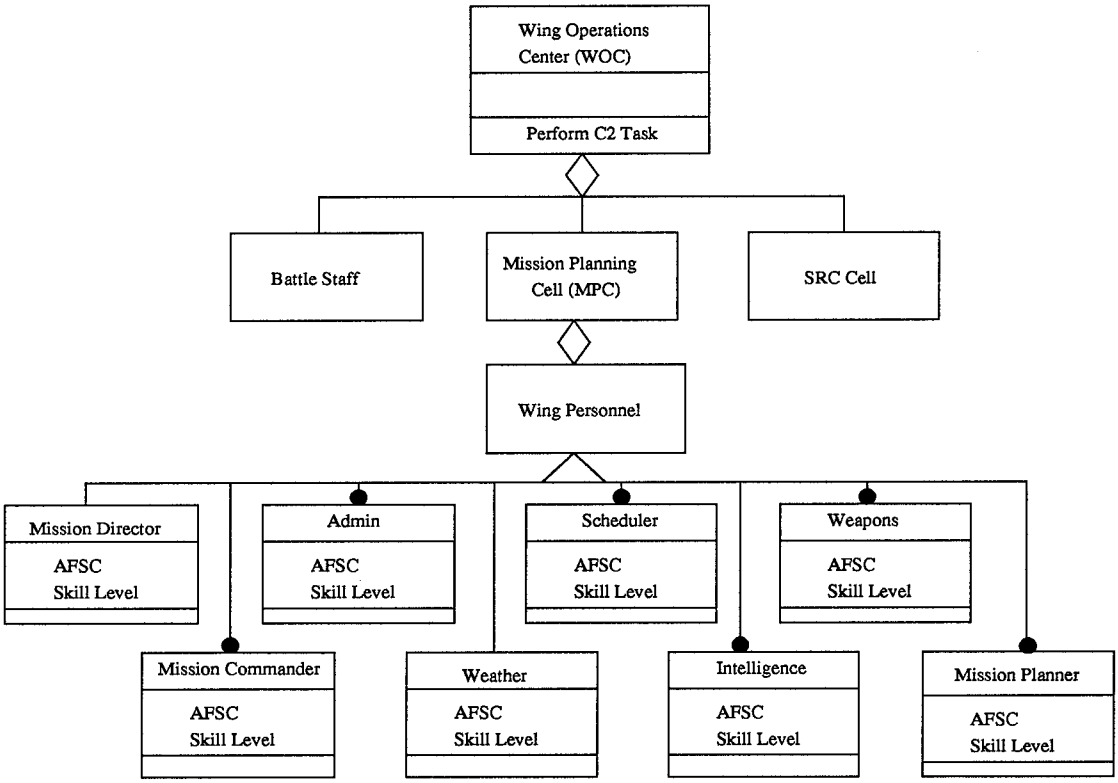


Figure 11. Wing Organization — Wartime Perspective

model because it represents who the players or "doers" are. However, without behaviors for the wing C2 objects the domain model would be inadequate for system development or any form of decision making support.

A major portion of the effort in this phase was the abstraction of various sources of mission-specific wing information (i.e., F-15 and F-16) into a more generic model of Air Force wing C2 operations. In the case of the F-15 and F-16 wings, the basic structure of tasks was the same (see Figure 7). However, further decomposition yielded numerous differences that needed to be resolved. For example, in the case of the F-16 wing, **Determine Mission Specifics** consists of the following tasks (11):

1. Obtain Intelligence
2. Obtain Weather Information
3. Analyze Threat to Mission
4. Coordinate With EW
5. Plan Flight Path/Tactics
    (a) Plot Route and Threats
    (b) Assess Air Support Assigned
    (c) Coordinate with Support Wings
    (d) Plan Route (Initial Point to Target Run)
    (e) Determine Fusing Delays

For the F-15 wing, **Determine Mission Specifics** consists of these tasks (10):

1. Determine Weapons Package
2. Coordinate with Support Wings
3. Determine Tactics
4. Obtain Weather Report

After discussions with the research sponsor and domain experts, the slight variations between the F-15 and F-16 wings were abstracted out to produce a generic model of **Determine Mission Specifics** (see Figure 15). This type of discrepancy (i.e., semantic differences, different interpretations, etc.) between various sources of domain information is quite common, and abstracting out the unnecessary details to produce a common, potentially reusable model is at the heart of domain analysis.

Another major goal in this phase of the analysis was to document the dependencies and precedences among the various tasks. The dependencies and precedences were based on the inputs and outputs of the individual tasks. This led to an understanding of required sequentiality or potential concurrency of the tasks in the scenario.

Figures 12, 13, 14, 15, and 16 depict the behavior associated with Air Force wing C2 operations and make up the functional model. The individual figures are discussed below. The reader is referred to the Data Dictionary in Appendix A for more detailed definitions of the data flows used in these figures.

Figure 12 shows wing C2 operations from the top level. The inputs are the **ATO FRAG** (which is the portion of the ATO that pertains directly to an individual wing) and **Airspace Information**, which includes intelligence information and weather information. The **ATO** and **Airspace Information** are modeled as data stores because they are passive (i.e., they respond to requests to access data) and generate no operations on their own. The **Wing** provides the resources (i.e., manpower, facilities, and equipment) to conduct C2 operations. The **Air Operations Center (AOC)** is responsible for all air operations

within the theater or campaign. As such, the **AOC** has multiple **Wings** it can task to perform specific portions of the **ATO**.

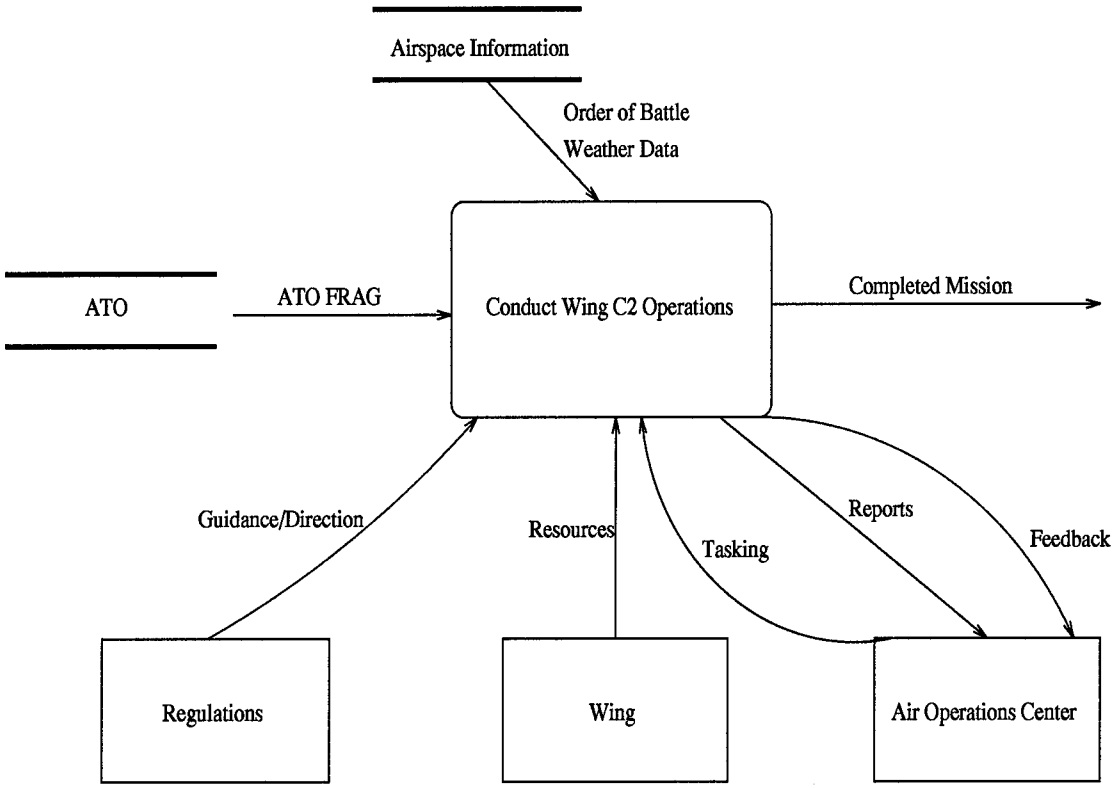


Figure 12. Functional Model — Air Force Wing C2 Operations

Figure 13 shows how **Conduct Wing C2 Operations** from the previous figure is broken down into the major elements required to process an ATO. **Prepare Mission Plans and Schedules** is the task where the majority of planning and scheduling takes place. This task is concerned with issues ranging from making sure the wing has the resources to complete the assigned tasking to planning/plotting the actual flight route and determining the tactics used to fly the mission. The outputs of this process are **Mission Packages**, which are packages of information containing every aspect of the mission from take off times to ingress/egress routes, and **Schedules**, which are detailed timelines of the

events to occur. **Conduct Pre-Mission Activities** is the activity of physically preparing the aircraft (i.e., loading munitions and fueling) and making sure the aircrews are fully briefed on the upcoming mission. The output of this process is a **Launched Mission**. **Manage Mission and Base Support** deals with all aspects of managing the mission once it's underway. For example, if friendly aircraft are shot down, a search-and-rescue mission must be launched. The output of this task is a **Recovered Mission** and occurs when the aircrews have returned to base. The final process, **Conduct Post-Mission Activities**, is concerned with debriefing the personnel involved in the mission and making sure the results of the mission are documented, reported to the AOC, and incorporated into future ATOs.

Through the functional model analysis, it was discovered that the majority of analysis, planning, and scheduling tasks reside in **Prepare Mission Plans and Schedules**. For this reason, it was decided that this research should concentrate here. The next paragraph discusses **Prepare Mission Plans and Schedules** in greater detail.

Figure 14 shows how **Prepare Mission Plans and Schedules** from Figure 13 is divided into more detailed sub-tasks. **Create Basic Setup/Softcopy Schedule** is the process that creates the initial mission schedule based on aircraft/aircrew availability and establishes the mission timeline. **Determine Mission Specifics** does exactly what its name implies. **Mission Specifics** include items like EW Guidance, Tactics, and Route Map, for example. **Finalize Schedules and Plans** transforms the **Softcopy Schedule** and **Mission Specifics** into the final **Schedules**. Minor adjustments to the **Softcopy Schedule** are made once the **Mission Specifics** have been determined. Another aspect of this process is the coordination of the final **Schedules** with the **AOC**. The final process

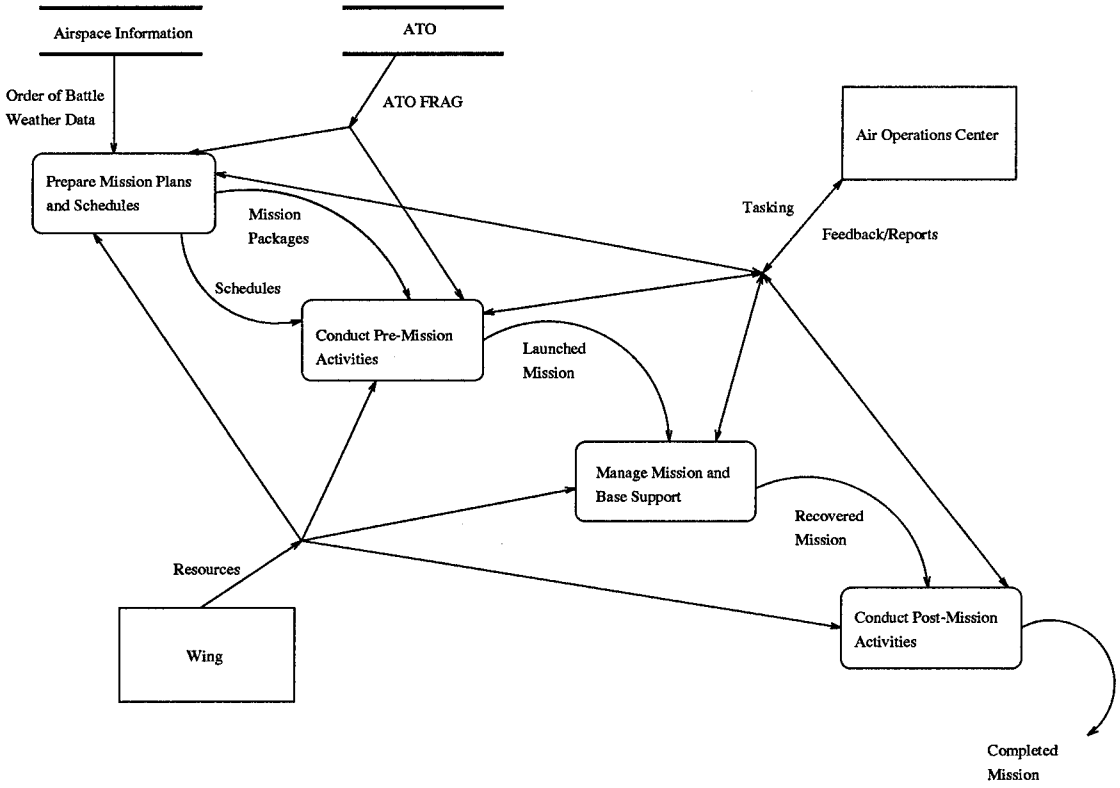


Figure 13. Functional Model — Conduct Wing C2 Operations

in Mission Scheduling and Planning is **Prepare Mission Packages** where the **Schedules** and **Mission Specifics** are used to build the **Mission Packages. Mission Packages** contain the information the pilot needs to fly the mission: maps, search-and-rescue information, communication frequencies, special instructions, code words, target imagery, altitudes, kill boxes, and the information for the flight computer, which is found in the Data Transfer Cartridge (DTC).

Similar to the reasons given above as to why this research focused on **Prepare Mission Plans and Schedule**, it was further discovered that the majority of analytical tasks reside in **Determine Mission Specifics**. For this reason, it was decided this research should further narrow in on this process. The next paragraph discusses **Determine Mission Specifics** in greater detail.

Figure 15 is a highly detailed view of **Determine Mission Specifics** and is the portion of the ATO process used to develop the prototype reasoning system, which is discussed in later chapters. The figure shows how the inputs are transformed into the various components that comprise Mission Specifics.

Figure 16 is a Activity on Node diagram of **Determine Mission Specifics** and shows the precedences and dependencies among the various sub-tasks more clearly than Figure 15.

3.3 Summary

This chapter has described the approach and results for the domain analysis of Air Force wing C2 operations. The domain model — consisting of the object model and

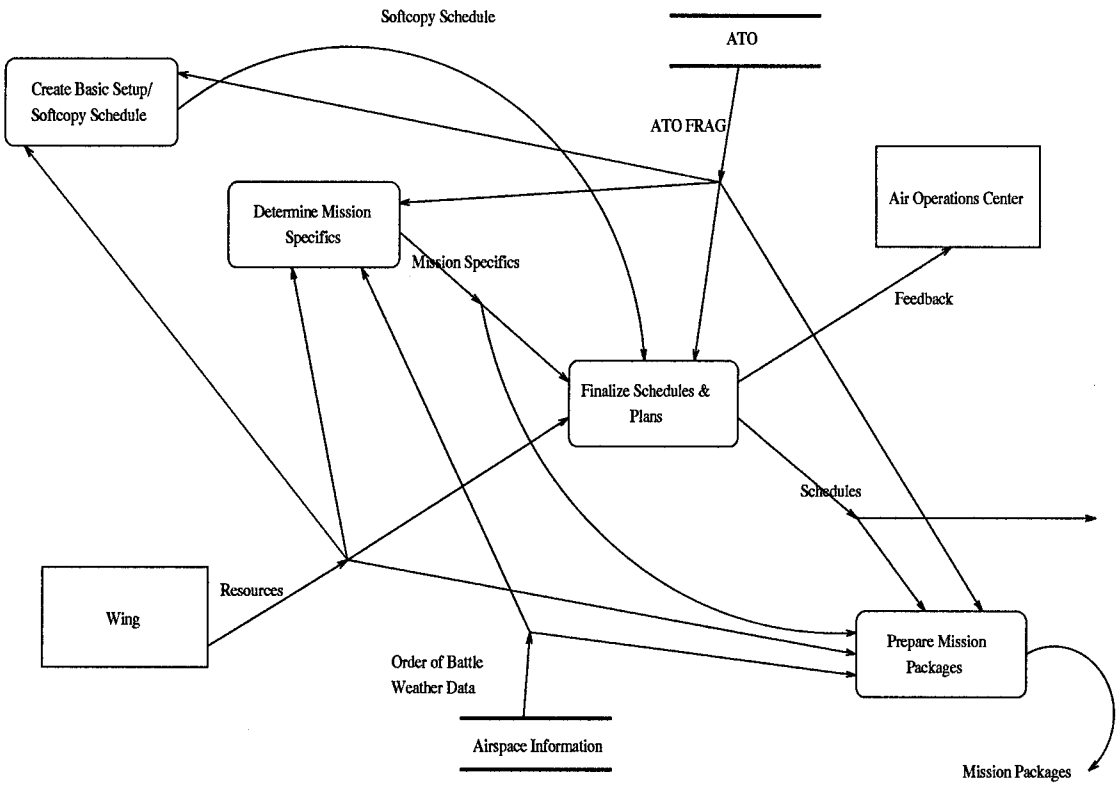


Figure 14. Functional Model — Prepare Mission Plans and Schedules

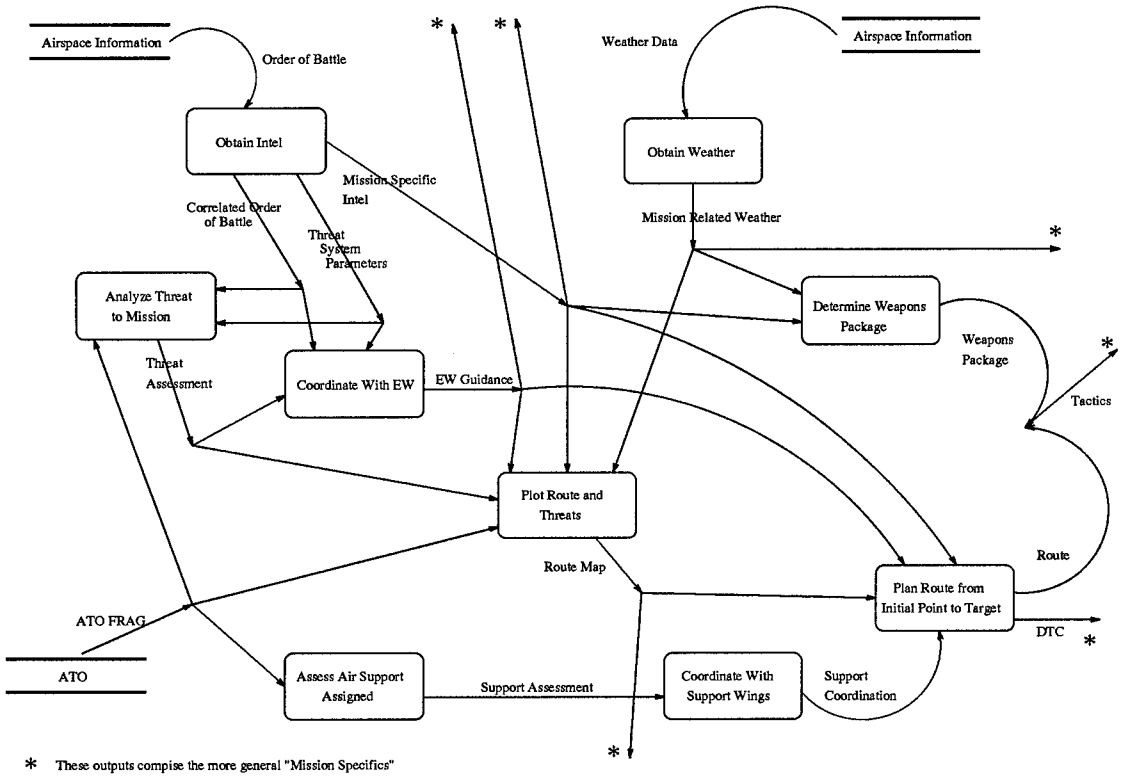


Figure 15. Functional Model — Determine Mission Specifics

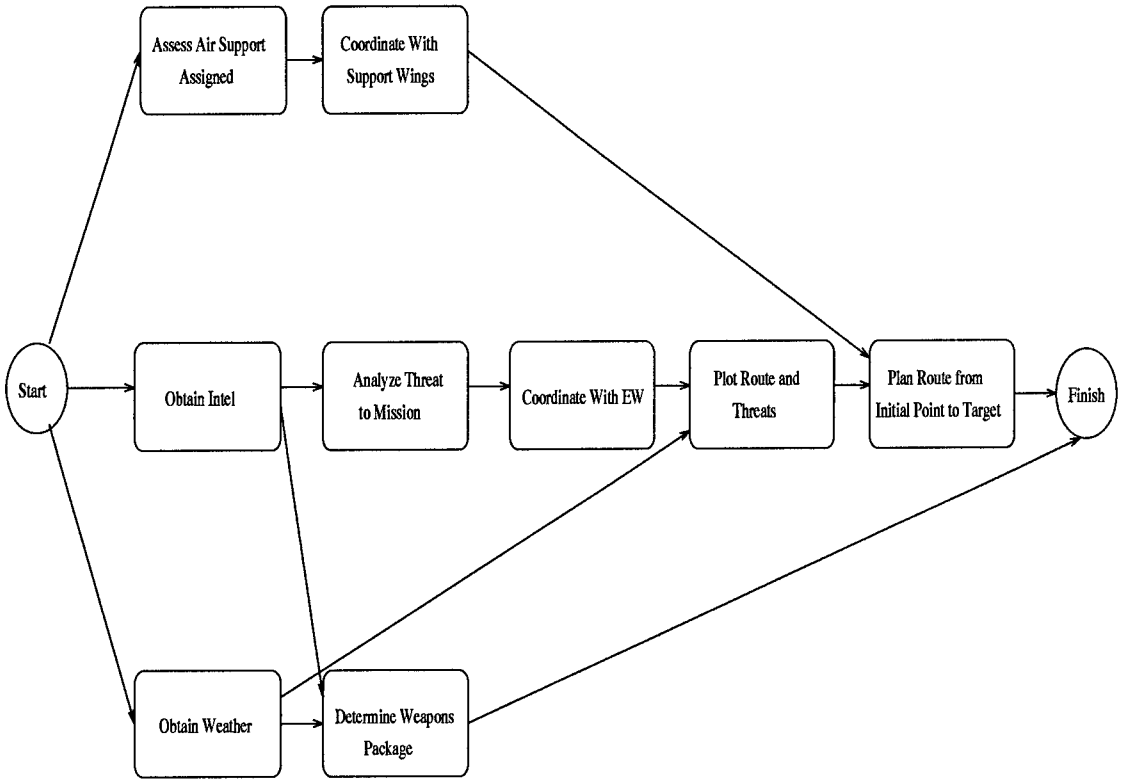


Figure 16. Dependency-Based Representation of Determine Mission Specifics

functional model — served as the knowledge base to analyze and design a prototype of the automated reasoning system, which is discussed in Chapter IV.

IV. Prototype Analysis and Design

4.1 Introduction

The domain model of Air Force wing C2 operations, discussed in Chapter III, along with sponsor research requirements provided the framework for developing the resource allocation tool. As discussed in Section 1.2, the sponsor needed decision making support for allocating resources and determining the effects of automation. To that end, analysis of the resource allocation needs yielded the object model shown in Figure 17.

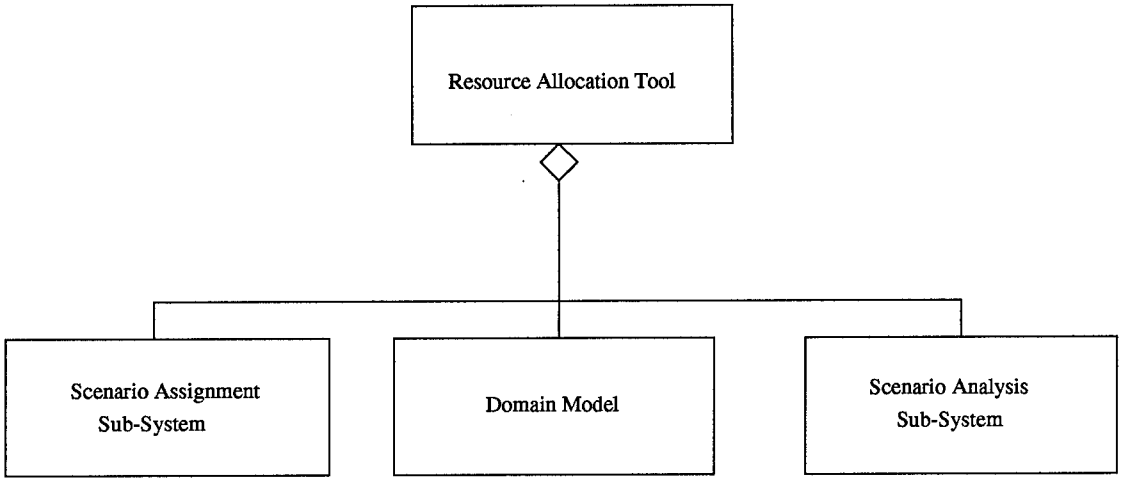


Figure 17. Object Model of Resource Allocation Tool

The Scenario Assignment Sub-System provides the wing user a means to assign personnel from the wing, as well as automation tools, to the various tasks that comprise the ATO process and provides an assessment for each doer/task pairing. The Scenario Analysis Sub-System, on the other hand, uses the doer/task pairings and individual assignment assessments to produce an overall assessment and the necessary feedback to give the wing user insight into how well he/she allocated the resources and if automation provided any value. Both sub-systems are based on information from the domain model. The design

and analysis of the Scenario Analysis Sub-System is discussed in Sarchet's thesis (19). The remainder of this chapter discusses the analysis and design of the Scenario Assignment Sub-System.

4.2 Analysis

The goal of this analysis is to determine the requirements for the Scenario Assignment Sub-System and define an initial object structure which can achieve these requirements. This analysis will state the capabilities of the Scenario Assignment Sub-System — what it must do — without stating how it will accomplish them.

4.2.1 Requirements. The primary requirement of the Scenario Assignment Sub-System is to allow the user to generate assignment pairs given a set of tasks to perform (a scenario), and a list of persons capable of performing these tasks. When combined with the Scenario Analysis Sub-System, the fulfillment of this requirement will achieve the goals set out in Chapter I. It will also give the sponsor a decision support system capable of helping the decision maker make the best possible assignments. This primary requirement is broken down into the following sub-requirements which are necessary steps in fulfilling the overall goal of the sub-system.

- Load a set of tasks to be performed.
- Load a set of persons who can perform tasks.
- Load a set of automated tools capable of supporting tasks.
- Determine the predecessor tasks for a given task.
- Display information about the current task.
- Display information about who is qualified to perform the current task.

- Display a list of automated support tools that can assist with the current task
- Allow the user to select a person to perform the current task.
- Allow the user to select an automated support tool to assist in the current task
- Based on user selections, calculate the time required to perform the current task by the selected person assisted by the selected tool.
- Based on user selections, calculate the accuracy of the task as performed by the selected person assisted by the selected tool.
- Store the assignment information, to include all task-person pairings with associated time and accuracy figures, in a structure to be made available to the Scenario Analysis Sub-System.

Some of these requirements are now described in greater detail. Predecessor tasks are defined as the set of tasks which must be completed before the given task can be accomplished. For any given task, this subsystem will be able to determine the immediate predecessor tasks by matching the given task's inputs to the outputs of other tasks. This will be important information for the Scenario Analysis Sub-System, allowing it to determine which tasks may be performed concurrently and which must be performed sequentially.

A user interface will display information about the current task in a scenario and a list of candidates who can perform the task. Enough information must be displayed so that the user can make an intelligent assignment. Task information should include the name of the task, the type of person who can best perform the task, the desired skill level for a person performing the task, and the average time for accomplishing the task. The type of person best suited for a task can be signified by an Air Force Specialty Code (AFSC). Personnel information should include the name, AFSC, and skill level of the person. While the user will be allowed to select a person who does not possess the desired AFSC for a

task, it should be clear to the user which candidates, if any, have the desired AFSC and which do not.

The user interface will also list a set of tools which can assist in the current task. Information on tools should include the tool name, which tasks the tool can support, and the level to which a tool supports a particular task. For example, the tool "typewriter" would provide a low level of support to the task "Prepare Mission Packages," while an automated reasoning system would provide a high level of support. Only tools which support the current task will be listed. The user will be able to select any one of these tools or none at all. If no tools exist to support the current task, the user interface will state this fact. It is important to note while many tools may support a task, only one tool may be chosen by the user to assist in a particular task.

This sub-system will calculate the actual time required to perform the task as assigned. The nominal time to complete a task, the time it would take the average qualified person to perform the task, is available as an attribute of **task**. The actual time will be calculated based on this nominal time and information about the person and tool selected by the user. If the assigned person does not possess the AFSC or skill level desired by the task, the actual time will be greater than the nominal time. If the person selected has the desired AFSC and a higher skill level than the task desires, the actual time will be less than the nominal time. If a tool is assigned to assist in the task, the actual time may be less than the nominal time, based on the level of the tool's support and the skill level of the selected person. The actual calculations for determining the time to perform the assigned task will be defined during the sub-system design phase and are based on timing information furnished by domain experts.

The accuracy of the task as performed by the selected tool will be calculated. Accuracy is a measure of the correctness of the output of a task. Calculating the accuracy will be similar to calculating the time of a task. If the person assigned to a task is fully qualified, that is, the person has the desired AFSC and skill level, the accuracy will be high. The use of support tools can also increase the accuracy figure. An assigned doer with the wrong AFSC or low skill level will lead to low accuracy figures, unless supported by a high level tool. Accuracy can be measured in different ways for different tasks. For example, accuracy for the task "Fly Mission" could be measured by the percentage of bombs on target, while the accuracy for the task "Obtain Weather Info" could be measured by the timeliness of the data. This system will not define what accuracy means for each task, but will calculate a relative accuracy figure which the user may "translate" into actual results.

4.2.2 Object Identification. Based on the requirements and the domain analysis of Chapter III, the following object classes are identified for the Scenario Assignment Subsystem:

- Task
- Doer
- Tool

The object class "task" is equivalent to the "ATO task" as defined in the domain model (see Section 3.2.6.1) and illustrated in Figure 8 and Figure 9. It is not necessary for the Scenario Assignment Sub-System to create the lower level object classes under ATO task since each of these lower levels possess the same attributes and operations. Instead,

the object class task will include an attribute NAME which indicates the type of ATO task. The attributes of the object class task are:

- Name: the type of task
- AFSC: the primary AFSC required of a person to perform the task
- Skill Level: the skill level required of a person to perform the task
- Inputs: a list of items which must be available or accomplished before the task can begin
- Outputs: a list of items which are available or have been accomplished after the task is completed
- Status: whether the task has not been started, is in progress, or has completed action
- Nominal Time: the amount of time to complete the task if accomplished by a person with the desired AFSC and skill level

The object class "doer" is equivalent to the Wing Personnel object as defined in the domain model and illustrated in Figure 8 and Figure 11. Since all doers will have similar attributes and operations, it is not necessary to define separate object classes for all the classes pictured in Figure 11. The attributes of the object class doer are:

- Name: the name of the doer's position
- AFSC: the doer's Air Force Specialty Code
- Skill Level: the skill level held by the doer
- Available: whether the person is available for assignment or is currently assigned to a task

The object class "tool" is identified in the domain model and is illustrated in Figure 8. The attributes of tool are:

- Name: the name of the tool
- Available: whether the tool is available to assist in a task or is currently in use and unavailable

The following associations between object classes are also identified:

- Assists: Task → Tool
- Supports: Task ↔ Tool
- Assignment: Task → Doer
- Precedes: Task → Task

The association "assists" links a tool to a task and shows the tool that has been selected to assist the person in performing the task.

The association "supports" links a tool to a task and shows that a tool supports a task. An attribute of this association is:

- Level: indicates the level of support provided by the tool for the task. Tool level values range from 0, meaning the tool provides no support for the task, to 5, meaning the tool can basically accomplish the task accurately and quickly.

The association "assignment" links a task to a doer and shows who is assigned to what task. Attributes of this association are:

- Accuracy: a measure of the accuracy of the task results as performed by the doer. Accuracy values range from 0, meaning the task results are totally inaccurate, to 5, meaning the task results are completely accurate.
- Time: a measure of the actual time to complete the task as performed by the doer

The association "precedes" links a task to a set of tasks and shows the set of tasks which must be completed before a task can be performed. This association has no attributes.

These object classes and associations are captured in Figure 18 and are the basis for the design of the Scenario Assignment Sub-System.

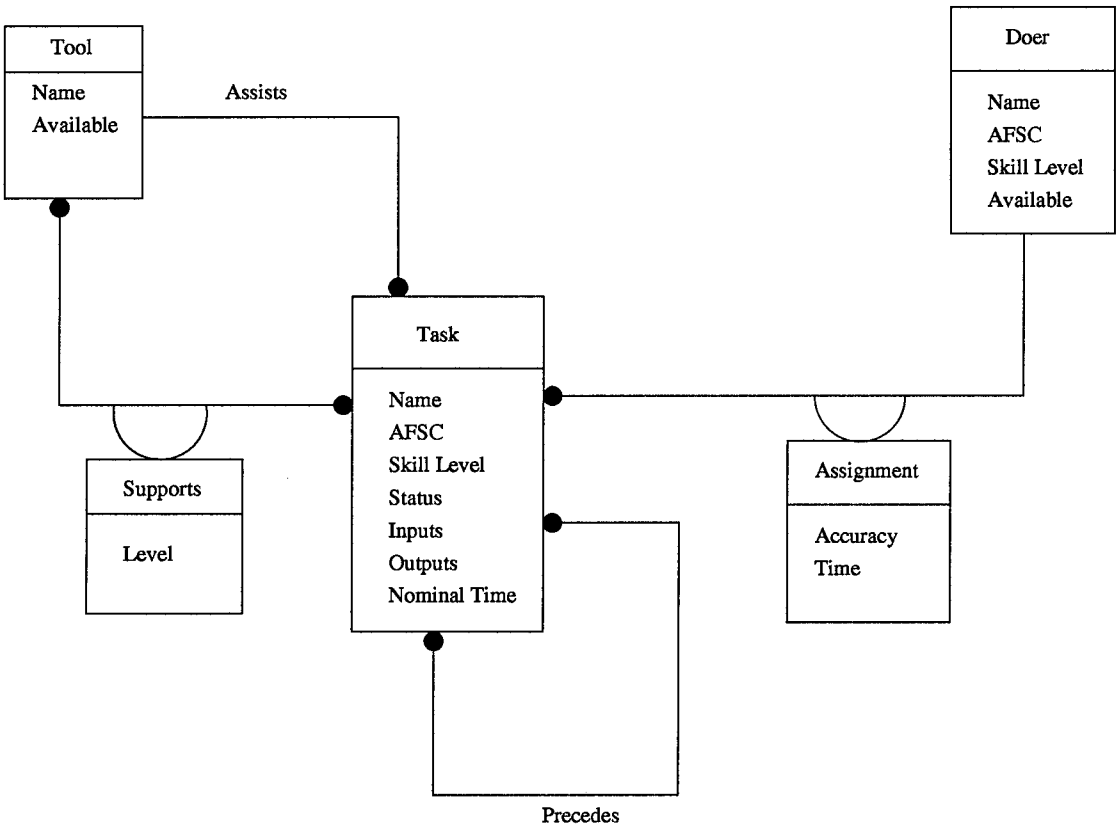


Figure 18. Object Model of Scenario Assignment Sub-System

4.3 Design

Identifying the object classes, attributes, and associations is the start of the design process. Next, the objects and associations must be further refined and methods and packages identified.

4.3.1 Object Refinement. The object classes have been identified in Section 4.2. It is necessary to store the instances of each of these classes as they are created. Thus, container classes are also necessary. The using object will create instances before adding them to the container class. Thus, the following container classes will be defined:

- Task_Container
- Tool_Container
- Doer_Container
- Container class for any associations implemented as associative objects

4.3.2 Associations. In the design phase, a strategy must be chosen for implementing the object model's associations. Since each of the associations in this sub-system is used in different ways, each association was analyzed separately for implementation rather than adopting a global strategy.

The association "assists" is a one-to-many relationship from **tool** to **task**. This association will typically be one-way. One may be given a task and wish to know the tool that has been selected to assist in that task, but rarely, if ever, will one be given a tool and need to know which tasks it assists. Therefore, this association can be implemented as a one-way pointer attribute within the task object class. The association "supports" is a many-to-many relationship between **tool** and **task** with one attribute named **level**. Thus,

"supports" is best implemented as an associative object with pointers to the task and tool. The link attribute can also be stored as an attribute of the task object. Thus, task will have two additional attributes: **tool**, which points to a tool instance, and **level**, which indicates the level of that tool's support for the task. This design decision is illustrated in Figure 19. As an associative object, there will be many "supports" instances. Therefore, "supports" will need a corresponding container class, **Support_Container**.

The association "precedes" is a many-to-many relationship between instances of **task** objects. This association is also typically one-way. One will need to determine the predecessors of a given task but will rarely need to know the task which it precedes. Thus, the association will also be implemented as a one-way pointer to a set of tasks. The object class **task** will have an additional attribute, **predecessor**, which will point to a set of tasks which precede the given task. This design decision is illustrated in Figure 20.

The association "assignment" is a one-to-many relationship from **doer** to **task**. This association will be two-way. One will want to find the doer assigned to a given task, as well as find the tasks assigned to a given doer. Since there are also link attributes, it is best to implement this association as a distinct associative object. Thus, an object class **assignment** will be created with four attributes: **doer**, which points to a doer instance; **task**, which points to a task instance; **time**; and **accuracy**. This design decision is illustrated in Figure 21. As an associative object, **assignment** will need a container class, **Assignment_Container**.

4.3.3 Methods. After refining the objects and associations, it is necessary to identify the methods for each object. Classic Ada PDL was used to design the methods

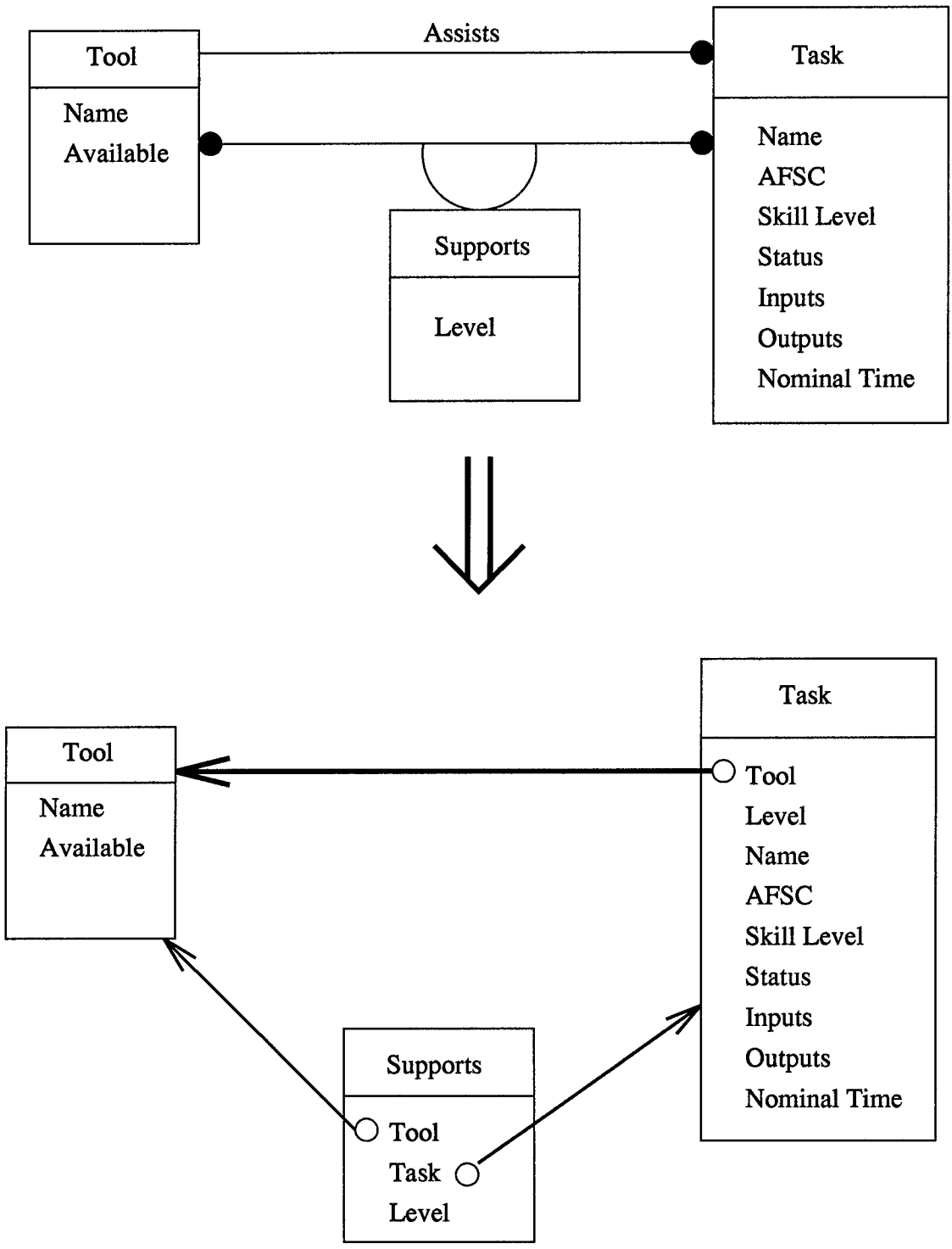


Figure 19. Implementation of Assists and Supports associations using one-way pointers

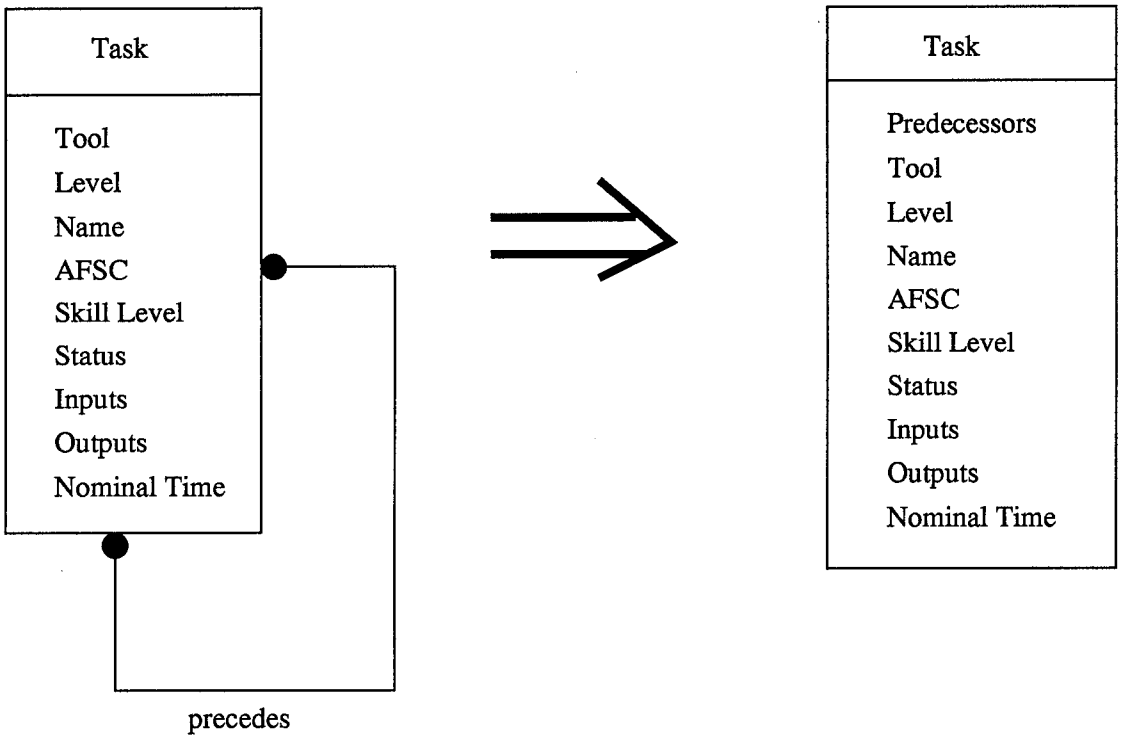


Figure 20. Implementation of Precedes association

for each object. The complete Classic Ada PDL specifications and bodies for each object are listed in Appendix B. For the most part, the methods are simple attribute assignment methods. In each object, there are "Set_Attributes" for each attribute which take a state variable as input and assign to the object's attribute the value in the state variable. There are also "Get_Attributes" which output the value of the object's attribute to an output variable. "Create" methods create an instance of an object, and "Initialize" methods set the initial values of the object's attributes. The methods for each object are listed below. Algorithms are also included for methods which are not straightforward.

- Doer
  - Create
  - Initialize

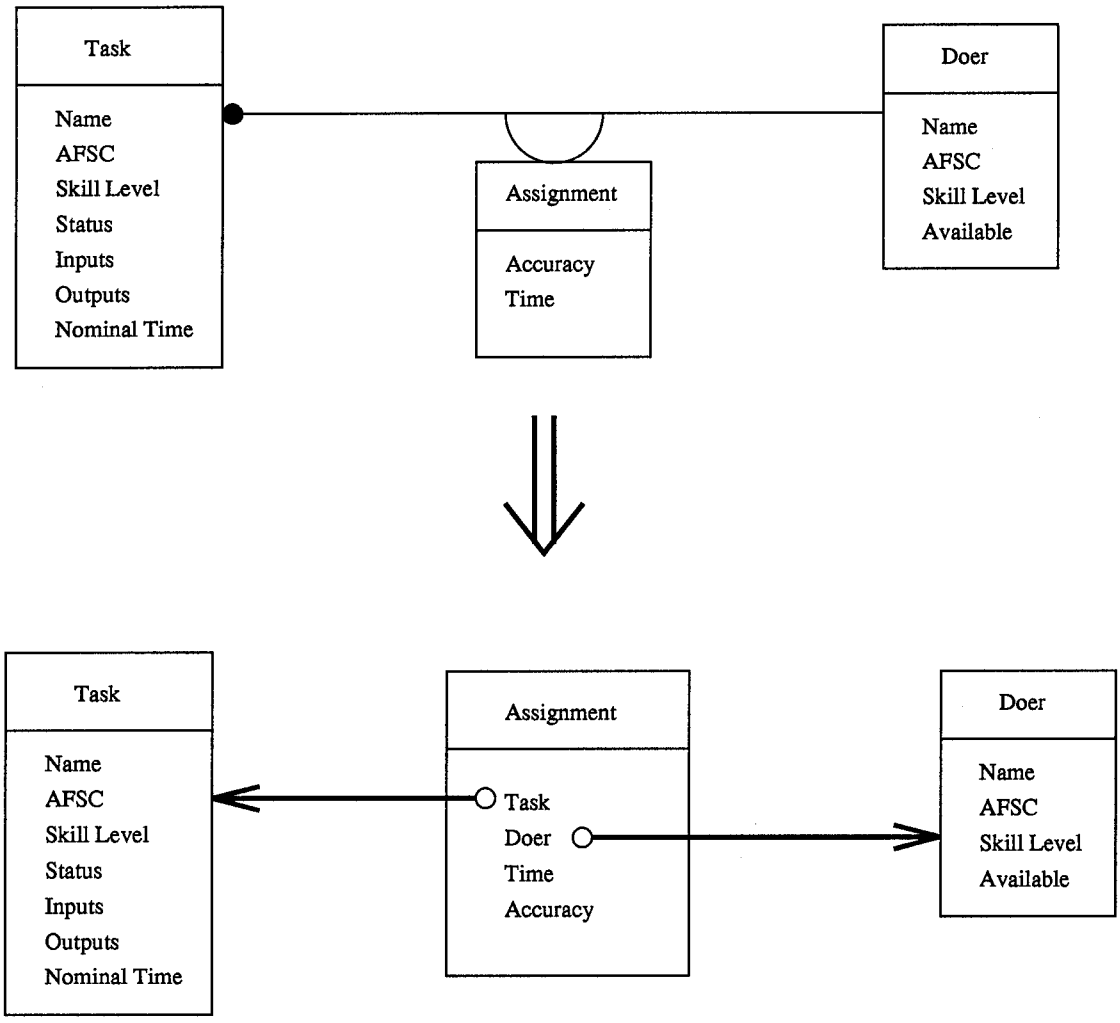


Figure 21. Implementation of Assignment association as associative object

  - Set_Name
  - Get_Name
  - Set_AFSC
  - Get_AFSC
  - Set_Skill_Level
  - Get_Skill_Level
  - Set_Available
  - Get_Available
- Tool
  - Create
  - Initialize
  - Set_Name
  - Get_Name
  - Set_Available
  - Get_Available
- Task
  - Create
  - Initialize
  - Set_Name
  - Get_Name
  - Set_AFSC
  - Get_AFSC
  - Set_Skill_Level
  - Get_Skill_Level
  - Set_Nominal_Time
  - Get_Nominal_Time
  - Set_Status
  - Get_Status
  - Set_Predecessors.

    The method to set the predecessors of a given task requires knowledge about the entire task set. Since it is setting an attribute of a task instance, however, it should be a task class method rather than a task container class method.

```
begin
   for all tasks do
     for all inputs in current task do
       for all tasks other than current task do
         if current task's current input =
                          any output from current other task then
            add other task to current task's predecessor list
         endif
       endloop
     endloop
   endloop
end Set_Predecessors
```

- Get_Predecessors
- Set_Tool
- Get_Tool
- Set_Tool_Level
- Get_Tool_Level
- Set_Inputs
- Get_Inputs
- Set_Outputs
- Get_Outputs

• Assignment

- Create
- Initialize
- Set_Assignment
- Get_Doer
- Get_Task
- Set_Accuracy.

  The accuracy figure is based on the difference between the skill level of the assigned doer and the skill level requested by the task. The skill level of the doer is decreased by 2 if he does not have the proper AFSC to perform the task. These figures were obtained by discussing the measurement of accuracy with domain experts and how unskilled workers can effect the accuracy of a task.

```
begin
   Get Doer_Skill_Level
   Get Doer_AFSC
   Get Task_Skill_Level
   Get Task_AFSC
   Get Tool_Level for supporting tool (if any)
```

```
    Skill_Level := Doer_Skill_Level
    if Doer_AFSC /= Task_AFSC then
       Decrease Skill_Level by 2
    end if
    Skill_Level := Skill_Level + Tool_Level
    Look up Accuracy figure based on derived Skill_Level and Task_Skill_Level
end Set_Accuracy
```

- Get_Accuracy
- Set_Time

  Time is adjusted based on skill levels, AFSCs, and tool support. The formulas to calculate time were derived through data obtained from the domain experts and through trial and error. The domain experts furnished nominal times in which to complete the task as well as information on how unskilled doers may affect the time. Different formulas were tried and the timing results analyzed until the resulting time figures seemed correct.

```
begin
   Get Doer_Skill_Level
   Get Doer_AFSC
   Get Task_Skill_Level
   Get Task_AFSC
   Get Tool_Level for supporting tool (if any)
   Get Task_Nominal_Time
   Time := Task_Nominal_Time
   if Tool_Level > 0 then
      Decrease Time by factor of (Tool_Level * 1.5)
   endif
   if Doer_AFSC /= Task_AFSC then
      Increase Time by factor of 1.5
   else
      Get Skill_Difference between doer and task
      if Doer_Skill > Task_Skill then
         Decrease Time by (Skill_Difference * Time/5)
     elsif Task_Skill > Doer_Skill then
         Increase Time by (Skill_Difference * Time/5)
     endif
   endif
  Assignment.Time := Time
end Set_Time
```

- Get_Time

- Supports
  - Create
  - Initialize

- Set_Support
- Get_Task
- Get_Tool
- Set_Tool_Level
- Get_Tool_Level

In addition to the methods described for each object, there are methods for each object's container class:

- Tool_Container
  - Load_Tools: Loads tool information from an input file into the container object.
  - List_Tools: Outputs to the screen a list of tools which may be chosen to assist in the current task. Only those tools which support the current task are listed. If no tools exist for the current task, a message will state this fact.
- Doer_Container
  - Load_Doers: Loads doer information from an input file into the container object.
  - List_Doers: Outputs to the screen a list of doers which may be assigned to a task. Information listed includes for each doer: name, AFSC, and skill level. This list will group together those doers who have the AFSC desired for the current task assignment.
- Task_Container
  - Load_Tasks: Loads the current scenario of tasks from an input file into the container object.
  - List_Task: Outputs to the screen information about the current task, including task name, AFSC, skill level, and nominal time.
- Assignment_Container
  - List_Assignments: Outputs to the screen information about all the assignments, to include task and doer names. This method is used for debugging and testing purposes only.
- Support_Container
  - Load_Support_Levels: Loads task-tool-support_level information from an input file into the container object.
  - Get_Support_Record: Finds an instance of a "supports" object which matches a given task and tool; returns zero if no match found.

- Get_First_Tool: Given a task, finds first instance of tool that supports that task; returns zero if no tool found; "remembers" its place in the container object.
- Get_Next_Tool: Given a task, finds the "next" instance of a tool that supports that task; returns zero if no tool found; "remembers" its place in the container object.

In addition to the methods described for each object class and container class, there are methods for a user interface to include:

- Get_Doer: Accepts user's selection of a doer via keyboard input. Will prompt user until user enters valid input.
- Get_Tool: Accepts user's selection of a tool via keyboard input. Will allow user to select "no tool." Will prompt user until user enters valid input.

Finally, routines are necessary in the driver program to input accuracy table data from a file, and to start the user interface, allowing the user to make assignments for all tasks in the scenario. After all assignments have been made, the Scenario Analysis Sub-System will take over and use the data from the Assignment object.

4.3.4 Packages. After the objects, attributes, associations, and methods have been defined, one must determine the physical packaging of these units. The Scenario Assignment Sub-System is partitioned into separate packages for each object class and container class, as well as a package for the user interface and a types definition package. The types definition package is separate since the Scenario Analysis Sub-System also shares this information. By designing a common types package up front, development can proceed on both sub-systems. Thus, the Scenario Assignment Sub-System consists of the following packages:

- Doer_Object package
- Task_Object package
- Tool_Object package
- Assignment_Object package
- Support_Object package
- Doer_Container package
- Task_Container package
- Tool_Container package
- Assignment_Container package
- Support_Container package
- User_Interface package
- Data_Types package

4.4 Summary

This chapter has described the analysis and design of the Scenario Assignment Sub-System. Analysis and design have resulted in definitions of the objects, methods, and packages to be used in implementing this sub-system. The next chapter describes how this sub-system was implemented, tested, and validated.

V. *Prototype Implementation, Testing, and Validation*

5.1 Introduction

The previous chapter detailed the analysis and design procedures followed in defining the structure of the Scenario Assignment Sub-System. This chapter describes how this design was implemented in software, tested, validated and integrated with the Scenario Analysis Sub-System.

5.2 Implementation

Ada 83 was chosen as the implementation language for several reasons:

- Ada provided a smooth transition from the Classic Ada Design PDL into code.
- The Meridian Ada compiler was readily available to the developers.
- The developers were familiar with Ada 83.
- Although not technically an object-oriented language, the encapsulation features of Ada were an asset in implementing the design.

Although Ada 9X has added many object-oriented features, the immaturity of the local Ada 9X compiler made Ada 83 the safer choice.

The data types package was defined first since both the Scenario Assignment and Scenario Analysis Sub-Systems would use this package in further development. The types for objects doer, task, tool, and assignment are defined as records, with their respective attributes making up the record components. Arrays are used for the container class types, and pointer types for each object are integers. Thus, a pointer to a doer, task, or tool object is an integer, which is a subscript into the appropriate container array. Dynamic lists with pointer access types were also considered for the implementation of the container

classes, but since there was a relatively small number of instances for each object, and the ordering of the objects in the container was of no consequence, the simplicity of arrays was preferred.

The implementation into packages of the Classic Ada PDL methods discussed in Section 4.3.3 was fairly straightforward. No unusual problems were encountered in coding the Doer_Object, Task_Object, or Tool_Object packages. Since objects are not being created dynamically, there is no need for "create" procedures. Instances are "created" within the initialize procedures where the attributes of an object are set by reading the data in from a file for the doer, task, and tool objects.

For the **task** object, the **predecessors** attribute can not be set at initialization, since its value depends on all the other tasks in the task set. Thus, the procedure to Set_Predecessors is more involved than the other SET procedures. The Set_Predecessors procedure looks for matches between any input of the task at hand and any outputs of any other task in the task set. When matches are found, a pointer to the matched task is added to the task at hand's predecessor list. The tool level attribute of the task object is set when the user selects a tool for that task. Using the selected tool pointer and the current task pointer, the corresponding tool support record is looked up in the **support_container**. The tool level is extracted from this record. As decided during system analysis, these tool level values range from 0, meaning the tool provides no support to the task, to 5, meaning the tool can basically do the complete task accurately and quickly.

The assignment object has a procedure Set_Assignment which assigns pointers to a doer and task to Assignment object's doer and task attributes, respectively, after the user

selects a doer via the user interface. The procedure Set_Accuracy calculates an accuracy measure through a factored skill level based on the doer's AFSC and skill level. A mismatched AFSC or inadequate skill level lowers the factored skill level. The tool level, if any, is then added to the factored skill level. Finally, the accuracy measure is looked up in a table. This accuracy table is implemented as a two-dimensional array. Each row represents the skill level desired by the task, which is an attribute of each task instance, while the column represents the factored skill level, which is calculated as described above. As decided during system analysis, the accuracy table values range from 0, meaning the task results were totally inaccurate, to 5, meaning the task results were completely accurate. An assignment with a factored skill level greater than the task skill level would have a high accuracy rating. An assignment where the factored skill level was less than the task skill level would have a lower accuracy rating. The user could then translate these figures into the actual results of the task assignment, for example, how many bombs on target.

Setting the time attribute of the assignment object follows a similar process. The time is initially set to the task's nominal time and is increased if the doer's AFSC doesn't match the task's AFSC, or if the doer's skill level is less than that desired by the task. The time is decreased by the level of tool support, if any exists, and if the doer's skill level is higher than the desired skill level.

Implementation was also straightforward for the container classes. The doer container class includes a procedure to list possible doers for a current task in the scenario. Procedure List_Doers first displays those persons with the same AFSC as the task. It then displays those persons who do not have the same AFSC. The procedure List_Tools, in the tool container class, only displays those tools that can assist in the current task. This is

accomplished by using the **Support_Container** class operations **Find_First_Tool** which takes a task and returns the first tool which supports it, and **Find_Next_Tool** which takes a task and a current pointer into the container class and returns the "next" tool which supports the task.

User Interface implementation was also routine. The procedures Get_Doer and Get_Tool both have exception handlers to take care of invalid user input. These procedures will continue in a loop until the user enters valid input. When invalid input is entered, a message shows the user the valid range of inputs. A sample screen in the user interface is displayed at Figure 22.

The driver procedure for the Scenario Assignment Sub-System is called Make_Assignments. It includes procedures to load the accuracy information, start the user interface, and make the assignments based on user selections. This driver also contains testing and debugging routines discussed in the next section.

5.3 Testing

In testing the Scenario Assignment Sub-System, the goal was to verify that the requirements were met, and that the sub-system performed correctly. The requirements to be tested are listed in Section 4.2.1. Sample data, including doer, tool, and task information, was created to test the requirements. The first three requirements were to load sets of data: tool, task, and doer information. The goal of this testing was to ensure that procedures loaded this data accurately. Test procedures to print out the contents of the tool, task, and doer containers confirmed that this data was being loaded correctly. The procedure Print_Task also printed the predecessors for each task. This list of predecessors

```
*************************************************************

THE CURRENT TASK IN THE SCENARIO IS...

   NAME            AFSC      SKILL LEVEL     NOMINAL TIME

Obtain Intel       19TXX          2               45

*************************************************************
THE FOLLOWING PERSONS ARE FULLY QUALIFIED TO PERFORM THE CURRENT TASK:

   ID        AFSC       NAME              SKILL LEVEL
 ------      -----    ----------------    -----------

   1         19TXX    Mission Planner          4
   2         19TXX    Mission Planner          2

THE FOLLOWING PERSONS ARE NOT FULLY QUALIFIED TO PERFORM THE
CURRENT TASK, BUT MAY BE SELECTED:

   ID        AFSC       NAME              SKILL LEVEL
 ------      -----    ----------------    -----------

   3         20PXX    F-16 Pilot               2
   4         20PXX    F-16 Pilot               4
   5         20PXX    F-16 Pilot               5


ENTER SELECTION BY ID FOLLOWED BY <CR>:


THE FOLLOWING TOOLS ARE AVAILABLE TO ASSIST WITH THE CURRENT TASK.
LEVEL INDICATES DEGREE OF ASSISTANCE:

  ID        NAME                LEVEL
  --    -------------------     -----

  5      ORACLE                   1



ENTER TOOL SELECTION BY ID FOLLOWED BY <CR> IF YOU DO NOT WISH TO CHOOSE
A TOOL, OR NONE IS AVAILABLE, TYPE 0 (ZERO) FOLLOWED BY <CR>:
```

Figure 22. Sample User Interface Selection Screen

was compared to a hand-generated list and they matched. The "display" requirements were easily tested by running the user interface and comparing what showed up on the screen to expected results. User selection requirements were tested by attempting to enter both valid and invalid input. Exception handling caught the invalid input and prompted the user to make a valid selection. The procedure Print_Assignments was used to test the results of making assignments. In order to check the validity of the accuracy and time attributes, assignments were made with a wide variety of selections. Assignments were made with mismatched AFSCs, matching AFSCs, equal skill levels, high doer skill level with low task skill level, low doer skill level with high task skill level, and a variety of tool levels combined with the previous variations. In all cases, the time and accuracy figures obtained were reasonable.

5.4 Validation

While verification deals with ensuring that the software operates correctly, validation deals with determining whether the software delivers what the user wants. It is difficult to validate the Scenario Assignment Sub-System apart from the Scenario Analysis Sub-System. The Scenario Analysis Sub-System gives the user results of his assignment selections, including the efficiency and timeliness of the resulting assignments. It is easy to validate with the users whether this information is valuable. For a discussion of the validation of the Scenario Analysis Sub-System, the reader is directed to (19). It is more difficult to validate the intermediate results given by the Scenario Assignment Sub-System. Thus, for the validation process, users were asked to run the user interface and make assignment selections. They were then asked how easy the selection process was and whether

improvements could be made. Fellow Air Force graduate students who were familiar with wing C2 were selected to test the system. They reported that the user interface was easy to work with and the selection process was clear. No enhancements were suggested.

5.5 Integration

The Scenario Assignment Sub-System was easily integrated with the Scenario Analysis Sub-System. It was decided to integrate the sub-systems rather than to have two separate sub-systems. An integrated tool would be easier for the user, and would make it unnecessary to pass files from tool to tool. The data types package was already being shared by the two sub-systems, so this was not a problem. Next, the driver programs for both sub-systems were combined to create one main driver. This driver loads the input files for doers, tools, tasks, tool levels, and accuracy figures. It then, through the user interface, prints the scenario tasks in order and allows the user to make doer and tool assignments for each task. After all assignments have been made, the driver calls the analysis procedures and finally prints out the analytical statistics, showing the user the efficiency and time constraints for the selections made. Testing was accomplished by running the integrated sub-system using the data sets created for testing the individual sub-systems. The output was hand checked against expected results. This testing showed the integrated sub-system was performing properly. The complete prototype reasoning tool was placed in a common directory, and the project sponsor was invited to evaluate it. Details on how to run the Resource Allocation tool are contained in Appendix C, Configuration Management.

5.6 Conclusion

This chapter showed how the Scenario Assignment Sub-System was implemented in Ada, tested, validated, and integrated with the Scenario Assignment Sub-System. The domain analysis and tool analysis and design work done up front ensured a smooth implementation. The next chapter shows the overall research results and provides recommendations for future research.

VI. Conclusions and Recommendations

6.1 Introduction

This chapter summarizes the accomplishments of this research. A review of the original problem statement and research goals is followed by a discussion of how these goals were met. It is then shown that the results of this research can be mapped to another problem domain with known solution techniques. Finally, suggestions for future research are offered.

6.2 Research Summary

The primary purpose of this research was to demonstrate the feasibility and benefits of applying software engineering technology to Air Force wing command and control. More specifically, the research goals were to:

1. Demonstrate how domain analysis techniques can be used to develop engineering models of AF wing C2 operations.
2. Show how these models can be used to develop a software tool that determines the effects of applying automation to the wing C2 process.

This research is important because, currently, wing decision makers do not have a way to systematically represent how a wing performs its mission. This has led to the development of software systems which are short-sighted, very limited, or even unusable. The domain models developed in this research have captured knowledge about the key objects, operations, and relationships involved in processing ATOs. These domain models can serve as a management tool for wing decision makers, and a reusable repository for system developers and maintainers.

The goal of demonstrating how domain analysis techniques can be used to develop models was achieved by first gaining an understanding of domain analysis, through review of current software engineering literature. Past and present domain analysis work in the C2 arena was also examined. Based on this research, a specific domain analysis process was defined consisting of the following steps:

1. Define the domain.
2. Scope the domain.
3. Identify sources of domain knowledge.
4. Obtain domain knowledge.
5. Choose model representation.
6. Develop the domain model.

Using this process, domain analysis was performed on wing C2 operations. The output of this process was object and functional models of how a wing accomplishes the mission of processing an ATO. The correctness of these models was verified by domain experts. This research successfully showed how a domain analysis process can be used to develop models of wing C2 operations.

The second goal was then achieved, showing how these models can be used to develop a software tool for determining the effects of applying automation to the wing C2 process. The domain model was used as a knowledge base to design and build an object-oriented resource allocation tool which allows wing personnel to make intelligent resource assignments for achieving ATO tasks. This system gives the wing decision makers an executable model of wing C2 operations and shows them where automation can provide the best value.

This is just one example of how the domain model can be used to design reusable, effective tools which help the wing decision maker increase unit effectiveness and readiness.

6.3 Comparison with Scheduling Theory

The approach followed in this research was to apply software engineering domain analysis techniques to wing C2 operations. However, the implementation of the prototype software tool developed from this research has also shown how sequencing and scheduling theory can be applied to wing C2. In particular, the resource allocation tool is an application of the classic job-shop scheduling problem. Job-shop scheduling problems are concerned with processing jobs through a set of machines or processors (3). In the case of wing C2, the jobs are the tasks necessary to process an ATO, and the machines or processors are the various wing personnel concerned with completing an ATO.

Like job-shop scheduling, ATO processing is concerned with performance measurement and many of the performance measures for job-shop scheduling apply directly to the processing of an ATO. French lists the following set of metrics as general performance measures (3), several of which were implemented in the prototype software tool:

- due date
- waiting time
- total waiting time
- completion time
- flow time
- lateness
- earliness

The bottom line is that processing an ATO is really concerned with the effective and efficient assignment of resources, which is the same as one of the objectives of job-shop scheduling. Thus, the problem of assigning resources to process an ATO is best categorized as a job-shop scheduling type of problem, and properly classifying a problem is the most important, and often the hardest, part of solving a particular problem. While the current prototype tool primarily performs an analysis of a specified schedule, scheduling theory also addresses algorithms for creating an optimal schedule.

6.4 Recommendations for Future Research

This research provides an important first step in capturing the domain knowledge needed to assist in future wing C2 automation efforts. There are several areas, however, that merit further investigation:

1. *Continue validation of domain model.* The object and functional models were validated by PACAF domain experts. These models should also be validated by different AF wings to determine whether they really have captured the essence of the ATO process. This is a necessary step in determining the true reusability and accuracy of these models.
2. *Continue validation of prototype reasoning system.* Validation of the resource allocation tool is incomplete. The lack of wing C2 domain experts in the local area, and the geographical distance to the sponsor, limited the amount and quality of system validation. Before proceeding with further research or enhancements of the system, full validation of the system should be accomplished to determine the value of this tool to wing personnel. Domain experts should be given the opportunity to use the tool and provide feedback on its usefulness.
3. *Add intelligent features to the Scenario Assignment Sub-System.* Currently, the Scenario Assignment Sub-System allows the user to assign personnel and automated tools to each task in a scenario but doesn't give the user much information to assist in picking the most efficient persons and tools for the job. The sub-system could be enhanced to show information, such as the relative workload of personnel or the concurrency of tasks, which would help the user make the optimal assignments. Ultimately, the sub-system could use artificial intelligent techniques to automatically

make the optimal assignments based on AFSCs, skill levels, availability of personnel and tools, and timing requirements.

4. *Add database features to the Scenario Assignment Sub-System.* Currently, task, tool, and personnel data are loaded from files to be used in the sub-system. The sub-system could be enhanced by allowing the user to add or update this information. Alternatively, the subsystem could tie into a DBMS so this data could be managed more effectively.
5. *Validate/Add metrics.* The time and accuracy metrics defined in the Scenario Assignment Sub-System were not thoroughly validated by domain experts. This should be done. In addition, the accuracy metric should be further refined to give a more precise indication of the accuracy of a task. It should be investigated as to whether the current accuracy measure of 1 to 5 can be translated into specific measurements of accuracy, for example, number of bombs on target. The domain experts may also offer ideas on additional metrics which may be useful in analyzing the effects of tool and personnel assignments on mission effectiveness and readiness.
6. *Add the ability to assign multiple persons to a single task.* The Scenario Assignment Sub-System is limited to assigning a single person to a task. In reality, several persons may actually perform a task in the ATO processing scenario. Further research should look for a feasible way to model the assignment of several persons to a task.
7. *Add the ability to assign multiple tools to a single task.* The Scenario Assignment Sub-System is limited to assigning a single tool to a task. In some cases, more than one tool may actually be desired for a task. Further research should investigate whether this is a valid requirement and determine a way to achieve it.
8. *Investigate the application of job-shop scheduling techniques.* Since the wing C2 problem maps directly to a scheduling problem, many of the potential improvements to the Resource Allocation tool relate directly to the field of job-shop scheduling. Further research into the application of scheduling theory should be pursued.

6.5 Summary

This research successfully developed a domain model of AF wing C2 operations and showed how this model could be used to develop a tool which assists the wing decision maker in applying automation to the C2 process. While more work and further research should be accomplished, this research has shown that software engineering technology can be effectively applied to AF wing C2 and can provide numerous benefits. Furthermore, the prototype tool has been shown to be an application of job-shop scheduling theory,

which opens up a whole new approach to solving wing C2 problems. These results and the domain model developed through this research can continue to be used in increasing the mission effectiveness and readiness of AF wing C2.

Appendix A. Data Dictionary

This is the Data Dictionary for the Air Force wing C2 domain model from Chapter III. It is a subset of the Data Dictionaries found in (10) and (11).

1. **Accuracy** is an attribute of the Assigned association and is a numeric ranking between 1 and 5 (1 is low and 5 is high) that indicates the accuracy (or quality) of the products from the particular task.
2. **Admin** is a general reference to a class of people who perform administrative tasks in support of mission execution.
3. **Aircraft** is the plane found in the wing, and for this research is an F-15 or F-16.
4. **Aircrew** is a pilot, and depending upon the aircraft, a pilot and weapons officer.
5. **Air Force Specialty Code (AFSC)** is an alphanumeric job specialty code assigned to Air Force personnel.
6. **AFSC Required** is an attribute of an ATO Task and is the job specialty best suited to perform the task.
7. **Air Operations Center (AOC)** is the theater level air operations management facility. The AOC monitors all air activity within the theater and is responsible for generation and dissemination of the daily ATO.
8. **Airspace Information** is a reference to a broad category of information dealing with the air combat environment.
9. **Air Tasking Order (ATO)** directs all planned airspace activity for the day. Missions are identified by wing and include mission number, name, and type, number of aircraft, time on target, and flying altitude among other things. The ATO is issued daily by the AOC.
10. **Analyze Threat to Mission** is an activity that involves comparing and analyzing correlated OB and threat system parameters with mission information to determine which threats will affect the mission. The threat assessment is used to plan the flight path and tactics. In addition, Intel produces a picture of the area threat environment that is primarily used for briefing purposes.
11. **Assigned** is the association (or relationship) between C2 Task and Wing/Wing Operations Center.
12. **Assess Air Support Assigned** is an activity where planners must identify the other wings that are to provide support to the mission and coordinate with them to ensure a smooth operation. If insufficient support is given to the wing, planners may call the AOC to request more support.

13. **ATO FRAG** is the portion of the ATO that specifically corresponds to the wing and the special instructions that accompany it. The ATO FRAG is needed before detailed mission planning can begin.
14. **ATO Task** is a category of C2 Task that is required to plan and execute an ATO.
15. **Battle Staff** is the senior staff function within the WOC.
16. **C2 Task** is a general category of tasks that refers to tasks that are necessary to perform wing operations.
17. **Conduct Pre-Mission Activities** is a sub-task of Conduct Wing C2 Operations and refers to conducting the pre-mission activities (i.e., preparing the aircraft, preparing the aircrews, etc.).
18. **Conduct Post-Mission Activities** is a sub-task of Conduct Wing C2 Operations and refers to conducting the post-mission activities (i.e., regenerating the aircraft, debriefing the aircrews, etc.).
19. **Conduct Wing C2 Operations** is process that encompasses all activity associated with command and control of fighter wing operations in a sustained deployed wartime environment.
20. **Coordinate With EW** is an activity where electronic warfare (EW) coordinates with mission planners to ensure that enemy threat systems have been appropriately considered during planning of the mission. In addition, EW provides guidance to personnel responsibility for carrying out the action. Examples of guidance include tactics, chaff and flare use, and loading/reprogramming directives for the electronic countermeasures.
21. **Coordinate With Support Wings** is an activity where planners must coordinate rendezvous point and lead times with other wings involved in the mission to ensure efficient mission execution.
22. **Create Basic Setup/Softcopy Schedule** is an activity where the schedulers build the softcopy schedule. The softcopy schedule must be approved by the wing commander, operations group commander, or a representative. At any time the softcopy schedule can be updated to reflect changes related to the mission.
23. **Correlated Order of Battle** is a correlation of multi-source threat data into a single comprehensive Order of Battle (OB).
24. **Determine Mission Specifics** is an activity where more detailed planning must be done, after general information has been extracted from the ATO. Further mission details include weather an threat information, communications and navigation requirements, tactics, fusing, and support wing coordination. Many experts on these topics are assigned to the MPC to help plan the missions.
25. **Determine Weapons Package** is an activity where the weapons officer determines which bombs to put on the targets and the type of fuse to be used. Target size/type, munitions availability, and changing weather and intelligence have an impact on which weapons are selected.

26. **Data Transfer Cartridge (DTC)** is a plug-in module that is used to program the flight computer with flight path information.
27. **EW Guidance** is guidance that is given on electronic warfare issues used in preparations of mission planning packages, aircrews, and aircraft.
28. **Feedback** refers to periodic updates of how the the mission is going.
29. **Finalize Schedule and Plans** is an activity where the softcopy schedule is modified (if necessary) and finalized after the mission specifics have been determined. The schedule will be maintained at the squadron. Later, squadron maintenance personnel will assign aircraft to the missions. The wing commander or his representative often check the schedule before it becomes final.
30. **Guidance/Direction** refers to the guidance and direction that Regulations provide to the Conduct Wing C2 Operations task.
31. **Input** is an attribute of an ATO Task and refers to the pieces of information required to begin the task.
32. **Intelligence** is a general reference to a class of people who perform intelligence-related tasks.
33. **Intel/Targeteer** is a person within the MPC who analyzes intel information and provides assessments to the mission planners in the MPC.
34. **In-Transit Activities** are activities associated with managing an on-going mission. Mission monitoring, reporting base and threat status, briefing higher authorities, and setting up alternate WOC facilities, and processing transient aircraft (if required) are examples. Many of these activities are the responsibility of the Mission Director, maintenance, disaster preparedness, and the battle staff who serve as facilitators.
35. **Launched Mission** is the actual departure of mission aircraft.
36. **Life Support** is a general reference to a class of people that provide life support to the aircrews.
37. **Logistics Group** is a group within the wing that is responsible for the logistics of sustaining a wing: equipment maintenance, supply, transportation, and logistics.
38. **Maintenance** is a general reference to a class of people within the wing that perform maintenance tasks on aircraft or equipment.
39. **Manage Mission and Base Support** is a sub-task of Conduct Wing C2 Operations and refers to managing the on-going mission and associated base support functions.
40. **Medical Group** is a group within the wing that provides medical services for wing personnel. In general, the Medical Group plays little role in planning and executing the ATO.
41. **Mission Commander** is the pilot in charge of a particular mission or strike package.
42. **Mission Director** is the senior person normally in charge of the MPC and a senior member of the WOC staff.

43. **Mission Packages** are folders containing all the information a pilot needs to fly the mission including: line-up card, maps with threat rings, search and rescue information, communications frequencies, special instructions, weapons and airplane checklists, code words, altitudes, ground spots, target imagery, kill boxes, and an Army information book for close air support missions.

44. **Mission Planner** is a person who performs mission planning tasks.

45. **Mission Planning Cell (MPC)** is the cell in the WOC responsible for all planning and scheduling activities.

46. **Mission Related Weather** is current weather conditions that affect the mission.

47. **Mission Scheduling and Planning** is an activity for preparing the schedules and plans for the next day's air operations. Planners match the taskings with available resources to meet mission requirements. This is accomplished by personnel in the MPC which can operate up to twenty-four hours per day. In the MPC, representatives from the squadrons including scheduling, intelligence, maintenance, munitions, electronic combat, weather, and command and control work together to determine the safest and most efficient route to accomplish the mission. The end result of this activity is the mission package and associated schedules for all missions to be flown.

48. **Mission Specific Intel** is information including SAM sites, enemy force location, imagery, minimum risk routing, search and rescue info, electronic order of battle, outside threats, threat airfields, threat CAPs, target type, makeup, location, and protection, and ingress/egress procedures that directly impact the mission.

49. **Mission Specifics** is a broad scope term including a variety of information not contained in the ATO (i.e., intel, weather, tactics, etc.) but necessary to perform the mission.

50. **Nominal Time** is an attribute of an ATO Task and represents the time it would take to complete the task for a person with an AFSC and skill level matching AFSC Required and Skill Level Required.

51. **Non-ATO Task** is a category of C2 Task that does not pertain to execution of the ATO.

52. **Obtain Intel** is an activity where mission planners use intel to provide them with the latest information concerning location and capability of enemy forces to plan the safest routes for mission completion. Intelligence includes SAM sites, enemy force locations, enemy ground and air order of battle, minimum risk routing, imagery, electronic order of battle, outside threats, threat airfields, threat CAPS, and egress procedures.

53. **Obtain Weather** is an activity where the weather representative assigned to the MPC coordinates with the Weather Shop and gets the weather information needed to plan and schedule the mission.

54. **Order of Battle (OB)** is intelligence information that reflects the battlefield situation. Information is collected from several sources including debriefings, intelligence systems, and force-level intelligence.

55. **Operations Group** is the Group within the wing that is responsible for all aircraft operations.
56. **Operations Squadron** is the operational part of the Operations Group. There are generally two Operations Squadrons per Operations Group.
57. **Output** is an attribute on an ATO Task and refers to the products of that task.
58. **Perform C2 Task** is an operation for the Wing and Wing Operations Center.
59. **Plan Route from Initial Point to Target** is the activity where the specific details of the route must be planned for. Mission planners determine way points, fuel requirements, take off times, location of allied and enemy ground forces, and target approach. The initial point (IP) is the last point on the route before the approach to the target. The IP to target run specifies the load delivery tactic and sequence, altitude, ingress and egress paths, countermeasure employment, and target and IP imagery if possible. In addition, planners consider tactics options so more than one tactic can be employed for multiple target passes.
60. **Plot Route and Threats** is an activity where planners are responsible for plotting the routes and threats on maps for aircrews to follow. Planners use the latest weather, intelligence and terrain information, threat information, rules of engagement, and guidance from the AOC to plot the routes.
61. **Post-Mission Activities** are those activities occurring from the time the aircraft return from the mission to the time the aircraft are re-generated and readied for another mission.
62. **Pre-Mission Activities** are those activities that are required to ensure the mission is properly prepared. This refers primarily to making sure the aircrews are rested, briefed, and outfitted. Also included is making sure the aircraft is properly fueled, maintained, and loaded to execute the mission.
63. **Prepare Mission Plans and Schedules** is a sub-task of Conduct Wing C2 Operations and refers to preparing the mission plan and associated schedules.
64. **Prepare Mission Packages** is an activity where mission planners produce the mission packages; this is the ultimate goal of mission planning.
65. **Recovered Mission** is the actual arrival and recovery of mission aircraft.
66. **Regulations** are documents that govern wing operations.
67. **Reports** is an output from the Conduct Wing C2 Operations task that gives the AOC an impression of how the mission went.
68. **Resources** are what the wing provides in support of performing the Conduct Wing C2 Operations (i.e., personnel, facilities, equipment, etc.).
69. **Route** is the planned path for the particular mission.
70. **Route Map** is the map showing the route to the target that mission planners and intel have determined to be the safest and most efficient.
71. **Scheduler** is a person who performs scheduling tasks.

72. **Schedules** are mission taskings that have been converted into the local sequence of events for aircraft departure/arrival. The schedule typically contains information about the weapons load, fuel, takeoff times, vulnerability times, return time, and pilots. The schedule is distributed to all base agencies concerned with the takeoff and arrival of aircraft.

73. **Skill Level** is a numeric ranking between 1 (low) and 5 (high) indicating how proficient the operator is at his or her job specialty.

74. **Skill Level Required** is an attribute of an ATO Task and is a numeric ranking between 1 and 5 indicating the skill level best suited to perform the task. This relates closely to the AFSC Required.

75. **Softcopy Schedule** is an initial copy of the schedule depicting number of sorties, mission flow, and turnaround times. Detailed mission planning must be performed before the schedule can be approved by the wing commander and finalized.

76. **Squadron Commander** is the first level authority of squadron operations.

77. **SRC Cell** is a cell within the WOC that is responsible for all aspects of the mission other than planning, scheduling, and executing the ATO.

78. **Support Assessment** is an assessment of the support required to successfully complete a mission. MPC personnel will contact other wings involved to ensure a safe and successful mission.

79. **Support Coordination** is an agreement between wings involved in the mission concerning the support required to complete the mission.

80. **Support Group** is a group within the wing that provides support to the Operations Group and the rest of the wing: security police, civil engineering, communications, and personnel.

81. **Support Squadron** is a squadron within the Operations Group that provides operations-oriented support to the Operations Squadrons: weather, weapons, and intelligence.

82. **Tactics** are techniques to be used to fly the mission and drop munitions in the most effective manner. Included are flight patterns, evasion techniques, countermeasure employment, weapons delivery, fusing, etc.

83. **Tasking** is what the AOC gives to the Conduct Wing C2 Operations task.

84. **Threat Assessment** is correlated threat data collected from multiple sources that serves as an assessment of the threat as it relates to the missions.

85. **Threat System Parameters** are parametric information on threat system performance and jammer capabilities. This information changes as systems are modified or new characteristics are observed such as war reserve modes.

86. **Time** is an attribute of the Assigned association and is the time required to complete the task. The amount of time required to complete a task varies depending upon the operator's AFSC, skill level, and use of an automated tool.

87. **Tool** is a generic reference to an automation tool that may be used by wing personnel to increase the accuracy and efficiency of a task.

88. **Weapons** is a general reference to a class of people who perform weapons related tasks.

89. **Weapons Officer** is a person who resides within the MPC and makes decisions about weapons use for given missions.

90. **Weapons Package** is a weapons configuration and fusing information that the weapons officer in the MPC has selected for the mission.

91. **Weather Data** is alphanumeric and graphical weather information.

92. **Weather** is a general reference to a class of people who perform weather related tasks.

93. **Weather Officer** is a person who resides within the MPC and is principally responsible for providing the weather support needed by the other people within the MPC.

94. **Wing** is the main fighting component of the Air Force. Wings are responsible for performing assigned tasking as spelled out in the ATO. To accomplish this, the wing performs a variety of tasks that are generally referred to as command and control tasks.

95. **Wing Operations Center** is the functional organization within the wing and is how the wing organizes itself to plan and execute a mission. The WOC serves as the hub for all wing aircraft operations (i.e., command post). The WOC provides several reports to the AOC to include aircraft, aircrew, and airfield status, mission scheduling, maintenance status, and intelligence reports.

Appendix B. Classic Ada Design Listings

This appendix contains the Classic Ada design specifications and bodies for the objects defined in the Scenario Analysis Sub-System.

B.1 Classic Ada Specifications

```
--Object Name: Doer

CLASS Doer is
--File Name: doer.ca
--Description: Object Doer is a person who can perform a task.
--Specified by:  Robert Hunt
--Specification Date:  20 Sep 94
--Modification Dates:

--Language: Classic Ada PDL
--Compiler Dependencies:  Does not compile using Meridian Ada

METHOD Create (New_Doer: out OBJECT_ID);

INSTANCE METHOD Initialize;

INSTANCE METHOD Set_Variable (State_Variable: in Variable_Type);
INSTANCE METHOD Get_Variable (State_Variable: out Variable_Type);
--Name
--AFSC
--Skill Level
--Available

end Doer;

*****************************************************************************

--Object Name: Task
--File Name: task.ca
--Description: Object Task is a task to be accomplished.
--Specified by:  Robert Hunt
--Specification Date:  20 Sep 94
--Modification Dates:
```

```
--Language: Classic Ada PDL
--Compiler Dependencies:  Does not compile using Meridian Ada

CLASS Task is

METHOD Create (New_Task: out OBJECT_ID);

INSTANCE METHOD Initialize;

INSTANCE METHOD Set_Variable (State_Variable: in Variable_Type);
INSTANCE METHOD Get_Variable (State_Variable: out Variable_Type);
--Name
--AFSC
--Skill Level
--Inputs
--Outputs
--Status
--Nominal Time
--Tool
--Level
--Predecessors

end Task;

************************************************************************

--Object Name: Tool
--File Name: tool.ca
--Description:  Object Tool is a tool which may assist the doer in
--performing a task
--Specified by:  Robert Hunt
--Specification Date:  20 Sep 94
--Modification Dates:

--Language: Classic Ada PDL
--Compiler Dependencies:  Does not compile using Meridian Ada


CLASS Tool is

METHOD Create (New_Tool: out OBJECT_ID);

INSTANCE METHOD Initialize;
```

```
INSTANCE METHOD Set_Variable (State_Variable: in Variable_Type);
INSTANCE METHOD Get_Variable (State_Variable: out Variable_Type);
--Name
--Available

end Tool;

*************************************************************************

--Object Name: Assignment
--File Name: assignment.ca
--Description: Object Assignment is an association between a Doer and a Task
--that shows who is assigned to what task.  Each instance represents one
--such Doer-Task pair.
--Specified by:  Robert Hunt
--Specification Date:  20 Sep 94
--Modification Dates:

--Language: Classic Ada PDL
--Compiler Dependencies:  Does not compile using Meridian Ada

CLASS Assignment is

METHOD Create (New_Assignment: out OBJECT_ID);

INSTANCE METHOD Initialize;

INSTANCE METHOD Set_Assignment (Doer, Task: in OBJECT_ID);

INSTANCE METHOD Get_Task (Task: out OBJECT_ID);

INSTANCE METHOD Get_Doer (Doer: out OBJECT_ID);

INSTANCE METHOD Set_Variable (State_Variable: in Variable_Type);
INSTANCE METHOD Get_Variable (State_Variable: out Variable_Type);
--Accuracy
--Time

end Assignment;

*************************************************************************
```

```
*************************************************************************
--Object Name: Doer

CLASS body Doer is

Name: Name_Type;
AFSC: AFSC_Type;
Skill_Level: Skill_Level_Type;
Available:  boolean;

METHOD Create (New_Doer: out OBJECT_ID is

begin
   New_Doer := INSTANTIATE;
   SEND (New_Doer, Initialize);
end Create;

INSTANCE METHOD Initialize is

begin
   Read (Name);
   Read (AFSC);
   Read (Skill_Level);
   Available := true;
end Initialize;

-- NOTE:  Unless otherwise defined below, all Set_Variable and
-- Get_Variable methods are simple assignment statements and are
-- not further elaborated.

INSTANCE METHOD Set_Variable (State_Variable: in Variable_Type) is

begin
   null;
end Set_Variable;

INSTANCE METHOD Get_Variable (State_Variable: out Variable_Type) is

begin
   null;
end Get_Variable;
```

```
-- Name
-- AFSC
-- Skill_Level
-- Available

end Doer;

*****************************************************************************

--Object Name: Task

CLASS body Task is

Name: Name_Type;
AFSC: AFSC_Type;
Skill_Level: Skill_Level_Type;
Inputs: Input_Type;
Outputs: Output_Type;
Status: Status_Type;
Predecessors: Predecessor_Type;
Nominal_Time: Time_Type;
Tool: INSTANCE OBJECT_ID;
Tool_Level: Level_Type;

METHOD Create (New_Task: out OBJECT_ID is

begin
   New_Task := INSTANTIATE;
   SEND (New_Task, Initialize);
end Create;

INSTANCE METHOD Initialize is

begin
   Read (Name);
   Read (AFSC);
   Read (Skill_Level);
   Read (Nominal_Time);
   Status := NotStarted;
   while not out of inputs loop
      Read (Inputs);
   end loop;
   while not out of outputs loop
      Read (Outputs);
```

```
   end loop;
   Tool := null;        --No tool at initialization
   Tool_Level := 0;
end Initialize;

-- NOTE:  Unless otherwise defined below, all Set_Variable and
-- Get_Variable methods are simple assignment statements and are
-- not further elaborated.

INSTANCE METHOD Set_Variable (State_Variable: in Variable_Type) is

begin
   null;
end Set_Variable;

INSTANCE METHOD Get_Variable (State_Variable: out Variable_Type) is

begin
   null;
end Get_Variable;

--Name
--AFSC
--Skill Level
--Inputs
--Outputs
--Status
--Nominal Time
--Tool
--Level

INSTANCE METHOD Set_Predecessors (Current_Task: in out OBJECT_ID;
                                  Task_Set: in Container_of_Tasks) is

-- Compare current tasks inputs to outputs of all other tasks.  If
-- another task's output matches a current task input, then add the
-- other task to the current task's predecessor list

begin
   for all inputs in Current_Task loop
     for all Comparing_Task in Task_Set loop
       for all outputs in Comparing_Task from Task_Set loop
        if Current_Task.Current_Input := Comparing_Task.Current_Output then
           Add_To_Predecessor_List (Comparing_Task);
        end if;
```

```
        end loop;
      end loop;
   end loop;
end Set_Predecessors;

end Task;

*****************************************************************************

--Object Name: Tool

CLASS Tool is

Name: Name_Type;
Available:  boolean;

METHOD Create (New_Tool: out OBJECT_ID is

begin
   New_Tool := INSTANTIATE;
   SEND (New_Tool, Initialize);
end Create;

INSTANCE METHOD Initialize is

begin
   Read (Name);
   Available := true;
end Initialize;

-- NOTE:  Unless otherwise defined below, all Set_Variable and
-- Get_Variable methods are simple assignment statements and are
-- not further elaborated.

INSTANCE METHOD Set_Variable (State_Variable: in Variable_Type) is

begin
   null;
end Set_Variable;

INSTANCE METHOD Get_Variable (State_Variable: out Variable_Type) is

begin
   null;
end Get_Variable;
```

```
-- Name
-- Available

end Tool;

*************************************************************************

--Object Name: Assignment

CLASS Assignment is

Assigned_Task: INSTANCE OBJECT_ID;
Assigned_Doer: INSTANCE OBJECT_ID;
Assignment_Time: INSTANCE Time_Type;
Assignment_Accuracy: INSTANCE Metrics_Type;
Task_Skill_Level: INSTANCE Skill_Level_Type;
Task_AFSC: INSTANCE AFSC_Type;
Task_Tool_Level: INSTANCE Level_Type;
Doer_Skill_Level: INSTANCE Skill_Level_Type;
Doer_AFSC: INSTANCE AFSC_Type;


METHOD Create (New_Assignment: out OBJECT_ID) is

begin
   New_Assignment := INSTANTIATE;
   SEND (New_Assignment, Initialize);
end Create;


INSTANCE METHOD Initialize is

begin
  Set_Assignment (null, null);
end Initialize;


INSTANCE METHOD Set_Assignment (Doer, Task: in OBJECT_ID) is

begin
   Assigned_Doer := Doer;
   Assigned_Task := Task;
end Set_Assignment;
```

```
INSTANCE METHOD Get_Task (Task: out OBJECT_ID) is

begin
   Task := Assigned_Task;
end Get_Task;


INSTANCE METHOD Get_Doer (Doer: out OBJECT_ID) is

begin
   Doer := Assigned_Doer;
end Get_Doer;


INSTANCE METHOD Set_Accuracy (Accuracy_Data: in Accuracy_Table) is

Factored_Skill: Skill_Level_Type;

begin
   SEND (Assigned_Task, Get_Skill_Level, Skill_Level => Task_Skill_Level);
   SEND (Assigned_Task, Get_AFSC,        AFSC        => Task_AFSC);
   SEND (Assigned_Task, Get_Tool_Level,  Tool_Level  => Task_Tool_Level);
   SEND (Assigned_Doer, Get_Skill_Level, Skill_Level => Doer_Skill_Level);
   SEND (Assigned_Doer, Get_AFSC,        AFSC        => Doer_AFSC);

   if Doer_AFSC /= Task_AFSC then    --mismatched AFSCs: decrease skill level
      Factored_Skill := Doer_Skill_Level - 2;
   else                              --matched AFSCS: maintain skill level
      Factored_Skill := Doer_Skill_Level;
   end if;
   Factored_Skill := Factored_Skill + Task_Tool_Level; --add tool support factor
   Accuracy := Get from Accuracy_Data based on
                     Factored_Skill and Task_Skill_Level;
end Set_Accuracy;


INSTANCE METHOD Get_Accuracy (Accuracy: out Metrics_Type) is

begin
   Accuracy := Assigned_Accuracy;
end Get_Accuracy;


INSTANCE METHOD Set_Time is
```

```
Factored_Time: Time_Type;

begin
   SEND (Assigned_Task, Get_Skill_Level, Skill_Level => Task_Skill_Level);
   SEND (Assigned_Task, Get_AFSC,        AFSC        => Task_AFSC);
   SEND (Assigned_Task, Get_Tool_Level,  Tool_Level  => Task_Tool_Level);
   SEND (Assigned_Doer, Get_Skill_Level, Skill_Level => Doer_Skill_Level);
   SEND (Assigned_Doer, Get_AFSC,        AFSC        => Doer_AFSC);
   SEND (Assigned_Task, Get_Nominal_Time,Nominal_Time=> Factored_Time);

   if Task_Tool_Level > 0 then      --decrease time due to tool support
      Factored_Time := Factored_Time / (Task_Tool_Level * 1.5);
   end if;
   if Doer_AFSC /= Task_AFSC then  --mismatched AFSCs: increase time
      Factored_Time := Factored_Time * 1.5;
   else
      Skill_Difference := Doer_Skill_Level - Task_Skill_Level;
      if Skill_Difference < 0 then
         --doer not skilled enough: increase Factored_Time;
      elsif Skill_Difference > 0 then
         --doer "over-skilled": decrease Factored_Time;
      end if;
   end if;
   Time := Factored_Time;
end Set_Time;


INSTANCE METHOD Get_Time (Time: out Time_Type) is

begin
   Time := Assigned_Time;
end Get_Accuracy;


end Assignment;
```

Appendix C. Configuration Management

This appendix describes the configuration management and file conventions for the Resource Allocation Tool.

C.1 Source Code and Executable Code

The source code files and executable files for the Resource Allocation Tool can be found on AFIT's Hawkeye system under the directory **hartrum/wing/tool**. The program has been compiled using the Meridian compiler's **amake** utility. Thus, if changes are made to any of the program specifications or bodies, simply type **amake**. All necessary recompiles and linking will be carried out automatically. There is no need to compile via the **ada** command nor to link via the **bamp** command. In order to execute the program, simply type **ato_system**.

The Resource Allocation Tool is composed of the following files:

Package Specification files:

- analyze-utils.ads
- analyze.ads
- assigncon.ads
- assto.ads
- doer.ads
- doercon.ads
- support.ads
- supportcon.ads
- task.ads
- taskcon.ads

- tool.ads
- toolcon.ads
- ui.ads

Package Body files:

- analyze-utils.adb
- analyze.adb
- assigncon.adb
- assto.adb
- data.adb
- doer.adb
- doercon.adb
- support.adb
- supportcon.adb
- task.adb
- taskcon.adb
- tool.adb
- toolcon.adb
- ui.adb

Driver procedure file:

- ato.ada

Testing and Debugging package files:

- test.adb
- testass.adb
- makeass.adb

Data files:

- accuracy.dat
- doers.dat
- tools.dat
- tlevel.dat
- tasks.dat

C.2 Data File Conventions

Five data files are needed in order to run the Resource Allocation Tool. These data files are described as follows:

1. accuracy.dat

   This file is a table which holds the accuracy figures and is composed of integers ranging from one to five. The file consists of five rows of five integers. In each of the five rows, an integer is at column 1, 7, 13, 19, and 25.

2. doers.dat

   This file contains personnel information. Each record is on a separate line, and contains these fields:

Field Name	Type	Length	Starting Column
AFSC	alphanumeric	5	1
Position	alphanumeric	15	11
Skill Level	integer	1	28

3. tools.dat

This file contains names of support tools. Each record is on a separate line and is composed of a single field:

Field Name	Type	Length	Starting Column
Name	alphanumeric	15	1

4. tlevel.dat

This file contains tool support information. Each record is on a separate line and contains these fields:

Field Name	Type	Length	Starting Column
Task Ptr	integer	2	1
Tool Ptr	integer	2	any (but at least one space between fields)
Support Level	integer	1	any (but at least one space between fields)

5. tasks.dat

This file contains information on the tasks in the scenario. Each record may contain several lines of data. Records are separated by a blank line. The first line of each record contains these fields:

Field Name	Type	Length	Starting Column
Name	alphanumeric	15	1
AFSC	alphanumeric	5	17
Skill Level	integer	1	25
Nominal Time	integer	2	31

The second line of each record contains the word **Inputs:**. Zero or more lines following this line contain individual input fields which are alphanumeric, length 30, beginning in column 1. After all the input fields, there is a line containing the word **Outputs:**. Zero or more lines following this line contain individual output fields which are alphanumeric, length 30, beginning in column 1. The final output field is followed by a blank line which indicates the end of the record.

Bibliography

1. Cecil, Danny A. and Joseph A. Fullenkamp. *Using Database Technology to Support Domain-Oriented Application Composition Systems.* MS thesis, AFIT/GCS/ENG/93D-03, Graduate School of Engineering, Air Force Institute of Technology(AETC), Wright-Patterson AFB, OH, December 1993 (AD-A274091).
2. Colley, Grant. "Retain the Object Model and Retain the Benefits," *Object Magazine* (May 1992).
3. French, Simon. *Sequencing and Scheduling: An Introduction to the Mathematics of the Job-Shop.* New York: John Wiley & Sons, 1982.
4. Hartrum, Thomas C., "CSCE 594 - Object-Oriented Design and Analysis." Class Notes, 1993.
5. Iscoe, Neil. "Domain-Specific Reuse: An Object-Oriented and Knowledge-Based Approach." *Tutorial from Software Reuse: Emerging Technology* edited by Will Tracz, 299–308, IEEE Computer Society Press, 1989.
6. Langloss, Randel K. *Execution Function Air Operations, Unit.* Technical Report, HQ PACAF/SCC, July 15, 1994.
7. Langloss, Randel K. *Knowledge Based Software Engineering (KBSE) Support: A Formal Model of Wing-Level C2 Applied to the 432nd Fighter Wing Misawa AB, Japan.* Technical Report, HQ PACAF/SCC, June 14, 1993.
8. Lowry, Michael R. "Software Engineering in the Twenty-first Century." *Automating Software Design*, edited by Michael R. Lowry and Robert D. McCartney. 627–654. Menlo Park, CA: AAAI Press, 1991.
9. McCain, Ron. "Reusable Software Component Construction: A Product-Oriented Paradigm." *AIAA/ACM/NASA/IEEE Computers in Aerospace V Conference.* 125–135. AIAA, October 1985.
10. MITRE. *Theater Battle Management Command, Control, Communications, Computer, and Intelligence (TBM C4I) Architecture Air to Air (F-15) Wing Operations Center (WOC).* Technical Report, HQ ACC/DRI, 1 November 1993.
11. MITRE. *Theater Battle Management Command, Control, Communications, Computer, and Intelligence (TBM C4I) Architecture Air to Ground (F-16) Wing Operations Center (WOC).* Technical Report, HQ ACC/DRI, 1 November 1993.
12. Neighbors, James M. *Software Construction Using Components.* PhD dissertation, University of California, Irvine, 1981.
13. Ogush, Mike. "A Software Reuse Lexicon," *CrossTalk*, 41–45 (December 1992).
14. Pressman, Roger S. *Software Engineering: A Practioner's Approach.* New York: McGraw Hill, 1987.
15. Prieto-Díaz, Rubén. "Domain Analysis: An Introduction," *ACM SIGSOFT Software Engineering Notes*, *15*:47–54 (April 1990).

16. Prieto-Díaz, Rubén. "Domain Analysis for Reusability." *Domain Analysis and Software Systems Modeling* edited by Guillermo Arango and Rubén Prieto-Díaz, 63–69, IEEE Computer Society Press, 1991.

17. RCA, Government Systems Division. *TAF Mission, Level, Functions Analysis: Volume I - Mission Related Functional Analysis.* Technical Report, TAC, February 1978.

18. Rumbaugh, James, et al. *Object-Oriented Modeling and Design.* Englewood Cliffs, NJ: Prentice Hall, 1991.

19. Sarchet, Michael. *Modeling Workload Effectiveness and Efficiency of Air Force Wing Command and Control.* MS thesis, AFIT/GCS/ENG/94D-22, Graduate School of Engineering, Air Force Institute of Technology(AETC), Wright-Patterson AFB, OH, December 1994.

20. Tracz, Will. "Domain Analysis Working Group Report - First International Workshop on Software Reusability," *ACM SIGSOFT Software Engineering Notes*, *17*:27–34 (July 1992).

21. Warner, Russel M. *A Method for Populating the Knowledge Base of AFIT's Domain Oriented Application Composition System.* MS thesis, AFIT/GCS/ENG/93D-24, Graduate School of Engineering, Air Force Institute of Technology(AETC), Wright-Patterson AFB, OH, December 1993 (AD-A274128).

22. Welgan, Robert L. *Domain Analysis and Modeling of a Model-Based Software Executive.* MS thesis, AFIT/GCS/ENG/93D-25, Graduate School of Engineering, Air Force Institute of Technology(AETC), Wright-Patterson AFB, OH, December 1993 (AD-A274087).

Vita

Captain Robert J. Hunt, Jr., was born February 23, 1963, in Columbus, Ohio. After graduating from Ocean Springs High School, Ocean Springs, Mississippi, in May, 1981, he accepted a 4-year AFROTC scholarship to the University of Texas at Austin. Majoring in Computer Science, he received a Bachelor of Arts degree in May, 1985, and was commissioned a Second Lieutenant in the US Air Force.

Second Lieutenant Hunt entered active duty and was assigned to the 3390th Technical Training Group in June, 1985, as an Instructor for World-Wide Military Command and Control Systems (WWMCCS) Information Systems (WIS), at Keesler AFB, Mississippi. While there, he deployed world-wide on Mobile Training Teams, instructing DoD and NATO personnel in the intricacies of WIS. In October, 1989, Captain Hunt moved to the Air Force Technical Applications Center, Patrick AFB, Florida, where he was Chief of the Nuclear Data Systems Branch. While there, he managed computer support for AFTAC's Nuclear Technology directorate, as well as manage the development and maintenance of critical software for AFTAC's Atomic Energy Detection System. In May, 1993, Captain Hunt entered the Air Force Institute of Technology (AFIT) at Wright-Patterson AFB, Ohio, to pursue a Master of Science degree with a Computer Systems major and concentration in Software Engineering.

Permanent address: 647 Fallow Ct
Gahanna OH 43230

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	December 1994	Master's Thesis

4. TITLE AND SUBTITLE
Modeling Operational Task Assignment in Air Force Wing Command and Control

5. FUNDING NUMBERS

6. AUTHOR(S)
Robert J. Hunt

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Institute of Technology, WPAFB OH 45433-6583

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/GCS/ENG/94D-09

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Capt Randel Langloss
HQ PACAF/SCE
25 E Street, Suite C-310
Hickam AFB, HI 96853

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Distribution Unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This research investigated the feasibility of applying software engineering technology to the Air Force wing command and control (C2) domain. As part of this research, domain analysis and object-oriented techniques were investigated and a specific approach was chosen to analyze the domain. Analysis of the domain resulted in an object-oriented domain model that captured the key objects, operations, and associations, and behavior of wing C2. The domain model was used to design and implement a prototype software tool that enables wing decision makers to assign resources to mission tasks and to make assessments about automation's impact on wing C2 operations.

14. SUBJECT TERMS
Software Engineering, Domain Analysis, Domain Modeling
Object-Oriented Modeling, Command & Control (C2)

15. NUMBER OF PAGES
101

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL