12-1995

An Object-Oriented, Formal Methods Approach to Organizational Process Modeling

Vincent S. Hibdon

AFIT/GCS/ENG/95D-06

AN OBJECT-ORIENTED, FORMAL METHODS APPROACH TO ORGANIZATIONAL PROCESS MODELING

THESIS
Vincent S. Hibdon
Captain, USAF

AFIT/GCS/ENG/95D-06

DISCLAIMER NOTICE

THIS DOCUMENT IS BEST QUALITY AVAILABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.

AFIT/GCS/ENG/95D-06

AN OBJECT-ORIENTED, FORMAL METHODS APPROACH TO ORGANIZATIONAL PROCESS MODELING

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

Vincent S. Hibdon, B.S.

Captain, USAF

December 19, 1995

Table of Contents

List of Figures

List of Tables

Acknowledgements

I wish to thank my research advisor, Dr Thomas Hartrum, for his expert advice and assistance. His patience throughout this ordeal was remarkable. He is definitely an expert at leading wayward students through the mine laden AFIT thesis grounds. I cannot imagine a better research advisor here at AFIT. I also wish to thank my readers, Maj Mark Kanko and Dr Robert Shock for their comments and suggestions. I now realize just how important time is and the amount of time donated by my committee member is greatly appreciated.

The real heros of this research effort are most definitely my wife, Elaine, and my children, Scott and Jordan. They began to understand my role at AFIT much quicker than I did when they began to think it odd if I were actually home at bedtime. Without Elaine's support and untiring efforts with our children, I could not have accomplished what I have so far in my life, only a small part of which was this research effort.

Vincent S. Hibdon

Abstract

This document presents a methodology for developing an organizational process model which is based on the principles of object-oriented design and formal software engineering methods. The methodology begins with the development of an object-oriented Rumbaugh model (27). The Rumbaugh model is then formally specified in Z (Zed) schemas. Finally, the Z specifications are translated into an executable model in the Software Refinery Environment^{TM}. This model is described based on the AF wing domain and developed in this domain. The proposed methodology is then shown to produce a very general model which is extendable across almost any domain. The proposed methodology is also shown to be very general and tailorable for specific domain applications.

AN OBJECT-ORIENTED, FORMAL METHODS APPROACH TO ORGANIZATIONAL PROCESS MODELING

I. Introduction

1.1 Background

With the increased capability and performance of today's computer systems, many areas of the Air Force mission have been automated. These improvements have appeared in widespread areas from the office to the battlefield. In particular, the AF wing command and control (C^2) area can benefit from the use of automated systems to better evaluate and prepare their resources for contingencies. In today's environment, commanders could benefit by making use of a formal mechanism which describes how a wing conducts its mission. This formal mechanism could help evaluate the readiness of their wings and the effectiveness of various resource allocation configurations.

Operations personnel and researchers have begun to realize that insufficient attention has been given to the area of AF wing C^2. This can be seen from actions taken at 432nd Fighter Wing, Misawa AB, Japan. The 432nd initiated efforts to model the C^2 activities of its organization through the use of Knowledge Based Software Engineering techniques (17). The absence of a formal representation of how a wing performs its mission limits the ability of wing decision makers to evaluate the effectiveness of the wing's performance. This deficiency means key personnel do not have the information necessary to assess the impact of C^2 automation on wing operations (12). Furthermore, system developers and maintainers do not have a formalized knowledge base to use in the specification and design of new computer-based C^2 systems or in the modification of existing systems.

One way to formally capture the knowledge about key objects, operations, and relationships relevant to AF wing C^2 is through domain analysis (28). Performing a domain analysis on an AF wing would produce a domain model consisting of information to describe the domain structure and rule bases necessary to capture domain knowledge. Wing and headquarters personnel could

then use the domain model in conjunction with other aspects of Knowledge Based Software Engineering (KBSE) to analyze unit readiness and effectiveness and to decide how to best allocate available resources. Furthermore, system developers and maintainers would have a knowledge base to establish the design of new C^2 systems or to aid in the modification of existing systems. By using formal methods of software engineering, the creation of a domain model of wing C^2 operations is possible. This model would serve as a management tool for Air Force decision-makers.

The KBSE research group at the Air Force Institute of Technology (AFIT) continues to conduct research into various aspects of knowledge based software engineering and formal methods. Recent research by Hunt and Sarchet developed a partial domain model of AF wing C^2 operations(12, 28). Their domain model displays the basic relationships between a wing's tasks, workers, and tools. This model along with formal methods techniques can be used to develop a tool which would assist wing decision makers in applying automation to the C^2 process. This would demonstrate that the formal methods of software engineering technology can be effectively applied to AF wing C^2. Additionally, this model could form the basis for a more general model for use in other organizations including those outside of the Air Force.

1.2 Problem

Current methods for evaluating how a wing performs its mission leaves Air Force decision makers with less knowledge about the current status of their commands than might be possible through a formal model. Key wing personnel could benefit from the use of tools to assess the impact of C^2 automation on wing operations and to understand what requirements are best satisfied with computer-based C^2 systems. This might also help system developers and maintainers in the specification and design of new computer-based C^2 systems or in the modification of existing

systems. Therefore, a domain model of the AF wing to use in conjunction with formal methods might prove to be be a useful tool for aiding decision makers in the assessment of their wing. The knowledge gained could also be useful to system developers and maintainers as a repository of reusable information in the form of applications and/or system specifications.

This research focuses on improving the AF wing domain model produced by Hunt and Sarchet (12, 28) by developing more realistic representations for tasks, resources, and workers. The improved model is then used to better analyze the task and resource assignments using formal methods. Metrics are defined to show the extent to which automation impacts the organization.

Figure 1 shows an overall "big picture" of the process used by organization leaders in the assessment of their organization. In this process, *Create New/Update Existing Task Model* uses the given taskings and resources of an organization to develop a new task model or to update an existing task model for that organization. Next, *Assign Resources to Tasks with Automation* allows the decision maker to consider different configurations of the organizational taskings with certain tools being modeled as automation aids. Following that, the organizational leader can *Analyze the Model* using organizational metrics to evaluate the tasking assignment. If the automation of specific tools indicates a significant impact on the organizational performance, then a Software Requirements Specification (SRS) can be developed; otherwise, the resource to task assignment model is implemented. The new SRS, if any, could then be used as input to the *Reuse-Based Application Development* Section of the process.

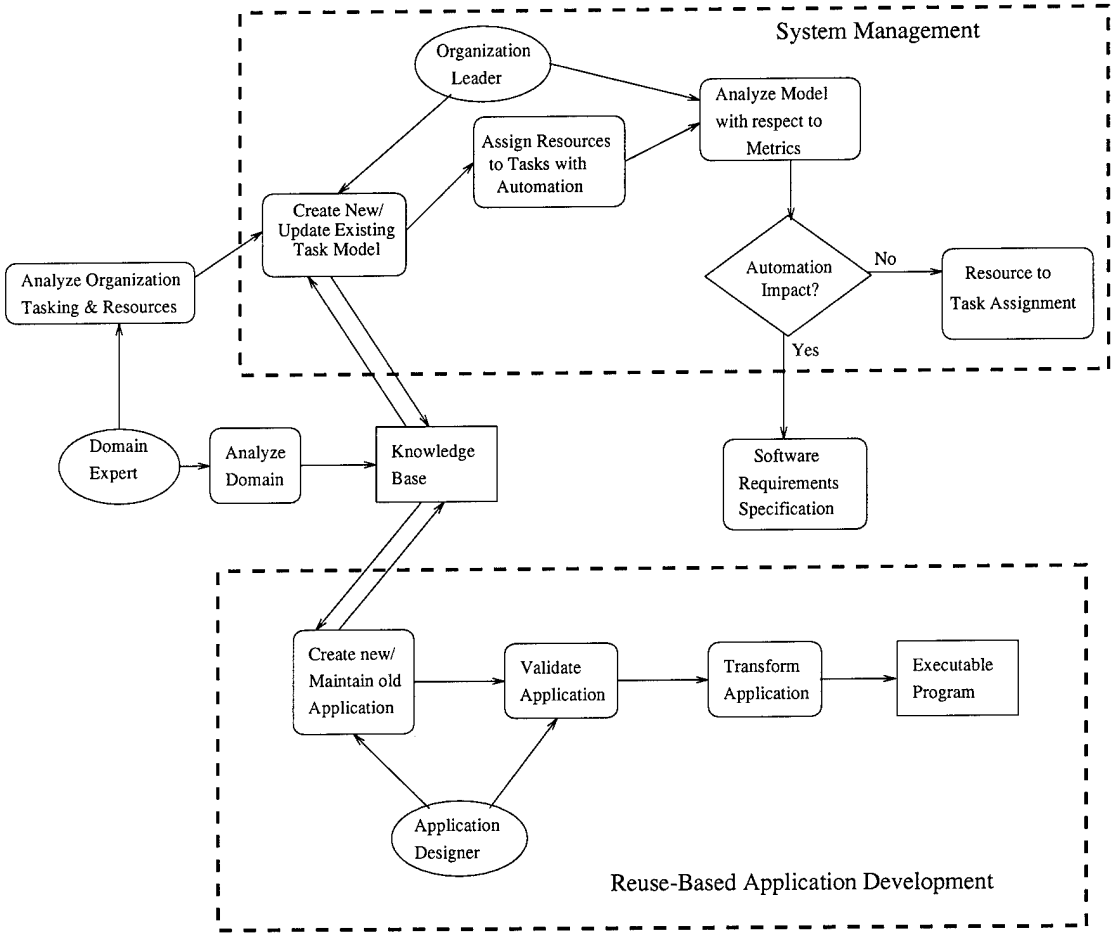

Figure 1. Formalized Model Development Process(32)

Problem Statement:

The AF has not made use of existing knowledge based software engineering and formal methods technology which could provide improved information concerning the status of their commands by using the formal representation and assessment of an AF wing.

1.3 Research Objectives

The overall objective of this research is to show the value of using KBSE and formal methods in the representation and assessment of an Air Force wing tasking model. The following are specific research objectives:

- Analyze the previously existing model of the AF wing. Improve the object model, dynamic model, and functional model using the formal algebraic language Z.
- Verify and, as much as possible, validate the modified model.
- Extend the formal representation of the model to contain metrics and domain-specific constraints.
- Translate the Z formal specifications to an executable specification using the Software Refinery Environment^{TM}.
- Generalize the model by identifying the domain specific constraints and dependencies within the model.
- Identify the parts of the formal model which would have to be modified for a new domain and evaluate modifying the model for new domains using formal methods.
- Show that formal methods can improve the process of evaluating and assessing wing metrics.

- Identify benefits provided by the use of formal methods (e.g. prototyping, providing an interface to other platforms, reuse of specifications, using new user specs to derive a new model).

1.4 Approach

To meet the proposed research objectives the following approach was followed:

1. *Gained an Understanding of Knowledge Based Software Engineering* - Conducted a survey of current software engineering literature to gain insight into what KBSE entails, to include domain analysis and modeling and formal methods. This was an essential step, for without a thorough understanding of KBSE, the full potential of this research could not have been realized.
2. *Reviewed Operations Research Areas which Impact the Domain of Interest* - Several areas within the operations research field have significant impact on the current model and the interaction between its components. Identification and analysis of these areas was necessary to improve the current model by injecting realism. This also led to better overall system metrics.
3. *Analyzed Current Model* - Performed an in-depth analysis of the current Air Force wing domain model proposed by Hunt and Sarchet. Assessed its current status and identified any shortcomings. Sponsor feedback was not available in this area.
4. *Performed Domain Modeling* - Analyzed past and present work in the C^2 area to determine applicability of this research project and to avoid any duplication of effort. This involved the search for areas of expansion and improvement for the current model (i.e. scope, metrics, etc.) as well as incorporating expansions and refinements into the domain model as appropriate.
5. *Implemented Domain Model* - The previous model existed in Rumbaugh and Z form, with an Ada simulation for implementation. The improved model is implemented in a formal executable language. This phase of the research entailed reviewing several formal languages and selecting one for use in implementation.
6. *Verified and Validated Domain Model* - Domain experts were not available to be consulted for help in validating the model to determine whether it really captured the behavior of the AF wing. This would be a necessary step in determining the true reusability of the model within the AF domain and in evaluating the accuracy of this model. Verification of the model was done internally by researchers, and formal methods were used as much as possible.
7. *Identified Domain Specific Constraints of the Model* - The model of the AF wing was analyzed to determine which portions of the model are inherent to the AF wing domain. Once these particular constraints were identified, they were tagged as the parts of the model which must be tailored for a particular domain of interest.

8. *Generalized the Model* - Within the chosen formal representation language, the model was abstracted to the necessary level to be of general use. The previously identified domain specific constraints were then specialized as needed for a particular domain.
9. *Demonstrated the Feasibility of Transforming the Model to Differing Organizations* - Using the generalized model and methodology, an approach was proposed to demonstrate the reuse of the methodology and model.
10. *Displayed Benefits of Model and Formal Methods* - Showed how the domain analysis process and formal development of organization models could improve Air Force wing operations. Also showed how formal methods lead to reusable models and specifications which can span several domains.

1.5 Initial Assessment of Past Effort

This research is based on an initial research effort put forth by Hunt and Sarchet (12, 28). Their research was based on a proposal by Langloss entitled *Knowledge Based Software Engineering (KBSE) Support: A Formal Model of wing-Level C^2 Applied to the 432nd Fighter wing, Misawa AB, Japan* (17). The informal domain model of the AF wing level command and control tasks is a first attempt to actually model how a wing operates. Rumbaugh's object modeling technique (27) was the basis for the model, depicted in Figure 2 (11). The model represents the following object classes.

- **Task**- a piece of work which must be accomplished by a single person.
- **Worker**- a resource to accomplish a task.
- **Tool**- a form of automation to assist a *Worker* in accomplishing a task.

These objects are related via the following associations.

- **Assignment**- represents the assignment of a specific *Task* to a specific *Worker*. Each *assignment* has the attributes *time* and *accuracy* which indicate the nominal time and accuracy to which the *Worker* can accomplish the specific *Task*.

- **Assigned-** indicates that a specific *Tool* has been designated to be used on a specific *Task*.

- **Supports-** indicates to what extent a specific *Tool* can aid in the accomplishment of a specific *Task*. This extent is indicated through the attribute *level*.

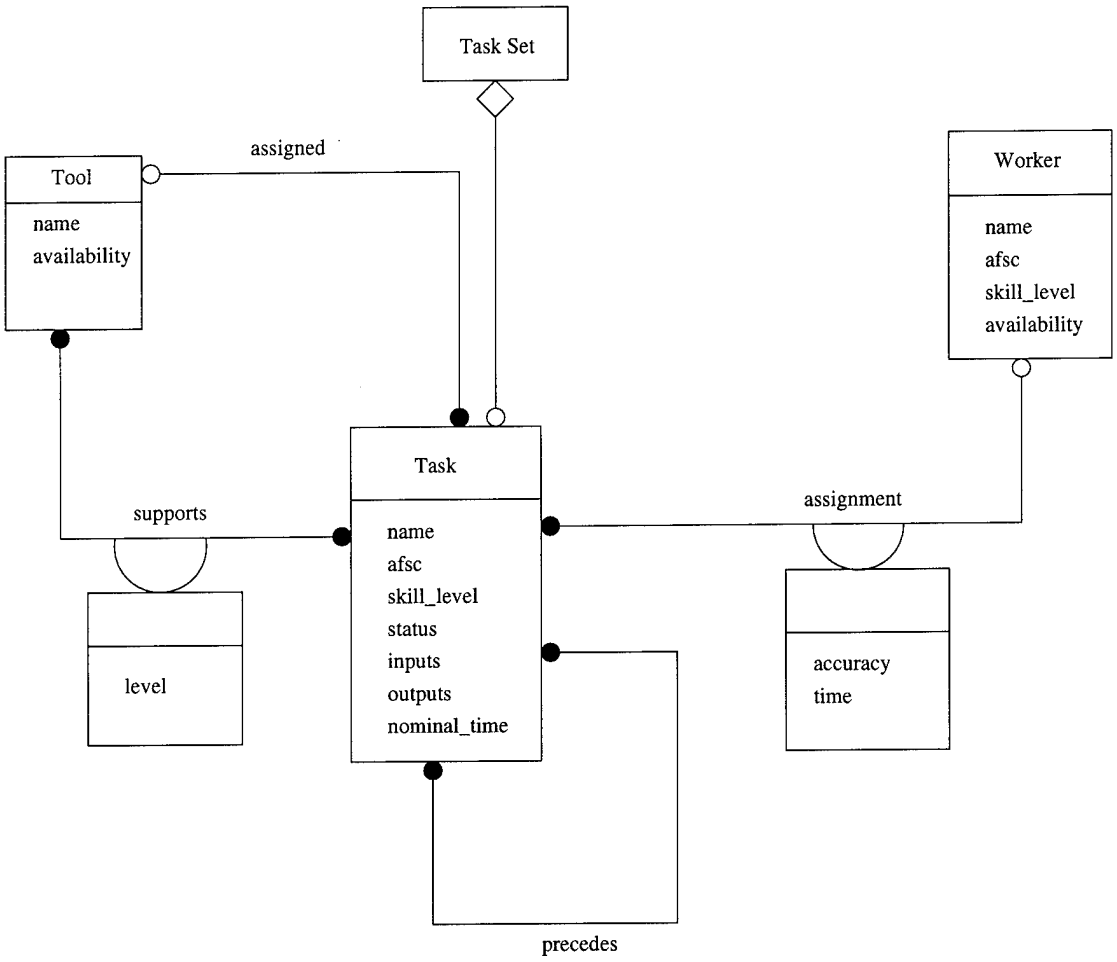

Figure 2. Object Model

This model demonstrated the feasibility of analyzing the various configurations of *Tool* to *Task* and *Worker* to *Task* assignments. It also exposed the difficulty associated with extracting detailed domain information from a complex real-world situation by non-domain experts (11). However, this model appears readily extendable from modeling a particular Air Force wing to modeling any

typical Air Force unit, and possibly other general organizations. Figure 3 shows the organization of a typical Air Force wing and the scope of the domain model (17). Further augmentation of the Rumbaugh model includes a formal model using the formal programming language "Z" (zed)[1] (11). This formal representation allows a precise methodology for expressing the actual behavior of the wing through the use of formal model-based specifications.

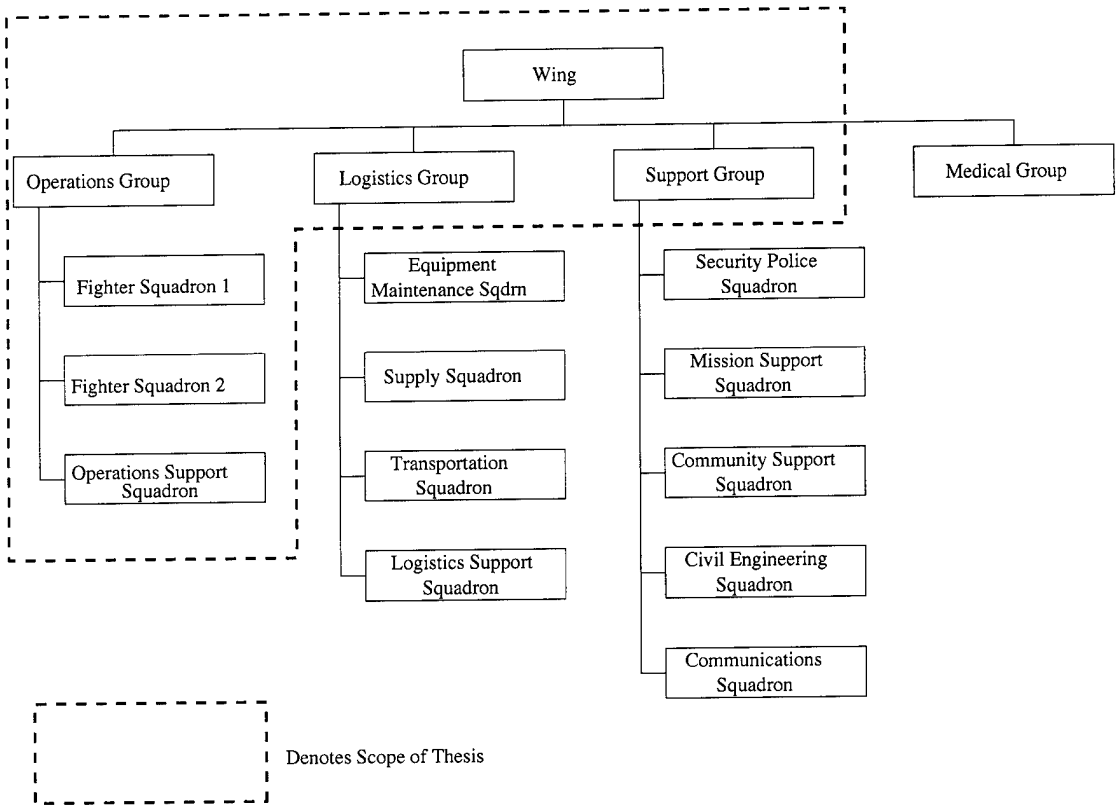

Figure 3. Typical wing Organizational Chart (17)

Although a good start, this model has some shortcomings. Its representation in Z has limited flexibility. The ability to generate executable code from a Z formal model is limited. Therefore, the wing's behavior can be modeled, but at this time it cannot be observed. Because of this, Ada was

[1] Z is an formal specification language based on predicate logic, sets, and functions.

used to the develop prototype tools for calculating unit metrics. These tools do not lend themselves to rapid prototyping or to easily changing model behavior.

The model's metrics were unrealistic and overly simplified. This was due to the lack of domain knowledge and the unavailability of domain experts. The *time* metric and *accuracy* metrics are based solely on a comparison of *Task* to *Worker* AFSC and *Worker* skill level, respectively. Further refinement of these metrics are needed for a more precise representation of the organization (11).

The time and accuracy metrics were not thoroughly validated by domain experts. This validation must be done for the domain model to meet the user's end needs. The domain experts should specify any additional metrics which would be useful in analyzing the effects of tool and worker assignments on unit effectiveness and readiness. The simplified metrics can lead to unrealistic values for the sponsor's specified metrics of overall unit effectiveness, unit efficiency, and unit readiness (17).

The model has the limitation of assigning only one *Worker* to a *Task*. In reality, several *Workers* may actually perform a *Task*. The model also limits the assigning of only one *Tool* to a *Task*. In many cases, more than one *Tool* may be desired to accomplish a *Task*. The model would be more realistic if it contained a way to model the assignment of multiple *Workers* and/or *Tools* to a single *Task*. The model has neither the ability to respond to real-time taskings nor the ability to handle multiple independent *Tasks*. Both of these situations would be the norm in almost any real-world situation. Therefore, the model would be greatly enhanced if it possessed the ability to handle these situations.

1.6 *Assumptions*

Prior to conducting this research effort the following assumptions were made. First, sufficient domain knowledge would be available through various sources. These sources included, but were not limited to, the sponsors, documentation, and various domain experts. This knowledge was necessary for complete analysis of the domain model, choosing appropriate expansion directions, and validating the value of the model. Also, the sponsors should provide access to valid and up-to-date information on past, present, and anticipated C^2 domain analysis activities. Another assumption which was made was that the chosen platforms would be available for development in the formal development phase of the research.

1.7 *Scope*

The modeling of the entire AF wing is not the concern of this research effort. The model's main focus is in showing that automation can have an impact on a portion of the wing. The focus is on the fighter squadron-level and does not model all activities at the group- or wing-level. The formal Refine implementation is limited to the Object Model. These scoping decision are necessary to conduct the proposed research on an organization of this size within the time available.

1.8 *Organization of Thesis*

Chapter II contains a literature review of KBSE technology, domain modeling, and operations research topics which are relevant to this research. Chapter III describes and follows the development of the domain model which is the focus of this research. Next, Chapter IV contains an analysis of the methodology and the resulting model which was developed. Chapter V discusses

the generalizations and reusability of the domain model. Finally, Chapter VI draws conclusions regarding this research and offers some recommendations for future research in this area.

II. Literature Review

2.1 Introduction

The focus of this review is knowledge based software engineering (KBSE) technology and how this technology can be applied to the Air Force wing. This review begins with a discussion of previous research in the area of domain analysis and modeling. This is necessary since domain analysis and modeling is a fundamental part of establishing a knowledge base. Next is a review of the theses which form the basis for this research. This review also include discussion about several facets of operations research which impact the research. Finally, how this all leads to a specific domain modeling approach is discussed.

2.2 Systems Modeling

"Studies have revealed that the most difficult part in modeling is not the creation of program code, but rather the creation of the system abstraction on which the code is based" (21). The current state of system development is very fragmented and conducted in an ad hoc manner. The process of system development needs to be a fully integrated process which allows a seamless transition from phase to phase in the system development process. Systems modeling may provide the means to progress towards this seamless type of development (7).

There are three main areas in systems modeling: dynamic performance modeling, rapid prototyping, and executable specification. "The use of these techniques in the development of large, complex systems helps the designers predict the behavior of the system, assess the feasibility of the design, and seek the optimal solution from the design space" (7).

Dynamic performance models are either analytic or simulation. If an exact solution may be obtained using a mathematical method then the model is analytic. In a technical report about systems modeling, Choi said,

"Simulation uses the numerical logic models of a system to examine its behavior over time and under different conditions and scenarios. Executable specification is a formal representation of the system which captures the functional, behavioral, and implementational aspects. Finally, rapid prototyping is a representation of the system which bears closer resemblance to the final system than either dynamic performance modeling or executable specification.

Most of the problems in modeling and analysis technology exist, not because of a lack of tools or methodologies, but because of a lack of integration and a failure to address system level issues. There are many tools that address some system problems or some phases of system development. Although these tools alone are not enough to develop all systems, when integrated, they could provide many capabilities for system development. Currently, this integration has been lacking, including the consistent usage of terms such as modeling, executable specification, and prototyping" (7).

2.2.1 Fundamentals of Modeling. A model is some form of an abstraction of a system which can be used to analyze certain aspects of the system. "Models may capture the function, behavior, structure, and/or implementation of a system" (7). In general, models are hierarchical in nature. They are used to develop a solution to a problem, analyze the proposed solution, and optimize the solution.

There is no single model which can capture all aspects of an entire system. Models can be used to capture the parts of the system which need investigation. However, models suffer from several problems. First, they tend to drastically increase in complexity in state-dependent systems. Second, mapping from one model to another model is quite difficult. Finally, it is very difficult to model the human aspect of the system.

Models have suffered from lack of widespread use due to their inherent complexity. They are also limited in models' ability to back-annotate changes to the model. As changes are made,

they need to be annotated all the way back to the requirements. Most modeling techniques fail to accurately model the human aspects of the system. The term "humanware" views humans as resources which are allocated to accomplish system tasks (7).

2.2.2 Executable Specification. "Executable specification, in its most fundamental definition, captures the functional and behavioral descriptions of the system" (7). A complete executable specification can serve as the design specification and the single representation of the system. This would be very useful in system modification since only one representation is needed for system modeling and design specification. All changes made during modification would automatically be reflected in the design specification. "An executable specification may be used for the evaluation of functional and behavioral correctness and consistency. Executable specification is being considered as a promising approach to rapid prototyping software" (7).

2.2.3 Prototyping. Recently, much use has been made of prototyping to improve software development. Software development through the use of prototyping is based on an evolutionary view of software development. It makes use of early functioning versions of the system (19). A prototype can be classified as a model of a system due to the fact that it is actually an abstraction of the system. It can contain very limited amounts of functionality or, on the other extreme, it may be fully functional, lacking only full testing. Prototypes are different from simulation in the fact that simulations only mimic the behavior of the system while prototypes act as the system. Prototypes are useful in testing the system as well as checking the feasibility of the system and its components (7). Prototyping allows the system to be tested under its operating conditions with limited capabilities. Rapid prototyping is a way of prototyping which provides incrementally added functionality during system development with a very fast turn around time.

2.3 *Knowledge Based Software Engineering*

Knowledge Based Software Engineering (KBSE) refers to the practice of developing and maintaining software systems using sound engineering principles, reusable software components, domain models, and domain analysis. Reuseable components constitute basic building blocks of a domain model. By employing sound engineering principles in conjunction with domain modeling and analysis, software systems can be developed rapidly and reliably.

It has been said that large productivity increases can be seen by employing intelligent computer-based tools to aid in the knowledge-intensive area of software engineering (1). KBSE will also save time and money by eliminating human introduced errors to the software development process. A vital part of KBSE is domain analysis and modeling.

2.3.1 Domain Analysis and Modeling. The initial area of concern in creating a knowledge base is capturing and representing the knowledge for the area of interest. This is done through domain analysis and modeling. A domain is a particular area of interest or study. The model is a formalization of the similarities and differences among members of a particular domain. A domain model is the result of conducting a domain analysis which consists of acquiring knowledge about a domain or reasoning about an area of interest(domain) (33).

Tracz (30), McCain (22), and Prieto-Díaz (25) have all proposed methods to conduct domain analysis and modeling. However, Prieto-Díaz's method is the most prevalent. This method contains the following steps:

1. Pre-Domain Analysis
   (a) Define Domain
   (b) Scope Domain
   (c) Identify sources of Domain Knowledge

(d) Define Domain Analysis Goals and Guidelines

2. Domain Analysis

   (a) Identify Objects and Operations

   (b) Abstract 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 (25)

The Prieto-Díaz method of domain analysis produces a model very similar to the Rumbaugh object model. Using the Rumbaugh object model, the members of the domain are represented as objects. Each object has associated with it a set of attributes or descriptors. Rumbaugh also uses associations, which may have attributes, to represent the relationships or interactions between the members of the domain (27). The Rumbaugh model is further discussed in Section 2.5.1.

2.3.2 Goals of a Domain Modeling Methodology. The goal of a modeling methodology is to develop an applicable model for a particular domain of interest which provides the user with pertinent information regarding the organization. The methodology should provide a smooth transition from the domain information to the model representation. The transitions should occur in almost seamless phases from the developers viewpoint.

Hunt (12) and Sarchet (28) present a basic domain model of the Air Force wing structure using a combination of existing techniques. Their approach is based on a combination of the domain analysis and modeling methods presented by Prieto-Díaz (25), Tracz (30), and the Rumbaugh OMT (27). The steps of their technique follow:

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

2.4 Formal Methods in Software Development and Maintenance

What are formal methods? The use of formal methods involves writing a formal specification of system requirements along with mathematically proving certain properties about these specifications. A formal specification is a precise definition of what the software system is intended to do. These specifications can then be used to mathematically and systematically derive a computer program to fulfill the requirements via manipulating the specifications. This program is then at least partially verifiable through mathematical argument (9). It has been said that "It is hard to fudge a decision when writing formal specifications, so if there are errors or ambiguities in your thinking, they will be mercilessly revealed" (9).

A method is said to be formal if it has a sound mathematical basis and, therefore, provides a systematic rather than ad-hoc framework for developing software. The acquisition of formal domain knowledge is difficult since it requires extensive collaboration between domain experts and knowledge engineers. However, it does provide many advantages. Using formal methods can help to assure correctness from specification to implementation. It also ensures programs are correctly implemented to fulfill the user's intentions. Formal methods can also be used in conjunction with automated reasoning tools, thus enabling the reuse of knowledge-based software engineering components (20).

Formal methods can be used in varying degrees within the development of a software system. Specification and implementation are often viewed as separate issues. First, the user totally

specifies the system requirements in a formal language at a high level of abstraction, free from any implementation language details. Then, as a separate development phase, the program is generated, taking into consideration the implementation details.

The program implementation of the specifications might be done totally automatically via an automatic programming tool, semi-automatically through the use of a transformation system, or manually using software engineering techniques. However the system is developed, the program arrives at the implementation through the use of the specification of requirements and verifying that the implementation is valid for the requirements. It should be noted that the steps of specification and implementation may be somewhat intertwined. If limitations are noticed in requirements during development of the implementation, the requirements analysis phase should be revisited.

By modifying user specifications and redeveloping the entire system from the formal specifications, maintenance becomes redevelopment and is easier to perform. If the formal specification and the semantics used to implement the specifications are in an executable form, then the specifications have an observable behavior and the implementation can be validated from the specifications. This enables the specification to be utilized as a prototype software system. The formal specifications are much easier to reuse than specific implementations in code. One of the biggest benefit of formal methods is that formal specifications lead to the ability to automate the software development process or at least gives rise to the use of an automated software development assistant (2).

It must be realized that the use of formal methods does not guarantee that a software system is perfect because it cannot be guaranteed that the user's specifications are always correct. This is due to the behavior of the real world. The real world cannot always be modeled and does not always react as modeled. However, using formal methods does provide some degree of proof.

One can be mathematically sure that the system fulfills the specified requirements, which is very useful. Formal proof techniques allow the errors in the specifications to be exposed. This enables the identification of errors early in the development cycle which equates to less costly fixes to the system. More benefits provided by using formal methods are fitness for purpose, ease of construction, ease of maintainability, and better visibility. Formal methods increase the chances of the right software being built by ensuring interaction between the system designer and the user regarding the system specifications when the specifications are being formalized. This leads directly to decreased development and maintenance costs (9).

2.5 Model Representation

2.5.1 The Rumbaugh Object Modeling Technique. Object-oriented modeling and design is a fairly new concept which places emphasis on the organization of software models making use of real-world concepts. This concept is based on a fundamental construct called the object. The object combines both data structure and behavior into a single entity. The Rumbaugh Object Modeling Technique (OMT)[1] is based on the object-oriented modeling and design concept. All phases of development, to include analysis, design, and implementation, are covered by the OMT.

Since the Rumbaugh OMT model is depicted using a basic set of graphical representations and is based on well-known object-oriented foundations, it is extendable across domains and allows modeling, design, and implementation using a consistent set of object-oriented concepts and notations. In the OMT methodology, three distinct models are used to describe a system: the object model, the dynamic model, and the functional model.

[1]The Rumbaugh Object Modeling Technique will hereby be referred to as OMT and the resulting model the Rumbaugh model.

2.5.2 Object Model. The object model is composed of objects and the associations between the objects. Object diagrams describe the structure of the objects in the system to include their identity, their relationships to other objects, and their attributes. An *object* is a distinguishable entity while an object *class* represents a group of objects with similar attributes, common operations, and common relationships. An object class is represented with a rectangle and attributes along with their type declarations. Figure 4 is an example of an object class and individual object instances of the object class.

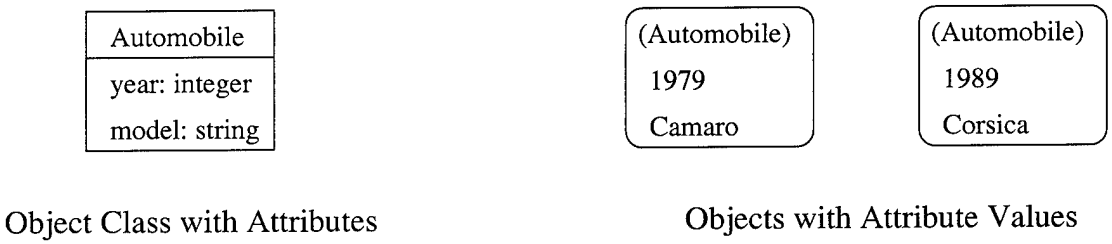

Figure 4. Object Class and Object Instances

2.5.2.1 Associations and Links. The relationships between object classes are established using *associations.* An association describes the group of potential *links* between all of the instances of the related object classes. Figure 5 is an illustration of this example.

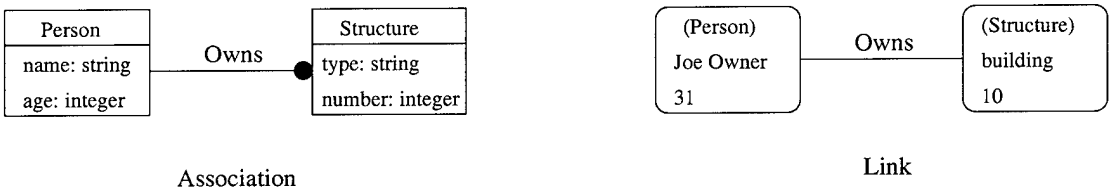

Figure 5. Association and Link Examples

In Figure 5, there is a solid black circle on one end of the association. This is the method used to specify the multiplicity of the association between objects. This representation allows associations to be from one-to-one up to many-to-many. It also is used to represent required and optional

membership in the links and associations. Table 1 describes the multiplicity representations in the Rumbaugh OMT method along with the formal representation using Z.

Table 1. Multiplicity Representation (10)

	Domain	Range	Specification	Symbol	Rumbaugh	LaTeX
m:n	either	either	Relation	$A \leftrightarrow B$	A •——• B	`\rel`
n:1	optional	optional	Partial Function	$A \nrightarrow B$	A •——○ B	`\pfun`
n:1	required	optional	Total Function	$A \rightarrow B$	A •—— B	`\fun`
n:1	optional	required	Partial Surjection	$A \twoheadrightarrow B$	A 1+——○ B	`\psurj`
n:1	required	required	Total Surjection	$A \twoheadrightarrow B$	A 1+—— B	`\surj`
1:1	optional	optional	Partial Injection	$A \rightarrowtail B$	A ○——○ B	`>\!\pfun`
1:1	required	optional	Total Injection	$A \rightarrowtail B$	A ○—— B	`>\!\fun`
1:1	required	required	Bijection	$A \rightarrowtail B$	A —— B	`>\!\surj`

2.5.2.2 Dynamic Model. The dynamic model represents the system as it changes over time by specifying and implementing the system's control. This is modeled using state diagrams. It is used to describe the the sequence of operations in a system without consideration for what the operations do, what they operate on, or how they are implemented. All actions in the state diagram correspond to functions in the functional model. Figure 6 displays a simple dynamic model of an Automobile.

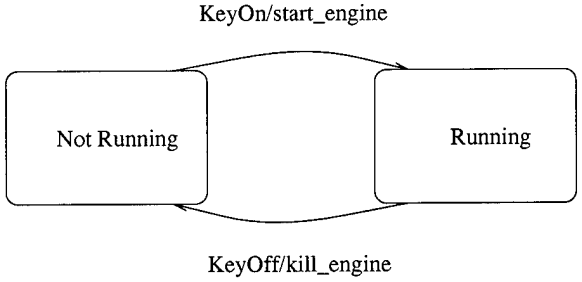

Figure 6. Automobile Dynamic Model

2.5.2.3 Functional Model. The functional model describes how the data within the system is changed. This part of the model uses data flow diagrams. The functional model captures what the system actually does and is not concerned with when or how the system does it. It describes the parts of the system which change values through functions, mappings, and functional dependencies. The functions within the functional model correspond to actions within the dynamic model and operate on the objects within the object model (27). Figure 7 is an example of a simple functional model.

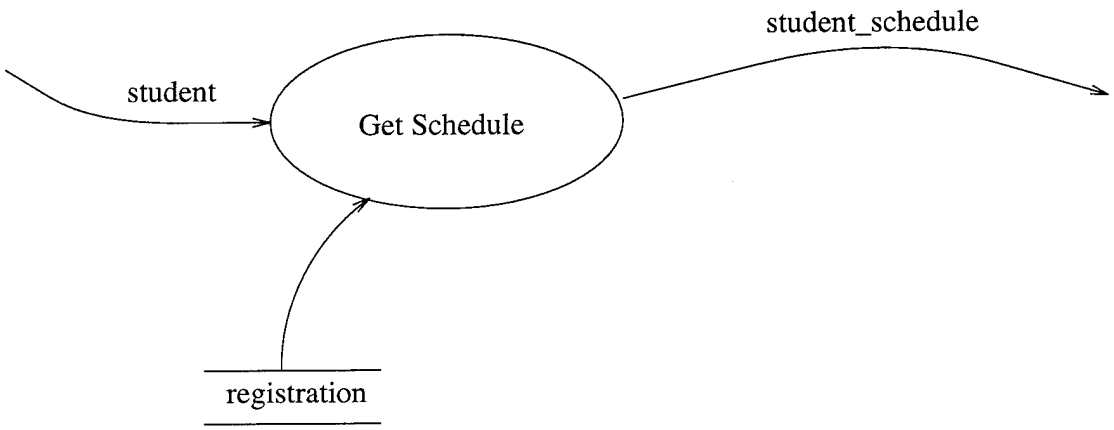

Figure 7. Functional Model

2.5.3 The Z Language. One manner in which to formally capture the information contained within the Rumbaugh OMT model is through the use of a formal specification language such as *Z* (29). *Z* is a formal language used to abstractly specify a system using mathematical constructs to represent the system's states and operations. This type of mathematical model allows the system developer to reason about the correctness of the system and its specified behavior. Since it is based on fundamental mathematical principles, the system's behavior can be abstractly represented and the system specifications produced are precise, unambiguous, concise, and amenable to proof (31).

The Z language is used to formally define the specifications of a system in a mathematically based notation. A system is described using modules called schemas which contain data definitions and constraints on the data (24). The following Z schema is a representation of the Automobile in Figure 4.

Automobile

$year : \mathcal{N}$

$model : \mathrm{seq}\, CHAR$

2.6 Modeling Organizational Activities

A process is simply a group of related activities for the accomplishment of a particular task. A process model is used to describe all of the objects, activities, and the relationships between them. A process model done correctly can directly support the understanding, evaluation, and improvement the process (21). A process model has many uses within an organization. There are several methods for modeling an organization and it activities. These methods vary and are based on goals of the model as well as the organization.

2.6.1 Generic/Specific Modeling. Research by Lung, et al, presents the concepts and practical experiences of generic/specific modeling. Generic/Specific (GS) modeling is a domain analysis approach to produce a discrete-event simulation in the manufacturing domain. Also presented is a meta-model based on the generic/specific approach from the software engineering perspective. Also discussed are the domain analysis lessons learned from the approach. Traditional simulation modeling suffers one of the same problems as traditional software development– the lack of reuse. Modelers tend to view each simulation as a new project and start to build the model from scratch (21).

One of the main advantages to the GS approach is that it reduces the amount of time and workload of the simulation modeler. This is accomplished through the ability to select a model from a number of generic models and modify the selected model as necessary as opposed to building a new model from scratch. A generic model is a one designed to meet a basic set of objectives for selected applications. A group technology (GT) classification scheme is used to classify or group these generic models (21). After the selection of a generic model, the modeler can then input more specific detailed data to derive a specific model. This model is then interpreted and linked to form an executable model. This is done by the SIMAN architecture which is a simulation program processor (21).

This approach has proven to be a very effective example of reuse within the discrete-event simulation domain. Reuse is accomplished by allowing the modeler to build a simulation model despite the fact he or she possesses only basic skills and knowledge. This reusability leads directly to increased productivity, reduced costs, improved quality, and enhanced reliability. This approach of providing model selection and modification, as opposed to building a model from the ground up, also avoids errors in creation and coding (21).

This approach integrates bottom-up and top-down analysis with domain modeling. Objects, operations, and relationships of systems are identified through the use of a bottom-up analysis. Knowledge structures or domain architectures are built and whole model reuse (not just code reuse) is promoted through the application of top-down synthesis (21).

2.6.2 Business Process Reengineering. The fundamental goal of business process reengineering is to identify and to correct fundamental deficiencies in the business process. This should be done prior to and opposed to only identifying tasks for automation. Once the deficiencies have

been corrected, automating the appropriate tasks will further improve the process. Basically, the idea is to avoid using technology for technology's sake (23).

The beginning steps in business process reengineering are quite similar to the total quality management philosophy which examines processes to identify non-value or limited value added steps. However, business process reengineering actually redefines the process based upon the findings. The overall goal of business process reengineering is to eliminate the assembly-line, sequential mentality and promote the idea of parallel process thinking where many people have access to available information simultaneously (23).

This idea of business process reengineering is needed because most businesses have had their processes in place for the most part since early in their conception. As the need arose for process change, the institutions simply adapted instead of redesigning the overall process. Then with the inception, or inclusion, of automation, existing tasks were identified to be automated without reevaluating the overall process. This resulted in a faster process but not necessarily a better process (23).

Another step which must be accomplished early in business process reengineering is to move the business' tasks from a sequential mode into parallel structured tasks. This can be accomplished by having separate units within the organization which perform the same tasks or by having the separate units perform different tasks which eventually come together. This parallel structure will allow for greater, more efficient, output of the overall process (23).

2.6.3 Job-Shop Development Model. Current software development models are based on the waterfall paradigm of software development. This paradigm has specific and differing phases for the software to progress through on its way to completion. These types of models do not lend

to accounting for software rework which is caused by errors, requirements changes, and software enhancements (18).

A better way to model software development is through the use of a model based on the job shop concept. This approach has been successfully used in industry to model manufacturing processes. This approach will allow a more detailed representation of software development by breaking each ongoing project into smaller batches and breaking the batches into even smaller jobs. This will allow the view that any individual parts of the software system may be in differing stages of development at any one time. This is a much better view of the real world of software development (18).

Many organizations which are involved in the development of several similar projects often divide their staff among the projects. The job-shop model is suitable for such organizations, especially if they have a well defined software development process. The particular job shop model can be calibrated using "appropriate probability distributions for items such as personal productivity, sick leave, and vacation, and for establishing scheduling priorities (18)."

In a job-shop model, a certain number of tasks are processed through a certain number of stations. Each station is made up of machines and/or people. The tasks involved all have their own independent processing requirements, a processing order, and a scheduling priority. This concept can be easily applied to the software development environment. Software modules are all subject to most of the following stages: requirements analysis, design, coding, testing, fielding, and maintenance. In fact, the modules may undergo iterations within the stages. The job-shop view of software development sees each software project as a group of batches, which are related jobs, and a job as a specific software item needing to be accomplished. This model depicts each worker

individually, which leads to a stochastic network of jobs and batches being routed through the life cycle phases of software development (18).

Prior to the start of any new project, the project is assigned a priority. Also, the number of anticipated batches is estimated. Next, the processing path is specified and finally, the intra- and inter-batch dependency chains are specified. Nonproductive times for workers are represented by filler jobs. A probability distribution is used to generate the filler jobs. An overall scheduler must be used to maintain the list of available jobs and their related information, along with the workers and the tasks they are qualified to accomplish (18).

Probability distributions are used to estimate when and where errors will occur and to where the batch in error must return. Calibrating the model involves much time accounting. Each worker's productivity must be recorded for the correct probability distribution in the model. Also, nonproductive times, error-capture rates, the phase to which rework is sent, maintenance times and efforts, etc. must be recorded and predicted. The data recorded is analyzed under a best-fit analysis and found to fit a gamma-distribution. The initial overhead incurred in setting up such a model is costly. The required set-up time is proportional to the complexity of the work flows, the number of workers, and the number of ongoing projects (18).

Many organizations will find that the job shop model has the potential to be an effective simulation tool for project effort prediction. It provides the flexibility to model stations through which a task must pass as a person, group, or people using machines. This will allow researchers to "investigate the effect of new technologies by making appropriate assumptions about model components (18)."

The job-shop model has several limitations. First, a large amount of historical data may be needed for calibration of the model. Second, much effort is required for the initial setup. Third, the tasks of the scheduler are so intensive that the processing time is lengthy, which makes testing and debugging time consuming. An improved scheduling algorithm could lead to faster processing times (18).

2.6.4 Improving an Industrial Software Process. The technology which supports software development has made impressive strides in the recent past; however, estimating and improving software development is still difficult. This shortcoming has led to much research in the area of software process improvement. An assessment of a mid-sized Italian company found that "[while] it was reasonably easy to identify the technical areas where improvement is needed (e.g. configuration management and requirements engineering), it is much more difficult to understand how to conduct the analysis of a company in order to identify its organizational and strategical deficiencies, and how to identify and propose reasonable changes and improvement actions accordingly (3)."

The proposed method, by Bandinelli, for process modeling is based on the SLANG (SPADE Language) process modeling language. SLANG is based on a Process-centered Software Engineering Environment called SPADE (Software Process Analysis, Design and Enactment). SLANG is a domain specific language for modeling software processes. "SLANG is based on high-level petri nets and is given formal semantics in terms of a translation scheme from SLANG objects into ER nets, a mathematically defined class of high-level Petri nets that provide the designer with powerful means to describe concurrent and real-time systems (3)." However, petri nets have been shown to be very complex for even the smallest of projects.

2.6.5 IDEF Models. The ICAM Definition Methodology (IDEF) is based on Integrated Computer Aided Manufacturing (ICAM). The construction of an IDEF model has been said to be the initial component in a comprehensive process modeling effort. The initial IDEF model is IDEF0 which is developed to model a wide variety of systems to include hardware, software, and people performing activities. The IDEF0 model has three major components– diagrams, text, and a glossary. All three components are cross-referenced to each other. The diagrams consist of boxes and arrows. The boxes represent functions and the arrows represent either inputs, outputs, controls, or mechanisms. An active verb phrase which represents the function is used to identify the box. "Inputs (I) enter the box from the left, are transformed by the function, and exit the box to the right as an output (O). A control (C) enters the top of the box and influences or determines the function performed. A mechanism (M) is a tool or resource which performs the function. The interfaces are generally referred to as the ICOM's (16)."

"IDEF3 was created specifically to model the sequence of activities performed in a manufacturing system. An IDEF3 model enables an expert to communicate the process flow of a system through defining a sequence of activities and the relationships between those activities (16)." The IDEF3 process description language has two main components, the process flow description and the object state transition network description. IDEF3 diagrams are developed by cross-referencing the two main components (16).

The IDEF3 model uses three components to make up the process flow description– units of behavior (UOB), links, and junction boxes. Each activity or function occurring in the process is represented using a UOB. The UOB's relationships with each other are modeled with three types of links. These links are precedence links, relational links, and object flow links. "Precedence links

express simple temporal precedence between UOBs. Relational links highlight the existence of a relationship between two or more UOBs, however, no temporal constraint is implied. Object flow links provide a mechanism for capturing object related constraints between UOBs and carry the same temporal semantics as a precedence link (16)." The branching within a process is modeled using *and, or*, and *exclusive-or* junction boxes.

The IDEF3 model uses object state transition network (OSTN) diagrams to model object state changes relative to the process flow description. OSTN diagrams are composed of nodes and arcs. Each object in the model may or may not have associated with it an OSTN diagram. The arcs of the diagram represent the possible state transition paths that may be taken by the object. The transition arcs may have associated with them certain scenarios, UOBs, or other OSTN diagrams. The components of the IDEF3 model can also be decomposed similarly to the IDEF0 model. There do exist some formal methods for analyzing the structure of an IDEF model, one of which is described in (16).

2.7 Domain Specific Issues

The particular domain of interest for this research is the AF wing. Within this domain, there exist several issues which are unique to the AF wing. The AF wing structure is basically a worker-task type domain. This type of relationship leads directly to literature which is concerned with representing and classifying both tasks and workers as well as describing the assignment of workers to tasks. Classification is a basic method used in many domains to aid in the understanding of a particular system; however, there is no existing method which can be used for all problems (21).

2.7.1 Task Classifications. To improve the attributes associated with tasks, it is important to be able to identify tasks which are somehow related and to group the tasks accordingly. Certain groups of tasks are inherently similar, while other tasks are drastically dissimilar. Tasks may be grouped along many boundaries. These boundaries include area of specialty, difficulty presented, and mental and physical abilities required just to name a few. Huo and Kearns state "It is necessary to classify all jobs into clusters, or job families, in accordance with the similarity among their requirements (13)". These clusters or groups would help identify meaningful task attributes for use in many domain models.

Huo and Kearns also suggest matching employees to jobs based on three input files. The first file represents the individual employee's qualifications and contains performance ratings, technical/functional skills and associated ratings, educational background, career history, and personality/attitude attributes. The second file holds the individual employee's constraints which include preferences and interests, mobility limitation, language limitations, current job level, and current pay level. The third and final file is the job profile file which contains job requirements, prior incumbents, job cluster ID, and search criteria. A search mechanism is used to evaluate all employees for an open job position. This search is based on the data in the three files. Making the best set of worker-to-task assignments is a desired outcome.

2.7.2 Worker Attributes. Any useful measures that can identify and predict the amount of work which can be accomplished by an individual on a certain task would be very valuable in domain modeling. "The military has interest in performance measurement for many of the same reasons as business: to support proper personnel decisions and enhance the output and operation of the organization (6)."

In 1989, the Human Resources Directorate of Armstrong Laboratory began a study of the job performance of workers. The goal of the study was to identify, develop, and evaluate the current technologies used in the measurement of job performance. This research led to the area of productive capacity (PC). PC could be used to aid in the forecasting of aptitude and experience levels needed by a worker in order to accomplish a given task or quantity of work. The findings of this research concluded that experience level was the most important influence on the PC of an individual (6). Pre-employment testing is an efficient and effective way to predict job performance. "Cognitive ability or intelligence testing seems to be the best known predictor of job performance (4)."

Knowing the abilities of particular workers would also be a valuable attribute within the domain model. Research by Faneuff (8) produced a study of the effects of mechanical aptitude and job experience on the performance of aerospace ground equipment (AGE) mechanics. Their job performance was expressed as PC and was found using estimated performance times on selected job tasks. This research's PC measures were obtained using airmen within the career field of AGE completing 50 typical tasks. Aptitude measures were the actual scores of the airmen on the Armed Services Vocational Aptitude Battery (ASVAB). This study indicated that experience is a much more reliable predictor of PC than was aptitude. It also found no indication of an aptitude/experience interaction (8). Therefore, the experience level and productive capacity of a worker warrant consideration as valuable worker attributes in a domain model. This seems to indicate that the associations between worker and job objects might benefit from derived attributes which describe the amount of time and expected accuracy between a particular worker and a particular job.

Faneuff's research suggests the use of three distinct competency levels – fastest possible, average or normal, and slowest possible (6), and indicates that a job might benefit from the additional attributes *average time, best time*, and *worst time* to complete.

A recent study of the effects of mechanical aptitude and job experience on the performance of AGE mechanics found that the job performance of the mechanics was influenced by both the experience level and aptitude of the particular worker.

2.7.3 Job Performance Measures. The ability to measure the performance of a worker on a particular job could lead to useful worker attributes within the domain model as indicated in AHRL-TR-87-15 (14). This report contains a description of the job performance measurement classification scheme the military currently uses. It describes the Air Force Human Resources Laboratory's (AFHRL) research program aimed at the development of individual job performance measures. It also describes a classification scheme which was developed by the AFHRL. Analysis of this study indicates that the ability to measure job performance accuracy would be a valuable attribute in the worker to task association of the Air Force wing domain model (14).

2.7.4 Skills Classification. Categorizing human resources is a very difficult task. Human talents are characterized by high degrees of complexity and heterogeneity. They cannot be easily classified in the same manner as normal capital resources. Classifying the qualities of a human resource are a prerequisite for making use of a computer platform to aid in the decision making process. This classification process is a major obstacle for the managers of personnel. Employee's qualification data can consist of education background, work experience, age, sex, and race. These qualifications tend to come in the form of "packages" of a set of these qualities. The diversity of

the various strengths and weaknesses within the packages make it hard to evaluate the individual human resource and to make objective comparisons between individuals (13).

It is also difficult to specify the combination of skills, knowledge, and attributes which are desirable for a person to have so that he or she will be successful at a given task. Many feel that employee performance and attitude are important characteristics. These attributes are often very subjective. However, these have been used to predict future job performance with more accuracy than education and possessed skills (13).

2.7.5 Unit Readiness. The sponsor of this research desires the ability to assess the impact of automating certain aspects of the AF wing's mission on certain overall organizational metrics. One of the metrics specifically proposed was unit readiness. A review of this area led to several reports concerning overall unit readiness. However, these reports were primarily Army and Navy outlooks. Based on these reports, the following discusses overall unit readiness and how it can be determined.

2.7.5.1 Initial Unit Readiness Dimensions. In an initial assessment, Research Triangle Institute defines unit readiness as "The capability of an Army unit to perform the mission for which it is organized." Their initial definition was based on nine dimensions. These dimensions are (15):

1. Equipment - Does the unit have all of the necessary equipment on hand and is the equipment operational?
2. Personnel Strength - Do the authorized and actual assigned slots of the unit's personnel match in both job and rank?
3. Training Status - How well trained are all personnel in mission essential tasks? Rating is based on the time necessary to have all personnel trained to proficiency.

4. Supervision - How proficient are the skills of the officers and non-commissioned officers in the organization?
5. Collective Performance - How well do groups of workers perform collective tasks?
6. Unit Performance - How well does the entire unit perform on normal requirements and to impromptu taskings?
7. Higher Level Support - How much support does the unit receive from higher level and external organizations?
8. Cohesion - How much overall loyalty, pride, and positive interaction do the members of the unit possess?
9. Stability - How stable are the personnel who make up the unit?

2.7.5.2 Modified Unit Readiness Dimensions. Through data collection, independent analysis, and review, the initial dimensions were modified to the following group which are more specific classifications of the initial dimensions.

1. Adherence to Standards
2. Ammunition, Supplies, Materials, and Other Equipment
3. Care and Concern for Families
4. Care and Concern for Soldiers
5. Cohesion and Teamwork
6. Communication Within Unit
7. Cooperation/Coordination with Other Units
8. Emergent Leadership
9. Higher Echelon Support
10. Leadership
11. Mission Performance
12. Personnel Capabilities
13. Personnel Deployability
14. Physical Fitness Program
15. Physical Security/Vigilance
16. Training Program
17. Unit Weapons

18. Vehicles/Transportation

However, this report did not detail how these individual dimensions are used to calculate an overall unit readiness factor or even what scale is used for the measurement.

Based on this research, it seems that the unit readiness metric is based on measures which cannot be assessed or impacted by the use of automation to complete a job or task. However, a good starting point in assessing this metric within the current model could easily be based on Personnel Strength and equipment readiness. Almost every organization in existence has a manning plan for the organization which states the number of persons needed for the organization to accomplish its mission along with the specific qualifications each should possess. By simply comparing the actual personnel to the needed personnel an initial readiness metric could be established. This metric can easily be added to the object model by adding a Manning object which has a readiness attribute. This attribute could be derived from a comparison of the Workforce to the Manning required for the Organization.

2.8 Conclusion

The focus of this review was knowledge based software engineering (KBSE) technology and how this technology can be applied to the Air Force wing. This review concluded with a specific domain analysis and modeling approach to be used in this research. Also discussed were many ideas related to process modeling.

This review has identified several areas of potential improvement in not only the model but the process to be used in developing the model. First, several key attributes were identified which will help improve the realism of the objects within the model. Also, the relationships between

the objects can be improved using the information gained by the review. These include the the relationship between the tool and job object classes and the worker and job object classes. These improvements are discussed in detail in Chapter III.

2.8.1 Role of Domain Analysis and Modeling in this Research. There are many aspects to domain analysis and modeling which have been brought to light through this literature review. Based on the research and information gathered, the following domain modeling approach is followed in the remainder of this thesis. This approach is loosely based on the Job Shop model and the information contained in Section 2.6.3.

- Describe the objects and their attributes as thoroughly as possible
- Identify and describe all relationships/associations between the objects
- Compose a Rumbaugh model of the system
- Fully describe the Rumbaugh model in Z notation
- Transform the Z notation into executable Refine
- Verify and validate the model
- Generalize the model

2.8.2 Model Development Methodology. This literature review has helped in the development of a methodology to aid in the development of a domain model for the AF wing. This proposed methodology is composed of three distinct phases. The first phase includes the gathering of all the necessary domain information and putting it into the Rumbaugh OMT model. The first transition, to phase two, is to capture the desired behavior of the model in a formal context

through the use of the Z schemas. This step helps facilitate the correct frame of mind for the model developer to achieve the third and final phase of development– the Refine transformation. This phase requires the developer to think functionally rather than procedurally to develop a model's behavior and allow the Refine environment to capture the behavior. Once a model's specifications have been defined within the Refine environment, modification of the model's behavior is achieved through a simple change of specifications, allowing the Refine environment to handle the execution details. This approach is fairly generic and allows for the development of similar models based on the same types of organizations, that is, organizations which inherently are composed of *Worker*, *Task*, and *Tool* objects, with assignments of these objects. Chapter III follows the implementation of this modeling approach and examines the resulting model.

III. Developing the Domain Model

3.1 Introduction

In this chapter, the AF wing domain is analyzed and a domain model is developed using the approach discussed in Section 2.8.1. The new model is based on the initial model developed by Hunt and Sarchet and the analysis of their model. In this chapter, the goals of the model and the approach for developing the model are detailed. The Rumbaugh model, Z specifications, and the Refine implementation are also presented.

3.2 Goals of the Model

When developing a domain model, it is necessary to know how the model is to be used and what information is needed from the model. This model will be used to assess overall organization metrics such as unit readiness, unit effectiveness, and unit efficiency. Other uses of the model will include evaluating the impact of automation tools on a fully assigned *Project* and evaluating the status of *Tasks* and *Projects*. The model will also be useful in the assessment of various proposed schedules for individual *Worker*, *Tool*, and *Job* assignments and their impact on the *Organization*. The model should also be easily modified for use by other organizations.

3.2.1 Modeling Issues. There are many different and difficult issues that arise in the development of any real-world domain model. Issues include what values to assign to the time and accuracy attributes of a job given that a certain *Worker-Job assignment* has been made. Another issue is how to model the impact of assigning a tool to a job and the effect it has on the time and accuracy attributes of the job. Also, many of the higher level objects of a domain model may have many derived attributes; the derivation of these attributes must be dealt with correctly.

3.2.2 Model Representation. A domain model is the result of a domain analysis. Therefore, it is important to choose a model representation up front. The informal model is based on Rumbaugh's OMT (27), briefly described in Section 2.5.1, including Rumbaugh's three specified components: the object model, the dynamic model, and the functional model. The Rumbaugh model is then converted into the Z schema language. The properties of Z allow for a formal approach to describing the model through the use of predicate logic and mathematical symbol descriptions. This is followed by the transformation of the Z specifications into executable Refine constructs, a model closer to the actual programming language of the desired final product.

3.2.3 Model Maintainability. Modification and maintenance of the model should be quick and easy. Changing the behavior of the model should simply involve changes to the formal specifications. An evolutionary or incremental approach to system development should be simple to implement with the availability of a functional prototype system to check behavior throughout the development.

3.2.4 Executable Platform Analysis. During this research, an assessment was made of the abilities of several KBSE tools available at AFIT. These platforms included the transformation system by Wabiszewski (31), Kestrel Institutes SpecWare (5), and Refine (26). Wabiszewski's transformation system needs further development to actually transform Z into Refine code. It provides the abstract syntax tree needed; however, the transformations stop at the function signatures. To utilize this tool in the context desired would require the completion of the transformations necessary to fully define the function bodies. This effort was deemed beyond the scope of this research effort. The SpecWare environment is still in the process of further refinement and was assessed to be beyond the ability of the researcher. The Refine environment is the most feasible to use in

demonstrating the value of the formal model in the development and assessment of an organization model.

3.2.5 Refine. Refine directly supports many of the common data types encountered to include *integer, real, boolean, symbol, sets, sequences, maps, objects* and many more. An object in Refine is represented as a class. A class is declared as a variable of the type object-class. All objects must be declared in the realm of some superclass, therefore, the highest level objects are declared as a subtype of the superclass user-class. Each attribute of an object is defined as a map from the object to the attribute domain.

3.3 Methodology for Developing the Model

One goal is to make the model more representative of the real-world through further domain analysis. A thorough analysis of the domain is necessary for a successful model to be implemented. This requires an understanding of the domain of interest and access to domain experts. The operations research literature exposed many potential areas for improvement. The approach used in this research to develop a domain model of an AF organization is based on Rumbaugh's OMT and the principles of Job Shop Scheduling as discussed in Section 2.6.3.

3.3.1 Rumbaugh Model Development. The first step in the approach is to fully describe all of the objects within the domain. In the domain of the AF, this includes many objects, all of which need attributes which describe their properties and states. The next step in the approach is to identify all relationships which might exist between the objects within the domain. These relationships are then transformed into associations and further analyzed to determine the attributes of the associations if necessary. After all of the objects and association, along with their attributes,

are been identified, a Rumbaugh object model is developed. Based on the object model and domain information, the dynamic and functional models are then developed. This completes the first phase of the development.

3.3.2 Z Formalization of the Object Model. The three parts of the Rumbaugh model (the object model, dynamic model, and functional model) fully describe the model; however, they do not formally show any correctness of behavior or exhibit any from of executability. The *Z* language was chosen to use as an intermediate step in the formalization of the domain model prior to implementing the model in Refine. Although not directly executable, the *Z* language gives the ability to mathematically specify the behavior of the model using a wide range of mathematical constructs. The descriptions include the object and association attributes along with the predicates which fully describe the behavior of the objects and associations within the overall model. To facilitate an understanding of the overall model, it is easiest to start with descriptions of the most basic objects. In many schemas the attributes and predicates are relatively simple and self-explanatory. However, in some of the more complex schemas the predicates are sometimes confusing and require further explanation.

3.3.3 Transforming Z to Refine. Now that the model's behavior is fully defined in the *Z* specifications, phase three is ready to begin.

3.3.3.1 Defining Objects in Refine as Classes. Using the Software Refinery Environment to develop the model starts with a bottom-up approach. The first step is to implement all of the objects in the Rumbaugh and *Z* models as Refine objects. This involves creating a separate file for each object. The object itself needs to be declared as a variable of type object-class. The

attributes of each object are then declared as mappings from the object to the defined attribute type.

Once all of the objects are declared, the methods for accessing and updating the attributes are added to each object file. This is done through the use of the *function* construct along with the *transform* construct.

3.3.3.2 Defining Associations in Refine as Objects. Defining associations within the model starts at the highest level object, in this case, the *Organization.* This is done since the associations are between objects within the model and the *Organization* has visibility into all of the objects in the system either directly or indirectly. The associations are defined as a mapping from the *Organization* to a set of links. Each individual link is initially declared as a subtype of user-object. The link is then given attributes which designate the objects it associated between along with any link attributes attached to the association. Next, a specialized object is declared for each association to represent a set of these links. This is to allow for specialized operations unique to each particular association. These association set objects are declared as subclasses of *Container.*

The *Container* superclass allows for generic operations which are needed for all of the associations such as AddItem, RemoveItem, GetItem, etc.

3.3.3.3 Adding Interaction Among Objects. Within the highest level aggregate object, the *Organization*, aggregate methods or functions between objects can be defined. Since the aggregate objects contain many set and sequence constructs, it is convenient and powerful to use the built-in set and sequence operations provided through the Refine environment. By making

use of the set and sequence former notation, many powerful functions can be defined and utilized. This allows the environment to do many mundane tasks such as go through a set of objects and return the ones which meet a certain criteria without the need to write code to loop through the set and check for the condition while storing the correct members in a temporary set to return. The Refine environment takes care of the "how"; all the designer supplies is the "what". The following is an example:

```
%% Functions to find and assign all of the tasks and jobs of an organization
function Getorg-taskset(o1: Organization)=
  TRUE --> org-taskset(o1) =
    reduce(union, {ptask_set(p1) | (p1: Project) p1 in org-projectset(o1)})
```

In this example, it is specified that the set to return is the set of all *Tasks* which are in the *ptask_set* (a *Project's* set of *Tasks*) for all of the *Projects* which are in the *Organization's* set of *Projects.*

3.4 AF Wing Domain OMT Model

3.4.1 Analysis of Existing Model. The organizations within the AF structure are inherently composed of *Projects* to complete, a *Workforce* to complete the *Projects*, and *Tools* with which to aid the accomplishment of particular parts of the *Projects.* A domain analysis of the AF *Organization* has identified uses of a domain model in the assignment of *Workers* to accomplish *Jobs* and *Tools* supporting the accomplishment of these *Jobs.*

Changes to the model include adding an approach to calculating overall unit readiness and the addition of an intermediate level work unit to allow for a *Project* object to be composed of *Task* objects which are further composed of *Job* objects. The Hunt and Sarchet model was limited to assigning a single *Worker* to a *Task* and single *Tool* to a *Task.* In reality, several *Workers* may actually perform a *Task* in a given *Project* and several *Tools* may also be used. This is accounted

for through the intermediate level *Task* object which is composed of one or more *Job* objects which have been defined as the smallest unit of work to be accomplished by a single *Worker*. The modified model also provides for intermediate level metrics to use in the derivation of higher level objects' attributes.

The initial model depicted in Figure 2 contained only two parts: the object model and the functional model. This initial model was modified to include more objects, more associations, and a dynamic model. These modifications were done to increase realism and to increase the usefulness of the model. This model might then be extendable to more general organizations beyond the AF.

3.4.2 Object Model Description. The preliminary object model was modified 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 not readily available. Refinements were based on domain documentation as analyzed by local personnel.

In phase one of the development process, the Rumbaugh model was developed. The improved object model is shown in Figure 8, without object attributes.

3.4.2.1 Object Identification. Based on a thorough domain analysis, the following objects were identified:

- *Organization* The highest level aggregate object with derived attributes for organizational level metrics.
- *Project* A composition of *Task* objects which represents an overall entity of work from start to finish.
- *Task* An intermediate level unit of work composed of *Job* objects used to collect intermediate level metrics through derivable attributes.
- *Job* The smallest unit of work which can be accomplished by a single *Worker* with or without the use of a *Tool*.

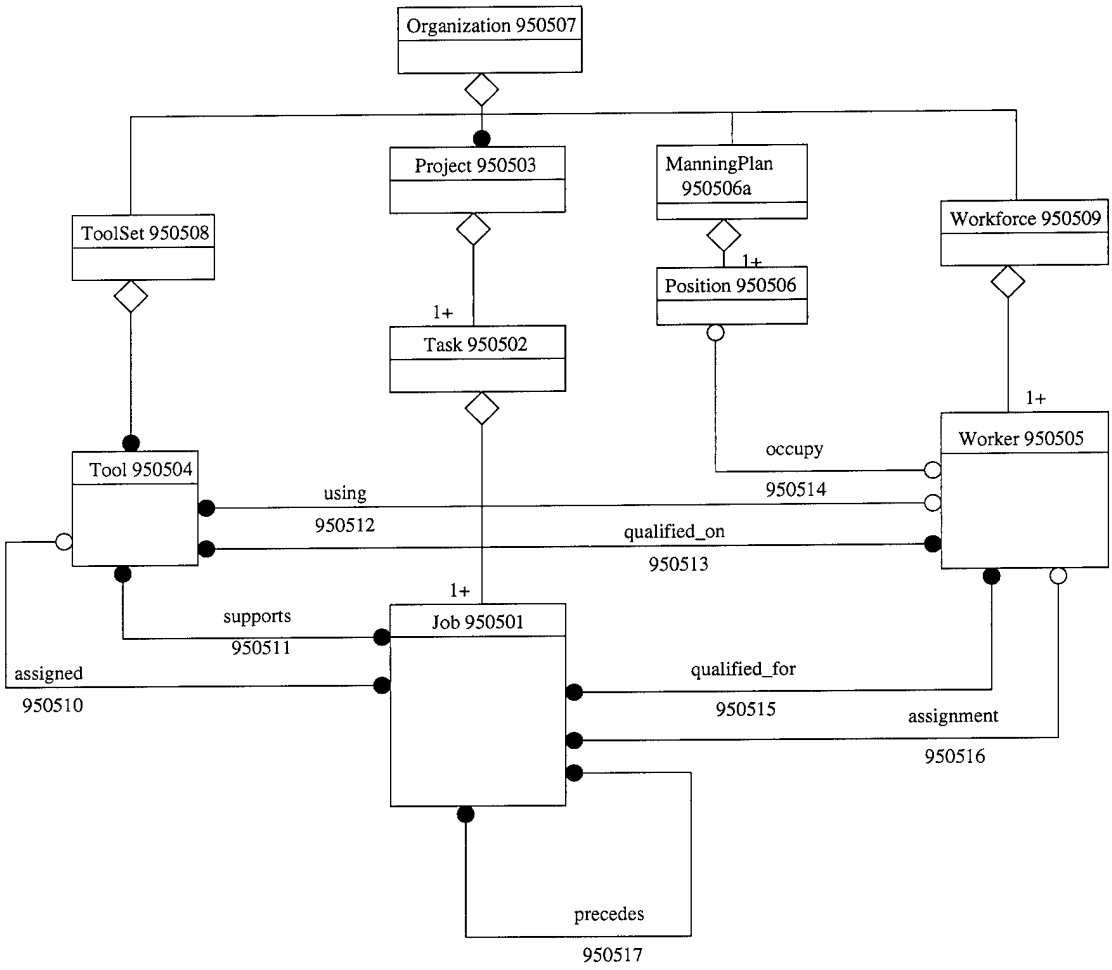

Figure 8. Object Model (without attributes)

- *Workforce* The total set of all of the *Workers* in an *Organization.*
- *Worker* The object used to represent each individual worker within the *Organization.*
- *Toolset* The total set of all of the *Tool* objects in an *Organization.*
- *Tool* The object used to represent each individual tool within the *Organization.*
- *ManningPlan* The set of all *Positions* in an *Organization.*
- *Position* The object used to represent each individual position within the *Organization.*

3.4.2.2 Worker. The *Worker* object is a resource which composes the *Workforce.* The experience, education, and aptitude attributes were added as a result of research concerning worker attributes and productive capacity as discussed in Section 2.7.2. The attributes of the *Worker* object are:

- name: the name of the individual *Worker*
- afsc: the *Worker's* Air Force Specialty Code
- skill_level: the skill level held by the *Worker*
- availability: Boolean indicator of whether the *Worker* is available for assignment or is currently assigned to a *Job*
- experience: indicator of the amount of experience a *Worker* has in this afsc
- education: the education level of the *Worker*
- aptitude: the evaluated aptitude of the *Worker*

3.4.2.3 Workforce. The *Workforce* object is composed of a set of *Worker* objects and has the following derived attribute:

- force_readiness: an assessment of the overall readiness of the *Workers* in the *Workforce*

3.4.2.4 Tool. The attributes of the *Tool* object are:

- name: the name of the *Tool*
- status: the current condition of the *Tool*, either go or no go
- availability: Boolean indicator of whether the *Tool* is available to assist in accomplishing a *Job* or is currently in use and unavailable
- afsc_set: the set of afscs for *Worker* objects who are able to use the *Tool*
- skill_level: the desired skill level of the *Worker* objects who may use the *Tool*

3.4.2.5 ToolSet. The *ToolSet* object is composed of a set of *Tool* objects and has the following attribute:

- tool_readiness: an assessment of the overall readiness of the tools in the *ToolSet*

3.4.2.6 Position. The attributes of the *Position* object are:

- number: the number of the *Position*
- afsc: the afsc desired for the *Position*
- skill_level: the desired skill level for the *Position*

3.4.2.7 ManningPlan. The *ManningPlan* is composed of a set of *Position* objects and has no attributes.

3.4.2.8 Job. The *Job* object is the smallest piece of work defined within the model and has the following attributes:

- name: the identifier of the particular *Job*
- priority: the priority of the *Job* relative to the other jobs in the *Task*
- afsc_set: the set of afscs which are allowed to perform the *Job*
- complexity: the difficulty of the *Job* from zero (least difficult) to 100 (most difficult)
- skill_level: the *Worker* skill level required to perform the *Job*
- status: the status of the *Job*, either ready, waiting, in-process, or finished
- inputs: a list of data items which must be available before the *Job* can be started
- outputs: a list of items which have been accomplished after the *Job* is finished
- average_time: the amount of time required for an average *Worker* to accomplish the *Job* without the use of a *Tool*
- best_time: the optimal amount of time for a *Worker* to accomplish the *Job* without the use of a *Tool*
- worst_time: the greatest amount of time required for the least qualified *Worker* to accomplish the *Job* without the use of a *Tool*
- automated_time: the amount of time required to accomplish the *Job* if it is completely automated
- average_accuracy: the amount of accuracy for an average *Worker* accomplishing the *Job* without the use of a *Tool*

- best_accuracy: the optimal accuracy for a *Worker* accomplishing the *Job* without the use of a *Tool*
- worst_accuracy: the lowest accuracy for the least qualified *Worker* to accomplish the *Job* without the use of a *Tool*
- percent_complete: a percentage of how complete the *Job* is at the current time
- start_time: the time when the *Job* actually started or is scheduled to start
- end_time: the time when the *Job* actually finished or is scheduled to finish
- accuracy: the calculated accuracy of the *Job* based on how it is accomplished and the attributes of the objects which accomplish it (i.e. *Tool* and *Worker*)
- time: the calculated amount of time required to accomplish the *Job* based on how it is accomplished and the attributes of the objects which accomplish it (i.e. *Tool* and *Worker*)

The *Job* object includes an attribute *name* which indicates the particular *Job*. The *average*, *best*, and *worst time* and *accuracy* attributes were added to improve the actual assignment of time and percentage attributes of the *qualified_for* association between *Workers* and *Jobs*. Attributes start_time and stop_time were added to the *Job* object to facilitate the actual timing of the accomplishment of a *Job*.

The *time* and *accuracy* attributes of a *Job* are determined based on how the *Job* is to be accomplished, either with or without a *Worker* and with or without a *Tool*, or if the *Job* is automated. These attributes are further impacted by the attributes of the *Tool* and *Worker* if they are involved in the accomplishment of the *Job*. A highly-skilled or well-educated *Worker* will have the impact of decreasing the time attribute of the *Job* if assigned to accomplish the *Job* and vice versa. The accuracy attribute can be affected in a similar manner with a highly skilled *Worker* having the effect of increasing the accuracy. The assigning of a *Tool* to aid in the accomplishment of a *Task* has a similar effect on the time and accuracy attributes of the *Job* based on the attributes of the *Supports* association.

3.4.2.9 Task. The *Task* object is defined as a set of one or more *Job* objects. The *Task* object includes an attribute name to identify individual tasks. The attributes of the *Task* object are:

- name: the identifier of the particular *Task*
- priority: the priority of the *Task* relative to the other *Tasks* in the *Project*
- status: The current status of the *Task*
- inputs: a derivable list of items which must be available or accomplished before the *Task* can begin
- outputs: a derivable list of items which are available or have been accomplished after the *Task* is completed
- time_so_far: amount of time spent accomplishing the *Task* up to the current time
- percent_complete: a derivable percentage of how complete the *Task* is so far
- number_of_jobs: the total number of *Jobs* which comprise the *Task*
- jobs_complete: the number of *Jobs* which comprise the *Task* which have been accomplished
- start_time: the time when the *Task* was actually started
- end_time: the time when the *Task* was actually finished
- wall_time: total time elapsed to complete the *Task*
- total_time: sum of all *Job* times in the *Task*

3.4.2.10 Project. The *Project* object is defined as a set of one or more *Task* objects. The *Project* object includes an attribute name to identify individual *Projects.* The attributes of the *Project* are:

- name: the identifier of the particular *Project*
- priority: the priority of the *Project* relative to all other projects in the *Organization*
- status: The status of the *Project*
- time_so_far: amount of time spent accomplishing the *Project* up to the current time
- number_of_tasks: the total number of *tasks* which comprise the *Project*
- tasks_complete: the number of *tasks* which comprise the *Project* which have been accomplished
- percent_complete: a derivable percentage of how complete the *Project* is so far
- number_of_jobs: the total number of *Jobs* which comprise the *Project*
- jobs_complete: the number of *Jobs* which comprise the *Project* which have been accomplished

- start_time: the time when the *Project* was actually started
- end_time: the time when the *Project* was actually finished
- wall_time: total time elapsed to complete the *Project*
- total_time: sum of all *Task* times in the *Project*
- accuracy: the average accuracy of all of the *Tasks* which comprise the *Project*
- proj_jobs: set of all *Jobs* involved in this *Project*
- task_set: set of all *Tasks* involved in this *Project*

3.4.2.11 Organization. The *Organization* object is defined as a set of zero or more *Project* objects, a *ToolSet*, a *Workforce*, and a *ManningPlan*. The attributes of the *Organization* are:

- unit_effectiveness: a derived metric which describes the *Organization's* overall effectiveness.
- unit_efficiency: a derived metric which describes the *Organization's* overall efficiency.
- unit_readiness: a metric which describes the *Organization's* overall readiness.

3.4.3 Object Model Associations. The interrelationships among the objects are depicted in the OMT method through the use of associations. Each association may or may not have attached to it a set of attributes. The values of the attributes of all associations are derived from the attributes of their associated objects. In the current model, the following associations between object classes are identified. These associations have their multiplicity defined using the symbology in Table 1. Each individual association is described in detail in the following sections.

- *using*: Tool $\twoheadrightarrow$ Worker
- *qualified_on*: Tool $\leftrightarrow$ Worker
- *assigned*: Job $\twoheadrightarrow$ Tool
- *supports*: (Tool $\times$ Job) $\rightarrow$ SupportsAttr
- *assignment*: Job $\twoheadrightarrow$ Worker
- *qualified_for*: (Worker $\times$ Job) $\rightarrow$ QualifiedAttr
- *occupy*: Position $\rightarrowtail\!\!\!\!\!\twoheadrightarrow$ Worker
- *precedes*: Job $\leftrightarrow$ Job

3.4.3.1 Using Association. The *using* association links a *Tool* to a *Worker* and shows that the *Worker* is currently using the *Tool* in the accomplishment of a *Job*. The association is a one-to-many relationship from *Worker* to *Tool*. This association has no attributes. This association is useful if a user wishes to know the *Tools* a *Worker* is currently using or the *Worker* who is currently using a certain *Tool*.

3.4.3.2 Qualified_On Association. The *qualified_on* association links a *Tool* to a *Worker* and shows what *Tools* a *Worker* is qualified to use in the accomplishment of a *Job*. The association is a many-to-many relationship from *Tool* to *Worker*. This association has no attributes. This association is useful if a user wishes to know which *Tools* a particular *Worker* is qualified to use or which *Workers* are qualified to use a certain *Tool*. For a *Worker-Tool* pair to be in the set of *qualified_on* associations, the afsc of the *Worker* must be a member of the afsc_set of the *Tool*.

3.4.3.3 Assigned Association. The *assigned* association links a *Tool* to a *Job* and shows the *Tool* that has been selected to aid in the accomplishment of a *Job*. The *assigned* association is a one-to-many relationship from *Tool* to *Job*. This association is useful if a user wishes to know the *Tool* that has been assigned to assist in the accomplishment of a particular *Job*, or the *Jobs* to which a particular *Tool* has been assigned.

3.4.3.4 Supports Association. The *supports* association links a *Tool* to a *Job* and shows that a *Tool* may support the accomplishment of a *Job*. This association is a many-to-many relationship between *Tool* and *Job* with two attributes, *time* and *percentage*. This association has the following attributes:

- time: indicates the amount of time saved when a qualified *Worker* uses the *Tool* to accomplish the *Job*

- percentage: indicates the increased accuracy amount when a qualified *Worker* uses the *Tool* to accomplish the *Job*. *Tool* percentage values range from 0, meaning the *Tool* provides no support for the *Job*, to 100, meaning the *Tool* can accomplish the *Job* in an automated mode without the need for a *Worker* assignment.

The *supports* association describes how well a *Tool* supports a particular *Worker* in accomplishing a *Job*. The link attributes *percentage* and *time* are used in the following manner. If a *Job* is to write a book, a *Worker* might use a pen, typewriter, or word processor. All of the *Tools* might be optional; however, the pen may have a *supports* level percentage of 10 while the typewriter might have a *supports* level percentage of 40 and the word processor might have a *supports* level percentage of 75. If this *Tool* is assigned to be used on a *Job*, then the *Job's* time and accuracy attributes are adjusted accordingly as described in Section 3.4.2.8.

3.4.3.5 Assignment Association. The *assignment* association links a *Job* to a *Worker* and shows that a *Job* to *Worker* assignment has been made. The *assignment* association is a one-to-many relationship from *Worker* to *Job*. This association is useful if a user wishes to find the *Worker* assigned to a given *Job*, as well as to find the *Jobs* assigned to a given *Worker*. The *assignment* association shows that *Worker* objects can be part of multiple *Worker-Job* assignments. Although there are no attributes attached to the *assignment* association, the attributes of the *Worker* and the *Job* do impact the time and accuracy attributes of the *Job* object. These attributes (time and accuracy) must be calculated based on each particular *Worker-Job* and/or *Tool-Job assignment* pairing(s). These attributes indicate the following about a *Job* object:

1. time - A calculation of how long it should take to complete a *Job*.
2. accuracy - A calculation of how error-free the output(s) of a *Job* should be.

In the original object model, the time and accuracy attributes had been attached to the actual *assignment* association between a *Worker* and a *Job*. This situation did not allow a *Job* to have

a time and accuracy measure without the assignment of a *Worker*. To prevent this situation, the attributes were moved to the *Job*, thereby allowing a *Job* to be accomplished without the assignment of a *Worker*. This was necessary to allow the ability of an automated *Tool* object to be assigned to accomplish a *Job* and still have the needed time and accuracy measures for the *Job*.

It is necessary to calculate the actual time required to perform the *Job* with the assignment of a particular *Worker*. The average time to complete a *Job*, the time it would take the average qualified worker to complete the *Job*, is available as an attribute of *Job*, as well as the best and worst time to complete a *Job*. Also available is the automated time, the time it would take to accomplish the *Job* if it is accomplished by an automated *Tool*. The actual time and accuracy are calculated based on the following criteria. If the assigned *Worker* does not possess an afsc which is in the *Job's* afsc_set then the assignment is not possible. If the *Worker's* skill level is lower than the skill level desired by the *Job*, the actual time will be greater than the expected time. If the person selected has an afsc in the *Job's* afsc_set and a higher skill level than the *Job* desires, the *Job's* actual time will be less than the expected time. If a *Tool* is assigned to assist in the *Job*, the *Job's* actual time may be less than the expected time, based on the time savings the *Tool* provides. The actual calculations for determining the time to perform the *Job* should be defined based on domain specific expected time values from research in this area, as well as information furnished by domain experts.

The accuracy of the *Job* as performed by the selected *Worker* is also calculated. Accuracy is a measure of the correctness of the output of a *Job*. Calculating the accuracy is similar to calculating the time of a *Job*. If the person assigned to a task is fully qualified, that is, the person has an afsc which is in the *Job's* afsc_set, the accuracy will be greater than or equal to the expected accuracy.

However, if the *Worker's* afsc is not in the *Job's* afsc_set, once again the assignment is not possible. The skill level of the *Worker* again affects the accuracy in the same manner as it affects time. The use of *Tools* to accomplish a *Job* can also increase the accuracy figure. Each *Tool* which supports a *Job* has associated with it a time attribute which indicates the amount of time a qualified *Worker* will expect to save by using the *Tool* to accomplish the *Job* as well as the increase in accuracy a qualified *Worker* will expect by using the *Tool* to accomplish the *Job*.

3.4.3.6 Qualified_For Association. The *qualified_for* association links a *Worker* to a *Job* and indicates that a particular *Worker* has the attributes necessary to accomplish a certain *Job*. This association is a many-to-many relationship between *Worker* and *Job* with two attributes named expect_accuracy and expect_time. The attributes are defined as follows:

- expect_accuracy: indicates the expected accuracy of a *Job* to *Worker* assignment in the accomplishment of a *Job*. This value is to be obtained from a data base of domain specific information concerning *Job* accomplishment by *Workers*.
- expect_time: indicates the expected time of a *Job* to *Worker* assignment in the accomplishment of a *Job*. This value is to be obtained from a data base of domain specific information concerning *Job* accomplishment by *Workers*.

3.4.3.7 Occupy Association. The *occupy* association is a one-to-one relationship between *Worker* objects and *Position* Objects. This association is needed to compute the *Workforce* object's force_readiness attribute. By comparing each *Position* in the overall *Organization's ManningPlan* object to the *Worker* object that is actually occupying that *Position*, a force readiness factor can be calculated. The force_readiness is a measure of the percentage of the *Positions* in the *Organization* which are filled by a qualified *Worker*. This is determined by comparing the afsc and the skill level of the *Worker* to the *Position* the *Worker* occupies.

3.4.3.8 Precedes Association. The *precedes* association is a many-to-many relationship between *Job* objects. The *precedes* association links *Job* objects to other *Job* objects and shows which *Jobs* must be accomplished before a particular *Job* can begin. This association has no attributes. This association is useful if a user wishes to determine the predecessors of a given *Job* or the *Jobs* which it precedes. This association is dependent on (derived from) the inputs and outputs of the *Job* objects.

3.4.4 Dynamic Model. The previous research effort did not include a dynamic model. However, the new model with its added objects and associations does need a dynamic model to describe the state, events, and actions of the lowest level objects in the model. These objects include the *Job*, the *Tool*, and the *Worker*. These objects are the only ones whose states must be described to fully define the dynamic behavior of the system, and are described in the following sections.

3.4.4.1 Job Dynamic Model. In the dynamic model, the *Job* object must be able to transition between its allowable states. These transitions occur based on certain guard conditions and system events. Figure 9 displays the dynamic model for the *Job* object.

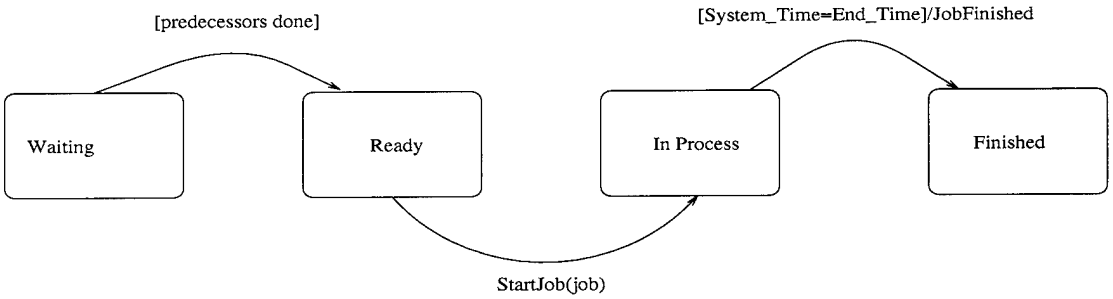

Figure 9. *Job* Object Dynamic Model

3.4.4.2 Tool Dynamic Model. The dynamic model for the *Tool* object is very simple since it involves only two possible states and responds to only two system events. A *Tool* transitions from the *available* status to the *in use* status upon receipt of the StartTool event and back to *available* upon receipt of the StopTool event. This is seen in Figure 10.

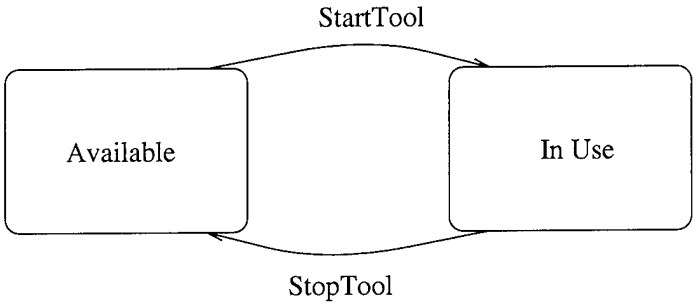

Figure 10. *Tool* Object Dynamic Model

3.4.4.3 Worker Dynamic Model. The *Worker* object dynamic model is the most complex of the three. The events and guard conditions it responds to and the actions it initiates are shown in the illustration in Figure 11.

3.4.4.4 Event Flow. A part of the dynamic model which is used to describe event communication between objects in the dynamic model is the event flow diagram (EFD). In the EFD for this model there is a scheduler which is used to issue StartJob and receive JobDone events. It was decided to evaluate whether it was necessary to have this scheduler or if the model could function properly without it. The dynamic model was modified to explore the possibility of eliminating the scheduler. This resulted in some conditions which require objects to know information about other objects. It was decided that this condition should be avoided if at all possible. Therefore, the scheduler remains part of the EFD and the system as depicted in Figure 12. This provides for

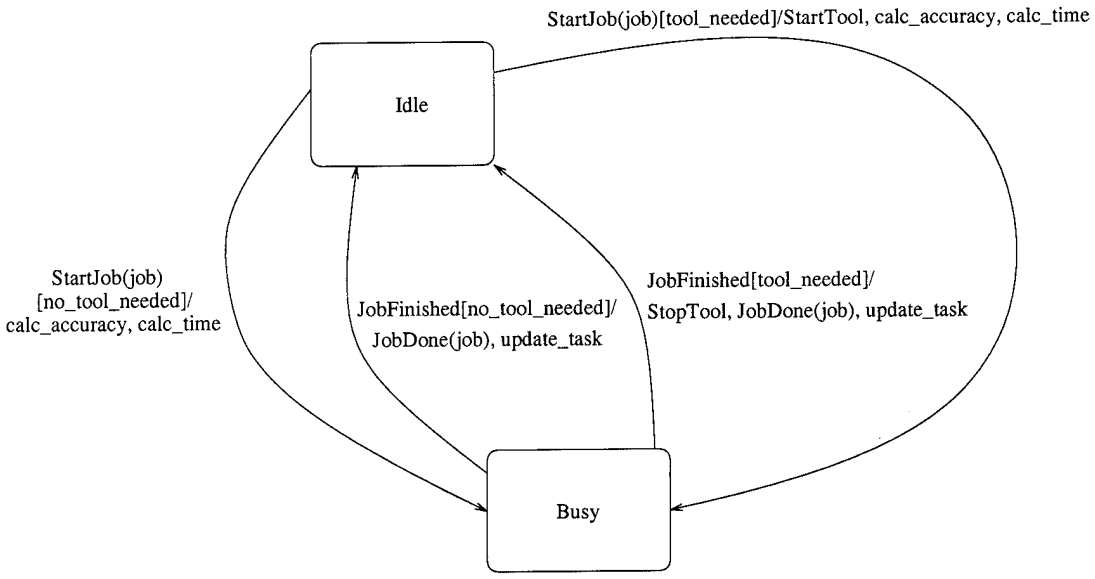

Figure 11. *Worker* Object Dynamic Model

better information hiding. The including of a scheduler follows the Job Shop approach as discussed in 2.6.3.

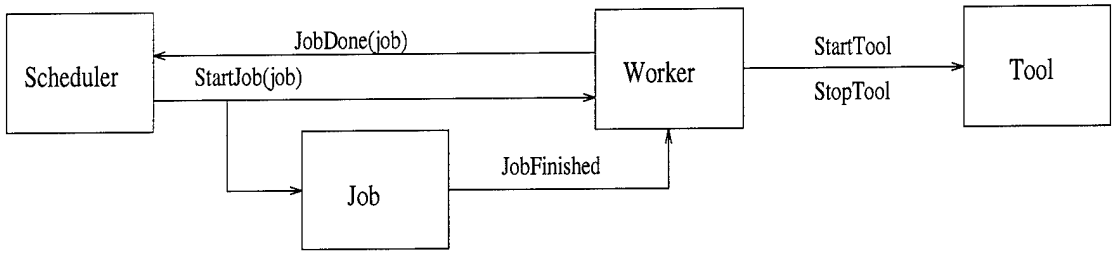

Figure 12. System Event Flow Diagram

3.4.5 Functional Model. Development of the functional model is necessary to capture the functional aspects of the domain model. The actions which must occur have to be described. This area is focused on the *Job*, *Worker*, and *Tool* objects and the attribute values which must be calculated for them and their associations. The functional model used for this research is based on

the one supplied by previous research (11). It was not modified, but only portions of it are actually included in the Z and Refine models.

3.5 AF Wing Z Model

3.5.1 Z Base Sets. In the process of implementing Z schemas it is necessary to define several basic sets for use within the individual object and association definitions. These sets are basically type definitions for domain specific types of data or information not already available through Z. The additional types are:

- **JOB_STATUS**: the set of all job status types
- **DATA**: the set of all input and output data types
- **AFSCS**: the set of all AFSCS

The JOB_STATUS set is defined as {waiting, ready, in_process, or finished} and is used to indicate not only the status of a *Job*, but also the status of *Tasks* and *Projects*. This type of information regarding the status of *Jobs*, *Tasks*, and *Projects* is fairly common and could be used in most domains. The DATA type set used will be more domain specific.

The data type set used for this research was generic numbered input and output symbols and is used for the input and output data for *Jobs* and *Tasks*. The level of domain knowledge required to fully define the exact inputs and outputs for each *Job* was not readily available but was not really necessary to achieve the goals of this research.

The AFSCS set is used to hold all of the Air Force Specialty Codes relevant to the organization being modeled. They are used for the definition of the afsc_set attribute of the *Tool* objects and *Job* objects as well as for the afsc attributes of the *Position* objects and *Worker* objects. These

three base sets were all that were required for the AF organization domain. This area will need to be examined for each particular domain during the model development process.

3.5.2 Worker Object. The *Worker* object is defined by the following Z static schema:

$$
\begin{array}{|l}
\hline
\textit{Worker} \\
\textit{name} : \mathrm{seq}\, CHAR \\
afsc : AFSCS \\
\textit{skill\_level} : \mathcal{N} \\
\textit{availability} : BOOLEAN \\
\textit{experience} : \mathcal{N} \\
\textit{education} : \mathcal{N} \\
\textit{aptitude} : \mathcal{N} \\
\hline
\textit{skill\_level} \geq 1 \\
\textit{skill\_level} \leq 9 \\
\textit{experience} \geq 0 \\
\textit{experience} \leq 30 \\
\textit{education} \geq 0 \\
\textit{education} \leq 18 \\
\textit{aptitude} \geq 0 \\
\textit{aptitude} \leq 100 \\
\hline
\end{array}
$$

3.5.3 Workforce Object. The force_readiness attribute of the *Workforce* is defined as all of the correctly occupied *Positions* in the *Organization.* The correctly occupied *Positions* are the members of the *occupy* association in which the *Worker's* afsc is equal to the *Position's* afsc and the *Worker's* skill level is greater than or equal to the *Position's* skill level. The *Workforce* object is defined by the following Z static schema:

$$
\begin{array}{|l}
\hline
Workforce \\
workerset : P\ Worker \\
force\_readiness : R \\
\hline
force\_readiness \geq 0 \\
force\_readiness \leq 100 \\
force\_readiness = \\
\quad \#\{\forall (p, w) : Occupy \bullet p.afsc = w.afsc \wedge w.skill\_level \geq p.skill\_level\}/\#(Workforce) \\
\hline
\end{array}
$$

3.5.3.1 Tool Object. The *Tool* object is defined by the following Z static schema:

$$
\begin{array}{|l}
\hline
Tool \\
name : \mathrm{seq}\, CHAR \\
status : symbol \\
afsc\_set : P\, AFSCS \\
skill\_level : \mathcal{N} \\
availability : BOOLEAN \\
\hline
skill\_level \geq 1 \\
skill\_level \leq 9 \\
\hline
\end{array}
$$

3.5.4 ToolSet Object. The tool_readiness attribute of the *ToolSet* is defined as the number of *Tool* objects in the *Organization's ToolSet* with a status equal to *go* divided by the number of *Tool* object in the *ToolSet*. The *ToolSet* object is defined by the following Z static schema:

$$
\begin{array}{|l}
\hline
ToolSet \\
toolset : P\ Tool \\
tool\_readiness : \mathrm{R} \\
\hline
tool\_readiness \geq 0 \\
tool\_readiness \leq 100 \\
tool\_readiness = \#\{t : Tool \bullet t \in toolset \wedge t.status = `GO\}/\#(toolset) \\
\hline
\end{array}
$$

3.5.5 Position Object. The *Position* object is defined by the following Z static schema:

Position

$number : \mathcal{N}$
$afsc : AFSCS$
$skill\_level : \mathcal{N}$

$skill\_level \geq 1$
$skill\_level \leq 9$

3.5.6 ManningPlan Object. The *ManningPlan* object is defined by the following Z static schema:

ManningPlan

$manningplan : P\, Position$

3.5.7 Job Object. The *Job* object is defined by the following Z static schema:

$$
\begin{array}{|l}
\textit{Job} \\
\hline
name : \mathrm{seq}\, CHAR \\
priority : \mathcal{N} \\
afsc\_set : P\ AFSCS \\
skill\_level : \mathcal{N} \\
status : JOB\_STATUS \\
complexity : \mathcal{N} \\
inputs : P\ DATA \\
outputs : P\ DATA \\
average\_time : \mathcal{N} \\
best\_time : \mathcal{N} \\
worst\_time : \mathcal{N} \\
automated\_time : \mathcal{N} \\
average\_accuracy : \mathcal{N} \\
best\_accuracy : \mathcal{N} \\
worst\_accuracy : \mathcal{N} \\
percent\_complete : \mathcal{N} \\
start\_time : \mathcal{N} \\
end\_time : \mathcal{N} \\
accuracy : \mathcal{N} \\
time : \mathcal{N} \\
\hline
priority \geq 0 \\
priority \leq 100 \\
skill\_level \geq 1 \\
skill\_level \leq 9 \\
complexity \geq 0 \\
complexity \leq 100 \\
percent\_complete \geq 0 \\
percent\_complete \leq 100 \\
end\_time > start\_time \\
time = start\_time - end\_time \\
automated\_time \leq best\_time \leq average\_time \leq worst\_time \\
(percent\_complete = 1.0) \rightarrow (status = finished) \\
(percent\_complete > 0.0 \wedge percent\_complete < 1.0) \rightarrow (status = in\_process) \\
inputs \cap outputs = \{\ \} \\
(time \geq automated\_time) \\
(time \leq worst\_time) \\
\hline
\end{array}
$$

In the Z static schema for the *Job* object, part of the behavior of the *Job* is based on the status and percent_complete attributes. When a *Job's* percent_complete is equal to 1.0 this implies that the *Job's* status must be equal to *finished.* Furthermore, if the *percent_complete* attribute of a

Job is less than 1.0 and greater than 0.0, then the *Job's* status must be *in_process*. These conditions are necessary for the behavior of the *Job* to be consistent.

To prevent a deadlock situation where a *Job* needs as one of its inputs a data item that it supplies as one of its outputs, the input and output data sets must not have a member in common. This is prevented with the following predicate.

$inputs \cap outputs = \{\ \}$

In the model, the *Job* attributes *average_time*, *best_time*, *worst_time*, and *automated_time* are related to each other. Obviously, the *worst_time* should be longer than, or at best the same as, the *average_time*. The *average_time* has a similar relationship with the *best_time*, and the *automated_time* will undoubtably always be the shortest. The *time* attribute is the actual time needed to complete the *Job* and must always be at least as much as the *automated_time* and never greater than the *worst_time*. These relationships are captured in the following predicates.

$automated\_time \leq best\_time \leq average\_time \leq worst\_time$

$(time \geq automated\_time)$

$(time \leq worst\_time)$

3.5.8 Task Object. The *Task* object is defined by the following *Z* static schema:

Task

$name : \text{seq } CHAR$
$priority : \mathcal{N}$
$status : JOB\_STATUS$
$inputs : DATA$
$outputs : DATA$
$time\_so\_far : \mathcal{N}$
$percent\_complete : R$
$number\_of\_jobs : \mathcal{N}$
$jobs\_complete : \mathcal{N}$
$start\_time : \mathcal{N}$
$end\_time : \mathcal{N}$
$wall\_time : \mathcal{N}$
$total\_time : R$
$accuracy : \mathcal{N}$
$job\_set : P\, JOBS$

$priority \geq 0$
$priority \leq 100$
$percent\_complete \geq 0$
$percent\_complete \leq 100$
$start\_time = \min_{\forall j \in job\_set} (j.start\_time)$

$end\_time = \max_{\forall j \in job\_set} (j.end\_time)$

$total\_time = \sum_{\forall j \in job\_set} j.time$

$wall\_time = start\_time - end\_time$

$wall\_time \leq total\_time$

$jobs\_complete = \#\{j : job \mid j \in job\_set \wedge j.status = finished\}$

$number\_of\_jobs = \#(job\_set)$

$jobs\_complete \leq number\_of\_jobs$

$(\exists j : job \bullet j \in job\_set \wedge j.status = inprocess) \rightarrow$
$time\_so\_far = j.start\_time + j.time * j.percent\_complete$

$percent\_complete = \sum_{\forall j \in job\_set \bullet j.status \,=\, finished} j.time / \sum_{\forall j \in job\_set} j.time$

Task(Continued)

$\forall j \bullet j \in job\_set \wedge j.status = finished \rightarrow status = finished$

$\forall j \in job\_set \bullet (j.status = waiting) \rightarrow status = (waiting)$

$\forall j \in job\_set \bullet (j.status = ready) \rightarrow status = (ready)$

$\exists j \in job\_set \bullet j.status = in\_process \rightarrow status = in\_process$

$(jobs\_complete = number\_of\_jobs) \rightarrow (percent\_complete = 100) \wedge (status = finished)$

$\{\forall j_1 \in job\_set \bullet j_1.inputs \notin [\{\forall j_2 : job\_set \bullet j_2.inputs\} \cap \{\forall j_3 : job\_set \bullet j_3.outputs\}]\} \rightarrow j.inputs \in inputs$

$\{\forall j_1 \in job\_set \bullet j_1.outputs \notin [\{\forall j_2 : job\_set \bullet j_2.inputs\} \cap \{\forall j_3 : job\_set \bullet j_3.outputs\}]\} \rightarrow j.outputs \in outputs$

$inputs \cap outputs = \{\ \}$

$$accuracy = \sum_{\forall j \in job\_set} j.accuracy/\#(job\_set)$$

In the *Task* object, the start and end times of the *Task* are dependent upon the start and end times of the *Jobs* in the job_set of the *Task*. This is captured in the following predicates.

$$start\_time = \min_{\forall j \in job\_set} (j.start\_time)$$

$$end\_time = \max_{\forall j \in job\_set} (j.end\_time)$$

The total time spent on the *Task* is simply the sum of the times of all the *Jobs* in the *job_set* and the *wall_time* of the *Task* is simply the *end_time* minus the *start_time* of the *Task*. Since the *wall_time* takes into account the concurrency of *Jobs* and the *total_time* does not, the *wall_time* can never be greater than the *total_time*. This behavior is captured in the following predicates.

$$total\_time = \sum_{\forall j \in job\_set} j.time$$
$$wall\_time = start\_time - end\_time$$
$$wall\_time \leq total\_time$$

To express the *time_so_far* attribute of the *Task* at any point in time requires knowing the maximum *end_time* of all the *Jobs* in the *job_set* which have been completed. Also required is the *Job* which has consumed the most time and is currently in the *in-process* status. These requirements are expressed in the following predicate.

$$(\exists j : job \bullet j \in job\_set \wedge j.status = inprocess) \rightarrow$$

$$time\_so\_far = j.start\_time + j.time * j.percent\_complete$$

The *percent_complete* attribute of the *Task* can be approached in two manners. First, it could be based solely on the percentage of *Jobs* in the *job_set* which have been completed. However, this does not take into account the event that the last *Job*, or any *Job(s)*, could be extremely time intensive. The second approach is to base the *percent_complete* on the amount of time of completed *Jobs* versus the total time to complete all *Jobs*. The second approach was decided to be more realistic and chosen for implementation. The predicates for both approaches respectively are as follows.

$$percent\_complete = \frac{\#\{j : job\_set \bullet j.status = finished\}}{\#(job\_set)}$$

$$percent\_complete = \sum_{\forall j \in job\_set \bullet j.status = finished} j.time / \sum_{\forall j \in job\_set} j.time$$

The status of the *Task* is dependent on the status of the *Jobs* which make up the *Task's* job_set. In order for the *Task* to be finished, all of the members of the job_set must be finished. Also, if all of the *Jobs* in the job_set are in waiting status or ready status then the *Task* must also be in the same respective status. Finally, once all of the *Jobs* in the job_set are complete,

jobs_complete will be equal to number_of_jobs. This implies that the *Task* must be finished and the percent_complete will be equal to 1.0. These behaviors are captured in the following predicates.

$$\exists j \in job\_set \bullet j.status \neq finished \rightarrow status \neq finished$$

$$\forall j \in job\_set \bullet (j.status = waiting) \rightarrow status = (waiting)$$

$$\forall j \in job\_set \bullet (j.status = ready) \rightarrow status = (ready)$$

$$\exists j \in job\_set \bullet j.status = in\_process \rightarrow status = in\_process$$

$$(jobs\_complete = number\_of\_jobs) \rightarrow (percent\_complete = 100) \wedge (status = finished)$$

To derive the inputs to a *Task* it is necessary to specify the inputs to the *Jobs* in the job_set which are not outputs from another *Job* within the job_set. The intersection between the inputs and outputs of the *Jobs* in the job_set are internal to the *Task*. Only the inputs and outputs not in the intersection can be inputs and outputs which cross the boundaries of the *Task* and are therefore members of the *Task's* inputs and outputs sets respectively. The following predicates specify this behavior.

$$\{\forall j_1 \in job\_set \bullet j_1.inputs \notin [\{\forall j_2 : job\_set \bullet j_2.inputs\} \cap \{\forall j_3 : job\_set \bullet j_3.outputs\}]\} \rightarrow$$
$$j.inputs \in inputs$$

$$\{\forall j_1 \in job\_set \bullet j_1.outputs \notin [\{\forall j_2 : job\_set \bullet j_2.inputs\} \cap \{\forall j_3 : job\_set \bullet j_3.outputs\}]\} \rightarrow$$
$$j.outputs \in outputs$$

3.5.9 Project Object. The *Project* object is defined by the following Z static schema:

Project

$name : \text{seq}\, CHAR$
$priority : \mathcal{N}$
$status : JOB\_STATUS$
$time\_so\_far : R$
$number\_of\_tasks : \mathcal{N}$
$tasks\_complete : \mathcal{N}$
$percent\_complete : R$
$number\_of\_jobs : \mathcal{N}$
$jobs\_complete : \mathcal{N}$
$start\_time : \mathcal{N}$
$end\_time : \mathcal{N}$
$wall\_time : \mathcal{N}$
$total\_time : R$
$accuracy : \mathcal{N}$
$task\_set : P\, TASKS$
$job\_set : P\, JOBS$

$priority \geq 0$
$priority \leq 100$
$percent\_complete \geq 0$
$percent\_complete \leq 100$
$job\_set = \bigcup_{\forall t \in task\_set} t.job\_set$

$\forall j : Job \bullet j \in ran(precedes) \Rightarrow$
$(\forall (k, j) : Precedes \bullet (k, j) \in precedes \wedge\ j.start\_time \geq\ max(k.end\_time))$

$start\_time = \min_{\forall t \in task\_set} (t.start\_time)$

$end\_time = \max_{\forall t \in task\_set} (t.end\_time)$

$total\_time = \sum_{\forall t \in task\_set} t.time$

$wall\_time = start\_time - end\_time$

$wall\_time \leq total\_time$

$tasks\_complete = \#\{t : task\_set \bullet t.status = finished\}$

$number\_of\_tasks = \#(task\_set)$

$tasks\_complete \leq number\_of\_tasks$

Project(Continued)

$$time\_so\_far = \max_{\forall t \in task\_set \bullet t.status\ =\ finished} (t.end\_time) + \max_{\forall t \in task\_set \bullet t.status\ =\ in\_process} (t.time * t.percent\_complete)$$

$$(\exists j : job \bullet j in job\_set \wedge j.status = in - process) \rightarrow time\_so\_far = j.start\_time + j.time * j.percent\_complete$$

$$percent\_complete = \sum_{\forall t \in task\_set \bullet t.status\ =\ finished} t.time / \sum_{\forall t \in task\_set} t.time$$

As can be seen from the predicates, the attributes of the *Project* object are very similar to the attributes of the *Task* object. However, in the *Project* attribute section there is a redundant job_set. This is the set of all *Jobs* which are in the *Tasks* within the *Project's task_set.* Since it was necessary to access some of the attributes of individual *Jobs* in deriving several of the attributes of the *Project*, it was necessary to have the *job_set* declared and derived at the *Project* level.

3.5.10 Organization Object. The *Organization* object is defined by the following Z static schema:

Organization

$Assigned$
$Using$
$Qualified\_On$
$Assigned$
$Assignment$
$Qualified\_For$
$Supports$
$Occupy$
$Precedes$
$toolset : P\ Tools$
$workforce : P\ Workers$
$projectset : P\ Projects$
$manningplan : P\ Positions$
$unit\_efficiency : \mathcal{N}$
$unit\_effectiveness : \mathcal{N}$
$unit\_readiness : R$

$unit\_efficiency \geq 0$
$unit\_efficiency \leq 100$
$unit\_effectiveness \geq 0$
$unit\_effectiveness \leq 100$
$unit\_readiness \geq 0.0$
$unit\_readiness \leq 1.0$

$[(\forall (w_1, j_1) \bullet (w_1, j_1) \in assignment) \rightarrow ((w_1, j_1) \in qualified\_for)]$

$[(\forall (t, j) \bullet (t, j) \in assigned) \rightarrow ((t, j) \in supports)$

$[(\forall (w, t) \bullet (w, t) \in using) \rightarrow ((w, t) \in qualified\_on)]$

$[(\forall (w, t) \bullet (w, t) \in using) \rightarrow (\exists j \bullet ((w, j) \in assignment) \wedge ((t, j) \in assigned)]$

$[(\forall (w_1, j_1) \bullet (w_1, j_1) \in assignment) \wedge (\forall t \bullet (t, j_1) \in supports) \wedge ((t, j_1) \notin assigned)] \rightarrow$
$[((t, w_1) \notin using) \wedge ((w_1, j_1) \in qualified\_for) \wedge$
$(j_1.accuracy = ((qualified\_for(w_1, j_1).expected\_accuracy) * normal\_distribution))]$

$[(\forall t_1, w_1 \bullet (t_1, w_1) \in using) \wedge (\exists j_1 \bullet (t_1, j_1) \in assigned) \wedge (w_1, j_1) \in assignment)] \rightarrow$
$j_1.accuracy = qualified\_for(w_1, j_1).expect\_accuracy + supports(t_1, j_1).percentage$

$(t, j) \in supports \rightarrow supports(t, j).percentage \leq 100$

Organization(continued)

$(w,j) \in qualified\_for \rightarrow qualified_for(w,j).expect\_accuracy \leq 100$

$(\neg\ \exists t_1, w_1 \bullet (t_1, w_1) \in using \wedge (w_1, j_1) \in assignment)) \rightarrow$
$j_1.time = qualified\_for(t_1, w_1).expect\_time * normal\_distribution$

$[(\exists t_1, w_1 \bullet (t_1, w_1) \in using) \wedge (\exists j_1 \bullet (j_1, w_1) \in assignment) \wedge$
$(t_1, j_1) \in supports \wedge (t_1, j_1) \in assigned)] \rightarrow$
$j_1.time = (qualified\_for(w_1, j_1).expect\_time * normal\_distribution) - supports(t_1, j_1).time$

$\forall (w_1, j_1) \in qualified\_for \bullet (w_1.afsc \in j_1.afsc\_set) \wedge$
$(w_1.skill\_level > j_1.skill\_level) \rightarrow qualified\_for(w_1, j_1).expect\_time = j_1.best\_time$

$\forall (w_1, j_1) \in qualified\_for \bullet (w_1.afsc \in j_1.afsc\_set) \wedge$
$(w_1.skill\_level = j_1.skill\_level) \rightarrow qualified\_for(w_1, j_1).expect\_time = j_1.average\_time$

$\forall (w,j) \bullet (w,j) \in qualified\_for \rightarrow qualified\_for(w,j).expect\_time \geq 0$

$\forall (w,j) \bullet (w,j) \in qualified\_for \rightarrow qualified\_for(w,j).expect\_accuracy \geq 0$

$\forall (w,j) \bullet (w,j) \in qualified\_for \rightarrow qualified\_for(w,j).expect\_accuracy \leq 100$

$\forall (t,j) \bullet (t,j) \in supports \rightarrow supports(t,j).time \geq 0$

$\forall (w_1, j_1) \bullet (w_1, j_1) \in qualified\_for \wedge (w_1.afsc \in j_1.afsc\_set) \wedge$
$(w_1.skill\_level < j_1.skill\_level) \rightarrow qualified\_for(w_1, j_1).expect\_time = j_1.worst\_time$

$\forall (w_1, j_1) \bullet (w_1, j_1) \in qualified\_for \wedge (w_1.afsc \in j_1.afsc\_set) \wedge$
$(w_1.skill\_level > j_1.skill\_level) \rightarrow qualified\_for(w_1, j_1).expect\_accuracy = j_1.best\_time$

$\forall (w_1, j_1) \bullet (w_1, j_1) \in qualified\_for \wedge (w_1.afsc \in j_1.afsc\_set) \wedge$
$(w_1.skill\_level = j_1.skill\_level) \rightarrow qualified\_for.expect\_accuracy = j_1.average\_time$

$\forall (w_1, j_1) \bullet (w_1, j_1) \in qualified\_for \wedge (w_1.afsc \in j_1.afsc\_set) \wedge$
$(w_1.skill\_level < j_1.skill\_level) \rightarrow qualified\_for(w_1, j_1).expect\_accuracy = j_1.worst\_accuracy$

$\forall (w_1, j_1) \bullet (w_1, j_1) \in qualified\_for \wedge (w_1.afsc \in j_1.afsc\_set) \wedge$
$(w_1.skill\_level = j_1.skill\_level) \rightarrow qualified\_for(w_1, j_1).expect\_accuracy = j_1.average\_accuracy$

$\forall (w_1, j_1) \bullet (w_1, j_1) \in qualified\_for \wedge (w_1.afsc \in j_1.afsc\_set) \wedge$
$(w_1.skill\_level > j_1.skill\_level) \rightarrow qualified\_for(w_1, j_1).expect\_accuracy = j_1.best\_accuracy$

$\forall (w_1, j_1) \bullet (w_1, j_1) \in qualified\_for \wedge (w_1.afsc \in j_1.afsc\_set) \wedge$
$(w_1.experience \geq 10) \wedge (j_1.complexity \leq 75) \rightarrow$
$qualified\_for(w_1, j_1).expect\_accuracy = qualified\_for(w_1, j_1).expect\_accuracy + 20$

$$\underline{\quad Organization(Continued) \quad}$$

$$\forall(w_1, j_1) \bullet (w_1, j_1) \in qualified\_for \wedge (w_1.afsc \in j_1.afsc\_set) \wedge$$
$$(w_1.education \geq 12) \wedge (j_1.complexity \leq 50) \rightarrow$$
$$qualified\_for(w_1, j_1).expect\_accuracy = qualified\_for(w_1, j_1).expect\_accuracy + 10$$

$$toolset.tool\_readiness = \#\{\forall t \bullet t \in toolset \wedge tool.status = go\}/\#(toolset)$$

$$workforce.force\_readiness =$$
$$\#\{\forall(p, w) \bullet (p, w) \in occupy \wedge (w.afsc = p.afsc) \wedge (w.skill\_level \geq p.skill\_level)\}/\#(manningplan)$$

$$unit\_readiness = toolset.tool\_readiness + workforce.force\_readiness$$

Most of the predicates for the *Organization* deal with the associations of the model. Some of the more interesting associations are discussed in the following sections.

3.5.11 Occupy Association. The Occupy association is defined with the following Z static schema:

$$\underline{\quad Occupy \quad}$$

$$occupy : Worker \rightarrowtail\!\!\!\!\!\!+\ Position$$

3.5.12 Using Association. For any *Worker-Tool* pair to be a member of the *using* association, there is the implication that there is a *Job* such that it is in an *assigned* association pair and that the *Worker* is qualified on the *Tool* and that particular *Tool* supports the *Job* and that the *Worker* has been assigned to the *Job* and that the *Worker* is qualified for the *Job*. The *Using* association is defined with the following Z static schema:

$$\underline{\quad Using \quad}$$

$$using : Tool \nrightarrow Worker$$

$$\forall(t, w) \in using \rightarrow [\exists j : JOB \bullet (t, j) \in assigned \wedge (t, w) \in qualified\_on$$
$$(t, j) \in supports \wedge (w, j) \in assignment \wedge (w, j) \in qualified\_for]$$

3.5.13 Qualified_On Association. To be a member of the *qualified_on* association, the *Worker's* afsc must be in the *Tool's* afsc set and the *Worker's* skill level must be greater than or equal to the *Tools* skill level. The Qualified_On association is defined with the following Z static schema:

$$
\begin{array}{|l}
\hline
Qualified\_on \\
qualified\_on : Tool \leftrightarrow Worker \\
\hline
\forall (t, w) \in qualified\_on \rightarrow (w.afsc \in t.afsc\_set) \wedge (w.skill\_level \geq t.skill\_level) \\
\hline
\end{array}
$$

3.5.14 Assigned Association. The conditions for being a member of the *assigned* association are very similar to the *using* association. The Assigned association is defined with the following Z static schema:

$$
\begin{array}{|l}
\hline
Assigned \\
assigned : Job \nrightarrow Tool \\
\hline
\forall (t, j) \in assigned \rightarrow [\exists w \bullet (t, w) \in using \wedge \\
(t, w) \in qualified\_on \wedge (w, j) \in qualified\_for \wedge \\
(w, j) \in assignment \wedge (t, j) \in supports] \\
\hline
\end{array}
$$

3.5.15 Supports Association. The Supports association is defined with the following Z static schema:

$$
\begin{array}{|l}
\hline
SupportsAttr \\
time : \mathcal{N} \\
percentage : \mathcal{N} \\
\hline
time \geq 0 \\
percentage \leq 100 \\
percentage \geq 0 \\
\hline
\end{array}
$$

Supports

$$supports : (Tool \times Job) \rightarrow SupportsAttr$$

$$\forall (t, j) \in \mathrm{dom}(supports) \rightarrow t.afsc \in j.afsc\_set$$

3.5.16 Qualified_For Association. The Qualified_For association is defined with the following Z static schema:

Qualified_ForAttr

$$expect\_time : \mathcal{N}$$
$$expect\_accuracy : \mathcal{N}$$

$$expect\_time \geq 0$$
$$expect\_accuracy \leq 100$$
$$expect\_accuracy \geq 0$$

Qualified_For

$$qualified\_for : (Worker \times Job) \rightarrow Qualified\_ForAttr$$

$$\forall (w, j) \in \mathrm{dom}(qualified\_for) \rightarrow w.afsc \in j.afsc\_set$$

3.5.17 Assignment Association. The *assignment* association is defined with the following Z static schema:

Assignment

$$assignment : Job \nrightarrow Worker$$

$$\forall (j, w) \in assignment \rightarrow w.afsc \in j.afsc\_set \wedge (j, w) \in qualified\_for$$

3.5.18 Precedes Association. All *Job* pairs in the *precedes* association must adhere to the following conditions: the outputs from the first member of the pair must intersect with the inputs of the second member and the end time of the first must be less than or equal to the start time of the first. The *precedes* association is defined with the following Z static schema:

Precedes

$precedes : Job \leftrightarrow Job$

$\forall (j_1, j_2) \in precedes \bullet j_1.outputs \cap j_2.inputs \neq \{\ \} \wedge$
$j_1.end\_time \leq j_2.start\_time$

$\forall (j_1, j_2) \in precedes \rightarrow \exists\, d_1 \in (j_1.outputs \wedge j_2.inputs)$

3.6 Automated Tasks

In the object model proposed by Hunt and Sarchet (12, 28), the attributes which describe the time and accuracy of a *Worker* performing a *Job* were associated with the *assignment* association. This imposed the limitation that for any *Job* to have a time or accuracy, there must be a *Worker* to *Job* assignment. It was desired to have the ability to model an automated *Job* as being accomplished without the need for a *Worker*. Therefore, it was decided to move the time and accuracy attributes to the *Job*. This allows time and accuracy attributes to be assigned independent of a *Worker* to *Job* assignment, thus allowing an automated *Tool* to perform the *Job*. Also added to the attributes associated with the *Job* object was the automated_time attribute. This attribute designates the actual amount of time required to do the *Job* if it is automated and is disregarded if the *Job* is done by a *Worker*.

If a *Job* is automated, a reasonable accuracy must be established. This accuracy may be *Job* specific. In most cases, however, the accuracy will most likely be very near 100 percent. Probability and reliability statistics will be needed to correctly assign accuracy values for each *Job*.

3.7 Efficiency and Effectiveness Metrics for the Organization

The efficiency and effectiveness of an organization can be interpreted in many different ways depending on the organization and the persons making the evaluation. An initial assessment by Hunt and Sarchet (12, 28) was made to determine what was needed to produce an overall assessment and to provide the necessary feedback to give the system user insight into how well allocated the resources were and whether or not automation provided any value. This assessment determined that the following broad categories of feedback measures were important to aid in the evaluation of an organization and determine how automation added value:

1. Timing
2. *Worker* Efficiency
3. Accuracy
4. Critical Path

The definitions for these metrics are in (12, 28). In the next section, the Z code to formally express and compute these metrics are presented.

3.7.1 Z Code for the Timing Metrics. Analysis of the timing measures yielded the following timing metrics along with the Z code to formally describe them.

1. Shortest *Job*

$$\min_{\forall j \in jobset} j.time$$

2. Longest *Job*

$$\max_{\forall j \in jobset} j.time$$

3. Shortest *Task*

$$\min_{\forall t \in taskset} t.time$$

4. Longest *Task*

$$\max_{\forall t \in taskset} t.time$$

5. Shortest *Project*

$$\min_{\forall p \in projectset} p.time$$

6. Longest Project

$$\max_{\forall p \in projectset} p.time$$

7. Total Time to Complete *Project*

$$\sum_{\forall t \in taskset} t.time$$

The approach to computing the Total Amount of Time for a *Project* is to accumulate the times for each individual *Task*. The Z code to calculate this follows:

$$\sum_{\forall t \in taskset} t.time$$

3.7.2 Z for Calculating Worker Efficiency. Analysis of the *Worker* Efficiency measures yielded the following efficiency-related metrics, each of which is described in further detail:

1. Time Spent Working for *Worker* w_1

$$\sum_{\forall (w_1, j) \in assignment} j.time$$

2. *Worker* Efficiency for *Worker* w_1

$$Time\ Spent\ Working(w_1)/Project.time$$

Time Spent Working is calculated for each *Worker* by accumulating the individual times spent performing *Jobs*. *Worker Efficiency* is the ratio between *Time Spent Working* and the amount of time required to complete the *Project*.

3.7.3 Z for Calculating the Accuracy Metrics. Analysis of the accuracy measures yielded the following accuracy-related metrics, each of which is described in further detail:

1. Least Accurate *Job*

$$\min_{\forall j \in jobset} j.accuracy$$

2. Most Accurate *Job*

$$\max_{\forall j \in jobset} j.accuracy$$

3. Least Accurate *Task*

$$\min_{\forall t \in taskset} t.accuracy$$

4. Most Accurate *Task*

$$\max_{\forall t \in taskset} t.accuracy$$

5. Overall Accuracy for *Project*

$$(\sum_{\forall t \in taskset} t.accuracy)/\#(taskset)$$

3.8 AF Wing Refine Model

The resulting Refine implementation of the model is included in Appendix A. Due to time constraints, the full *Z* specifications were not implemented. However, many of the key aspects of the model are implemented and display much versatility and functionality.

3.8.1 Objects in Refine. The mapping of the *Z* specifications into the Refine model produced the executable version of the model. The files for base objects in Refine were typical. The following is a partial example of the *Worker* file:

```
!! in-package("RU")
!! in-grammar('user)
```

```
#||
--------------------------------------------------------------------------
-- Object Name: Worker
-- Specification Date: 08/28/95
-- Filename: worker.re
-- Specified by: Hibdon
--------------------------------------------------------------------------
||#

%% Define base sets (types):
constant AFSCS : set(string) = { "33S3B", "33S3C", "4921"}

%% Define class structure:
var Worker: object-class subtype-of user-object
    var wname: map(Worker, string) = {||}
    var wafsc: map(Worker, string) = {||}
    var wskill_level: map(Worker, integer) = {||}
    var wavailability: map(Worker, boolean) = {||}
    var wexperience: map(Worker, integer) = {||}
    var weducation: map(Worker, integer) = {||}
    var waptitude: map(Worker, integer) = {||}

function ShowWorker(w: Worker) =
 format(TRUE, " Worker name: ~S~% Afsc: ~S~% skill_level: ~d~% Availability: ~b~%
   Experience: ~d~% Education: ~d~% Aptitude: ~d~%~%~%", wname(w), wafsc(w),
   wskill_level(w), wavailability(w), wexperience(w), weducation(w), waptitude(w))

function Setwname(w: Worker, n: string) =
  TRUE --> wname(w) = n

%% Checks for valid afsc
function Setwafsc(w: Worker, a: string) =
  a in AFSCS --> wafsc(w) = a;
  a ~in AFSCS --> wafsc(w) = undefined

%% Checks boundary ranges.
function Setwskill_level(w: Worker, s: integer) =
  (s >= 1 & s <= 9) --> wskill_level(w) = s

function Getwskill_level(w: Worker): integer =
  wskill_level(w)
```

3.8.2 Creating Aggregate Classes. Aggregate objects are declared in the same fashion as primitive objects, that is they are declared of type object-class and subtype of user-object. The *Organization* is an example of such an aggregate object. The *Organization* is composed of a *ToolSet*, a *Workforce*, a *ManningPlan*, and a *ProjectSet*. It also contains the mappings for all of the associations among all objects. The declaration for the *Organization* is in Appendix A. Notice that some of the variables of the *Organization* are mappings from the *Organization* to other already declared objects. This is the method for representing aggregation in Refine.

3.8.3 Testing Refine Model. The Refine implementation of the domain model was tested using a variety of tests. Initially, test drivers were written for the individual objects as they were developed. These tests mainly consisted of functions to initialize and modify the attributes of the individual objects. These tests evolved to include the range constraints of the attributes and finally checked the specific behavior of the objects. A sample test function for the *Worker* object follows:

```
!! in-package("RU")
!! in-grammar('user)
#||
------------------------------------------------------------------------
-- Object Name: Worker test routine
-- Specification Date: 08/28/95
-- Filename: testworker.re
-- Specified by: Hibdon
--
-- Based on: Worker (950505)
--
-- History:
-- 08/28/95 (Hibdon): original design.
--
-- Test program for Worker
------------------------------------------------------------------------
||#

var myworker: Worker = NewWorker()
```

```
function TestWorker1(w: Worker) =
  SetwName(w, "Joe Smith");
  SetwAfsc(w, "33S3B");
  Setwskill_level(w, 5);
  SetwAvailability(w, TRUE);
  SetwExperience(w, 10);
  SetwEducation(w, 15);
  SetwAptitude(w, 125);
  ShowWorker(w)

%%---------------- End of testworker ------------------------------------
```

After the completion of the *Organization* object and the transformation of much of the *Z* specifications, it was necessary to develop a test which checked the full functionality of the model. This was a very extensive step and required many additional functions be created to set up the system to provide the required members in the appropriate associations to allow the observation of the model. For example, to model an actual assignment of a *Worker* to a *Job* required the function *AddAssignment* which created the *Assignment* object and associated the specified *Worker* and *Task* as attributes of the object. The model as developed actually provides for a snapshot in time of the *Organization* and its parts. It does not actually have dynamic behavior. A complete set of associations and objects must be in place for the model to execute.

The test functions in the Refine environment were used to verify that the model behaved as expressed in the Rumbaugh and *Z* specifications. The results of all conducted tests confirmed that the Refine portions of the model behaved as expected.

3.9 Summary

This chapter has described the methodology for creating a formal model of an AF organization. The three main phases of the methodology, OMT, *Z* specification, and transformation to

Refine, were described in detail and the resulting model was presented. The domain model of the AF organization, consisting of the Rumbaugh OMT, the Z specification, and Refine implementation, served as the basis for analysis of the described methodology. Chapter IV provides an analysis of the methodology used to develop the model as well as analysis of the model itself.

IV. Analysis and Results

4.1 Introduction

The domain model of an Air Force organization which was the result of the described methodology is discussed in Chapter III. As discussed in Section 1.2, the sponsor needed decision making support for allocating resources and determining the effects of automation. To provide this support, the proposed formal methodology for developing an organization model was followed. The previous attempt at providing this support was developed using Ada 83 by Hunt and Sarchet. The Ada version required a procedural simulation to run for actual evaluation of the model. That version also included the inherent maintainability problems associated with any procedural language. All updates to the model require the maintainer to locate and change all dependencies related to any modification of the model. This is not only difficult but also time consuming. In the Refine model, the environment locates dependencies and makes the appropriate internal changes necessary for the new requirements defined in maintenance. This equates to redevelopment of a system through a changed set of system requirements. The aim of this analysis is to examine the overall methodology proposed for developing a formal model of an organization and to evaluate whether or not the goals of the model were actually met. This analysis focuses on the goals of the model and the contributing factors to the related successes and failures in achieving these goals.

4.2 Analysis of Model

The Rumbaugh model developed during this research is certainly more comprehensive than the previous model. The added objects and attributes make it more truly reflect the real-world AF wing. Additionally, the model is more versatile and maintainable. The objects and attributes

ensure that the model has a higher probability of reuse in different domains. This model may be able help leaders more accurately assess the status of their organizations. This model has many uses, from the assessment of the assignment of workers and tools to accomplish jobs to providing organizational metrics to leaders. The model has the ability to assess overall organization metrics such as unit readiness, unit effectiveness, and unit efficiency. If used in conjunction with an optimizing scheduling environment such as KIDS/SpecWare (5), the model could be enhanced to include evaluating assignment schedules. It could also then be used to assess the impact of automation tools on various assignment schedules. The model will also be useful in the assessment of various proposed schedules for worker, tool, and job assignments and their impact on the organization.

The improved model implemented in Refine is easier to modify for use by other organizations. This is due to the object-oriented approach taken in the design phase and the environment chosen for implementation. Not only is the model reusable, so is the methodology used to develop the model. If necessary, a system developer could use the proposed methodology from the beginning to design his or her own domain-specific organizational process model and finish with a set of executable specifications. Modification and maintenance of the model involves simply making the changes to the specifications. The changes could be possibly made at The Refine level and back documented into the Z and Rumbaugh model or carried forward from the Rumbaugh model to the Refine. An evolutionary or incremental approach to system development should be simple to implement with the availability of a functional prototype system to check behavior throughout the development when using this methodology.

An initial goal was to make the model more representative of the real-world through further domain analysis and improvements to the Rumbaugh model. Due to the past experience of the

researcher and the knowledge gained by the previous research of Hunt and Sarchet (12, 28), it was possible to make great strides in this area. The new object model contains increased numbers of objects and associations as well as more relevant object and association attributes. These include the added *Project* and *Task* objects which not only allow for the collection of intermediate level results but also allow the flexibility to overcome the previous limitations of assigning only one *Worker* or *Tool* to a *Job*. This is done by having the *Job* object be the smallest possible piece of work to accomplish. The *ToolSet*, *ManningPlan*, and *Workforce* objects were added to provide a more accurate method for determining the *Organization's* overall *unit_readiness*. The *Using* and *Qualified_On* associations were included to ensure that *Worker* to *Tool* relationships that were relevant to the assigning of *Tool-Worker-Job* combinations were done in a valid manner. This was to ensure that the proper *Worker* was using a *Tool* he or she is *Qualified_On*. Also, a similar relationship was modeled between the *Worker* and the *Job* objects through the *Qualified_For* and Assignment associations. Not only does this ensure that a qualified *Worker* is *Assigned* to the *Job*, it also has the additional information contained in the attributes of the *Qualified_For* association which provides specific information pertaining to the amount of time and degree of accuracy a particular *Worker* is expected to have in the performance of a particular *Job*. A very similar relationship can be seen in the *Supports* and *Assigned* associations between the *Tool* and *Job* objects. The *time* and *percentage* attributes of the *Supports* association indicate how much time can be saved on a particular *Job* by using a particular *Tool*. These additional objects and associations have helped achieve some of the goals of this research by improving the realism of the model.

The review of operations research literature exposed many of the areas which were improved. These included the approach to calculating overall unit readiness and the addition of an intermediate level work unit to allow for a *Project* object to be composed of *Task* objects which are further

composed of *Job* objects. The Ada model implementation is limited to assigning a single *Worker* to a *Task* and single *Tool* to a *Task*. In reality, several *Workers* may actually perform a *Task* in a given *Project* and several *Tools* may be used also. This is also accounted for through the intermediate level *Task* object which is composed of several *Job* objects which have been defined as the smallest unit of work. This model also provides for intermediate level metrics to use in the derivation of higher level object's attributes.

The addition of the dynamic model to the OMT provides for the developer to realize the states and actions necessary for the model to behave in the desired fashion. This was helpful in the formal process by identifying which functions would be required for the model. However, during the formal development of the model, the dynamic model was seen as only necessary for understanding system behavior and was not used in the final model. The model was able to capture the required information without the need to simulate dynamic behavior.

4.3 Analysis of Methodology

4.3.1 Contributions to Success. The development of the Rumbaugh model was a very helpful and necessary step in the methodology. Not only was it the perfect way to capture the domain information, it was a great aid in the visualization of the model to be developed. If the domain information was not captured and collected in an object-oriented manner, this research would have stood little chance of success. Also, without some sort of visual representation of the desired system, development would be difficult if not impossible, and the documentation of design decisions and resulting maintenance of the model would also be very difficult.

Another key contributor to the successful implementation of the model in the Refine environment was the transformation of the OMT model to Z schemas. The Z schemas allow the developer to begin thinking functionally as opposed to the more traditional manner: procedurally. The mathematical notations and descriptions of the system provide the initial construct needed to implement the "what" instead of the "how" for the desired model. This transformation led directly to successful implementation of the model in a formal environment. This made the transformation to the Refine executable model almost a one-to-one translation of the Z schemas which were by nature declarative as opposed to procedural. The Refine environment provides the ability to implement a specification of a system using constructs very similar to the specifications in the Z model. The translation phase of the system development was the most challenging but also the most rewarding. Many of the model specifications could be expressed as "what" the system was to do as opposed to "how" the system was to do it. The ability of the Software Refinery environment to deduce the "how" from the "what" was a key to the success of this approach.

The Software Refinery environment was a great asset in the incremental approach to developing the executable model in Refine. Changing and implementing different system specifications was a simple process which required only modifying the desired specifications to the new behavior and a two keystroke recompilation prior to execution. This process usually requires seconds as opposed to minutes or even hours which might have been required to locate and change all of the dependencies in a classical implementation such as the Ada implementation. The next six functions demonstrate the incremental development of the *qualified_for* association. Each new function definition has additional requirements specified which increase the capabilities of the association.

```
function
  SetQualified_For(qf: Qualified_For, w: Worker, j: Job,
```

```
                    ex_accuracy: integer, ex_time: integer) =
    TRUE --> (wqualified_for(qf) = w & jqualified_for(qf) = j &
      expect_accuracy(qf) = ex_accuracy & expect_time(qf) = ex_time)
```

This first increment simply accepts input and sets up the association without regard for what is input.

```
function SetQualified_For2(qf: Qualified_For, w: Worker, j: Job) =
  wafsc(w) in jafsc_set(j) & wskill_level(w) >= jskill_level(j) -->
    (wqualified_for(qf) = w & jqualified_for(qf) = j & expect_time(qf) = jbest_time(j))
```

The second increment compares the *afsc* and *skill_level* of the *Worker* to the *afsc_set* and *skill_level* of the *Job*. If this comparison is okay, the function then uses attributes from the *Worker* and *Job* to establish the *qualified_for* association's *expect_time* attribute using only the *best_time* attribute of the *Job*.

```
function SetQualified_For4(qf: Qualified_For, w: Worker, j: Job) =
  wafsc(w) in jafsc_set(j) & wskill_level(w) > jskill_level(j) -->
 (wqualified_for(qf) = w & jqualified_for(qf) = j & expect_time(qf) = jbest_time(j));
  wafsc(w) in jafsc_set(j) & wskill_level(w) = jskill_level(j) -->
 (wqualified_for(qf) = w & jqualified_for(qf) = j & expect_time(qf) = javerage_time(j));
  wafsc(w) in jafsc_set(j) & wskill_level(w) < jskill_level(j) -->
 (wqualified_for(qf) = w & jqualified_for(qf) = j & expect_time(qf) = jworst_time(j))
```

This increment now makes the association assignment based on the *Job's average_time* and *worst_time* attributes.

```
%% Add qualified_for expect_accuracy checks and assignments.
function SetQualified_For4a(qf: Qualified_For, w: Worker, j: Job) =
  wafsc(w) in jafsc_set(j) & wskill_level(w) > jskill_level(j) -->
    (wqualified_for(qf) = w & jqualified_for(qf) = j & expect_time(qf) =
     jbest_time(j) & expect_accuracy(qf) = jbest_accuracy(j));
  wafsc(w) in jafsc_set(j) & wskill_level(w) = jskill_level(j) -->
    (wqualified_for(qf) = w & jqualified_for(qf) = j &
     expect_time(qf) = javerage_time(j) & expect_accuracy(qf) = javerage_accuracy(j));
  wafsc(w) in jafsc_set(j) & wskill_level(w) < jskill_level(j) -->
    (wqualified_for(qf) = w & jqualified_for(qf) = j &
     expect_time(qf) = jbest_time(j) & expect_accuracy(qf) = jworst_accuracy(j))
```

This increment has the additional checks for the *expect_accuracy* attribute.

```
%% Added axioms to allow other attributes of the worker and job to affect the expect_time
and expect_accuracy attributes of the qualified_for association.
function SetQualified_For5(qf: Qualified_For, w: Worker, j: Job) =
  wafsc(w) in jafsc_set(j) & wskill_level(w) > jskill_level(j) -->
    (wqualified_for(qf) = w & jqualified_for(qf) = j & expect_time(qf) = jbest_time(j));
  wafsc(w) in jafsc_set(j) & wskill_level(w) = jskill_level(j) -->
    (wqualified_for(qf) = w & jqualified_for(qf) = j & expect_time(qf) = javerage_time(j));
  wafsc(w) in jafsc_set(j) & wskill_level(w) < jskill_level(j) -->
    (wqualified_for(qf) = w & jqualified_for(qf) = j & expect_time(qf) = jbest_time(j));
  wafsc(w) in jafsc_set(j) & weducation(w) >= 10 & jcomplexity(j) <= 75 -->
    ((expect_time* = expect_time(qf) --> expect_time(qf) = expect_time* + 20) &
     (expect_accuracy* = expect_accuracy(qf) --> expect_accuracy(qf) =
      expect_accuracy* + 20))
```

This final incremental development of the *SetQualified_For* function now includes specifications which take into account the impact of the *Worker's education* attribute and the along with the *Job's complexity* attribute in the assigning of the association's attributes. It should be noted that this type of development allows for the developer to verify the functionality of the model as it is being developed. Also, many more similar types of specifications could be added if system requirements change.

The ability to use mathematical constructs and set former notation within the Refine environment allows the developer to build powerful and complex constructs in a compact and efficient manner. The following is an example:

```
%%This function calculates all of the accuracy attributes for jobs in the
organization's jobset based on the worker-job assignment and the qualified_for
association expect_accuracy attribute.  Also takes into account the tool-job
assigned association and the supports association percentage attribute.


function CalculateAccuracys(o1: Organization, proj1: Project)=
 enumerate p over range(contents(org-assignment(o1))) do
   jaccuracy(GetJAssignment(p)) <-
     GetExpect_Accuracy(GetQualified_For(org-qualified_for(o1), GetJAssignment(p)))
       + percentage(GetSupports(org-supports(o1), GetJAssignment(p)))
```

Maintenance of the system or updating the model from sponsor input to meet changing requirements is accomplished easily using the current methodology. However, after initial model

development it may not add value to the model to add such changes to the Z model other than to document design changes.

Compactness of the necessary code to extract metrics in both Z and Refine were great when compared to the Ada code to accomplish similar objectives. For example, the Ada pseudo code generated to accomplish the setting of a *Task's* predecessors is as follows:

```
Set_Predecessors
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
```

While the Z construct needed to specify the same objective was much simpler.

Precedes

$$precedes : Job \leftrightarrow Job$$

$$\forall (j_1, j_2) \bullet (j_1, j_2) \in precedes \rightarrow j_1.outputs \cap j_2.inputs \neq \{\ \} \wedge j_1.end\_time \leq j_2.start\_time$$

The corresponding Refine code to actually implement the behavior was quite easy to construct from the Z specification.

```
function GetAllPrecedes(ps: PrecedesSet): Set(Precedes) =
  {pre | (pre: Precedes) (pre in contents(ps) & (joutputs(GetJ1Precedes(pre)) intersect
  jinputs(GetJ2Precedes(pre)) ~= {}) & jend_time(GetJ1Precedes(pre)) <=
 jstart_time(GetJ2Precedes(pre)))}
```

Another comparison of Ada versus Z and Refine appears in the calculating of the time to complete a *Project*. The following comes directly from (28): "The approach for calculating the Total Time To Complete The Scenario is to trace through execution of the scenario keeping track of the order of individual task execution and the associated times. The algorithm for computing this metric is defined as follows:

```
Scenario_Time <- 0
while not Scenario_Complete(Tasks) loop
   loop from 1 .. Num_Of_Task
      if Predecessors_Complete(Current_Task) and
         Worker_Available(Selected_Worker)   and
         Tool_Available(Selected_Tool)       and
         Current_Task.Status /= Complete     then
               Current_Task <- In_Progress
               Selected_Worker.Available <- false
               Selected_Tool.Available <- false
      end if
      Current_Task <- Next_Task
   end loop

   Scenario_Time <- Scenario_Time + Shortest_Task.Time
   Shortest_Task <- Complete
   Selected_Worker.Available <- true
   Selected_Tool.Available <- true

   loop from 1 .. Num_Of_Tasks
      if Current_Task.Status = In_Progress then
         Current_Task.Time = Current_Task.Time - Shortest_Task.Time
      end if
      Current_Task <- Next_Task
   end loop
end loop ''
```

Notice the amount of code needed for the Ada version. Note that in the Ada version the Scenario is equivalent to a *Project* in the Z and Refine. The same information is captured in the following Z specification:

$$total\_time = \sum_{\forall t \in task\_set} t.time$$

The amount of Z necessary is small due to the fact that a discrete event simulation is not in progress as is needed for the Ada implementation. The desired behavior is simply specified. The Refine for the same specification is about as simple.

```
%% Project total time is based on the tasks' times.
function Setptotal_time(p1: Project) =
  TRUE --> ptotal_time(p1) <-
   reduce(+, [ttotal_time(t1) | (t1: Task) t1 in ptask_set(p1)])
```

Notice the amount of information captured in the following Refine specification which is used to calculate the amount of time spent on a *Project* so far giving a snapshot view of the time attribute.

```
%% Calculates project's time so far based on the tasks
function Setptime_so_far(p1: Project) =
  Let (temptask: Task = arb({t1 | (t1: Task) t1 in ptask_set(p1) &
       tstatus(t1) = 'in-process}))
  TRUE --> ptime_so_far(p1) <- tstart_time(temptask) +
           twall_time(temptask) * tpercent_complete(temptask)
```

During the Refine transformation phase some of the Z schemas specified were found to be redundant or specified situations which would never be possible. This somewhat simplified the transformation but added time necessary to make these observations and determine if the specifications were indeed necessary or not.

4.3.2 Limitations to Success. One of the key limitations to the development process was the lack of a Z compiler or checker. Several problems were discovered during the Refine implementation that were incorrect in the Z schemas. These were not apparent during development of the Z model and could only be checked by the developer and other domain specialists. It was then necessary to correct the problems discovered in the Z specifications which required revisiting the Z model. This could have been avoided if there existed the ability to check the Z schemas during development. This further limited the depth of the development possible in the Refine model due to the time requirements to revisit the Z. This could be attributed to the limitations of the currently available tools for formal development. Much current research is concerned with the development of such tools. These tools should greatly increase the power of this methodology.

Although the Software Refinery Environment does much work for the model developer, it does have some limitations. Since this research did not make use of Refine's package environment, any object attributes named within the current environment are global. Therefore, common object attributes, such as *name*, had to be modified from the Z definitions to be totally different (e.g.

toolname and *taskname*). All maps declared within the Refine environment without the use of packaging are global thus requiring unique naming conventions for every individual object's attribute names.

During the translation of the *Z* schemas into Refine, occasionally specifications were discovered which the Software Refinery Environment could not decide how to interpret correctly. For example, it could not decide what to do with the attribute tstatus(t1) in the following specification. The Refine environment wanted to know what the value should be so that it could set it, not just what it could not be. Apparently, it requires more information.

```
%% Refine cannot determine how to do the following:
  (s ~= 'finished) --> (pc < 100)
  (ex j in tjob_set(t1))(jstatus(j) ~= 'finished) =>
   tstatus(t1) ~= 'finished
```

It appears that there currently exists a line between the common procedural style and a style which contains only pure declarative specifications. At times, Refine has some trouble crossing this line.

4.4 Summary

This chapter has analyzed the goals of this research, the contributions to successful accomplishments of these goals, and some of the limitations which left some goals unreachable. This analysis has resulted in identifying the helpful and valuable techniques employed in the formal development process: developing a visual model via Rumbaugh, formally defining the model in *Z* mathematical notation, and transforming the model into executable specifications in Refine. A comparison to an Ada implementation displayed the great diversity and compactness of the formal specifications. This chapter has also identified the limitations of this methodology: the lack of formal tools, and difficulty in fully defining a model's full behavior. The methodology and resulting formal model are the basis for extracting the characteristics of the methodology and model which may lead to a more generalized approach to apply this methodology to organizations outside of the AF domain. The next chapter discusses the generalization of the methodology and model.

V. *Generalization and Reusability of Model*

5.1 Introduction

The previous chapter provided an analysis of the methodology and the resulting model. This chapter describes how the methodology and model might be generalized to provide for maximum reusability. In the original plan, the methodology was to be tested on another domain. However, time constraints did not allow such a test. This chapter also discusses the domain specific aspects of the model. Next, the modeling methodology and its generalization are discussed. The model and its components are examined for their possible reusability. Finally, a proposed use of the methodology on another domain is discussed.

5.2 Generalization of the Modeling Methodology

The methodology followed in the development of the AF organization model was by and large a very general concept to begin with. If a system developer felt that it was necessary to begin from scratch and execute the methodology from the ground up, it would not be a difficult task. The first phase of the methodology is where the domain specific issues are brought out. The Rumbaugh model with its objects and system behavior essentially drive the next two phases of the methodology. In the domain analysis of the Rumbaugh model, the required attributes and model behavior are developed. The development of the Z schemas follows directly from the given Rumbaugh model, and as previously stated, the Refine model is almost a one-to-one transformation of the Z schemas. However, the approach and resulting model are quite general and easily applicable to a large number of organizations which by nature are composed of *Jobs*, *Tools*, *Workers*, and the assignment of *Workers* and *Tools* to accomplish *Jobs*. Although the methodology is easy enough to use, the developer should first look at reusing an existing model.

5.3 Reuse of Model

The existing object model developed for the AF wing domain illustrated in Figure 13 might at first appear to be very domain specific. However, a careful analysis of the model, beginning with the basic objects will reveal that the model is adaptable for use in many other organizations.

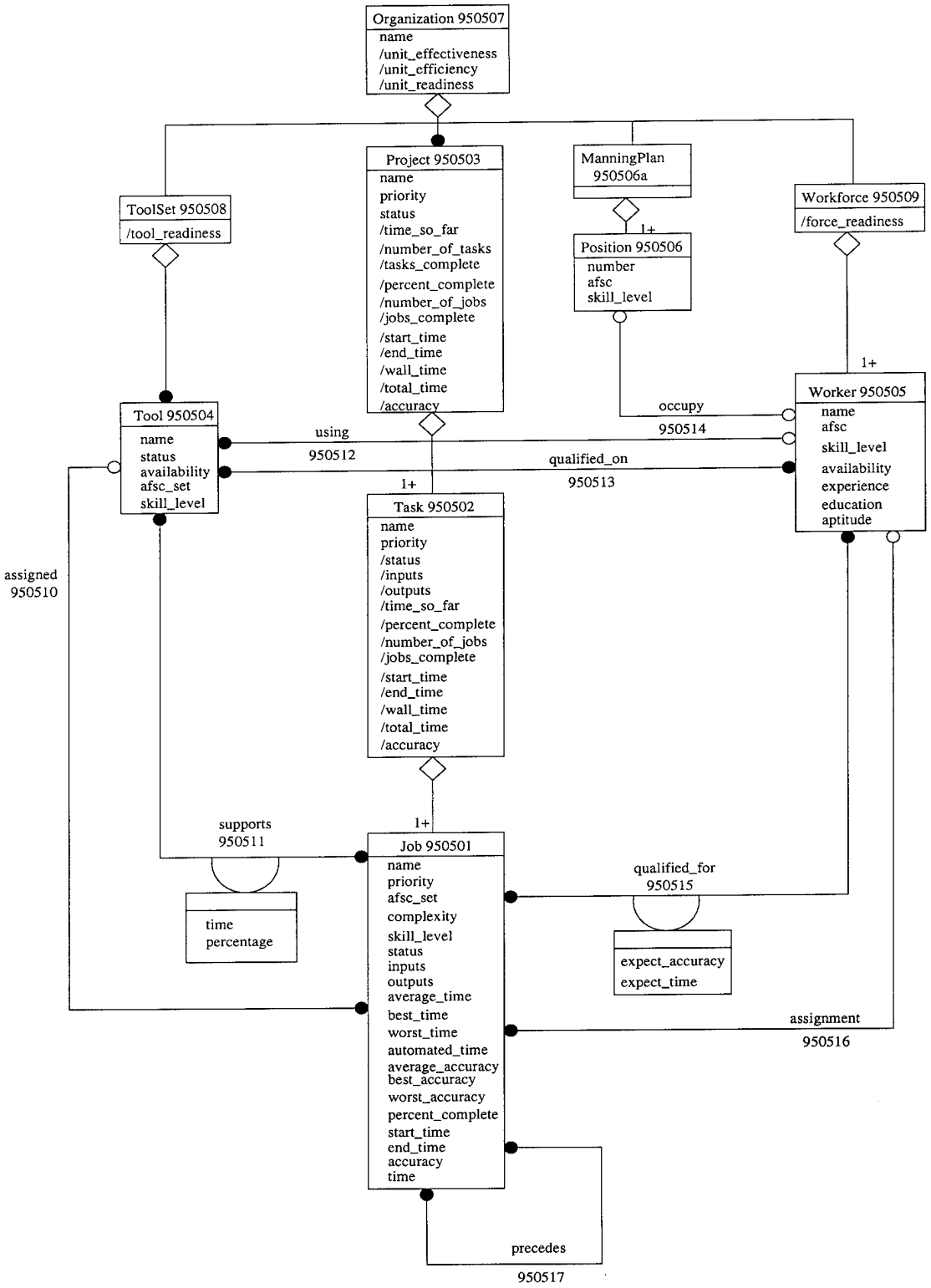

Figure 13. Object Model

5.3.1 Basic Objects. The most basic objects which make up the model are the *Job*, *Worker*, and *Tool*. When examining the *Job*, most of the attributes it contains are general in nature and could be used to describe almost any job in any domain. The two attributes which look domain specific are the *afsc_set* and the *skill_level*. An *afsc* is the job descriptor or code for the class of worker who is preferred to accomplish this *Job*. But, on further review, almost any organization which has multiple different jobs to accomplish has some criteria for determining the workers who can accomplish the *Job*. This might be done through the use of a job number, word description, or organization code. The method used in the organization can easily be adapted into the existing model through the changing of the attribute and possibly its type declaration if necessary. The same can be said for the *skill_level* attribute. If an organization distinguishes between various skills of workers within the same job code, then it can use the existing *skill_level* attribute. If it is not used, then it can be ignored.

Other basic objects are the *Worker*, *Position*, and *Tool*. The attributes of these objects can also be seen to cover many domains. Again, the *afsc*, *afsc_set*, and *skill_level* attributes are the only domain specific ones. These can also be easily tailored to meet the needs of a different domain. The additional attributes of the *Worker* and *Tool* are used to further distinguish between the abilities of particular workers and *Tools* and in the establishment of other object and association attributes. These attributes are also tailorable or removable for a particular domain. This shows that the basic objects of the model are indeed quite general in nature. These basic objects are used to compose the aggregate objects of the model. The next section examines these aggregate objects.

5.3.2 Aggregate Objects. The aggregate objects of the model are composed of sets of basic objects or possibly of other aggregate object as in the case of the *Organization*. These too can be seen as possible to use in many domains. Almost every *Organization* will have *Projects* to complete along with a *Workforce* and *ToolSet* to accomplish the *Projects*. Most *Organizations* also have a desired *ManningPlan* which is the desired composition of the actual *Workforce*. This is used for organization leaders to assess the decisions made in hiring and firing as well as in the readiness of the organization. The attributes of these aggregate objects are almost exclusively derived attributes based on attributes of the basic objects which means that the lower level attributes are read only from the aggregate object's view. The attributes all are common to many organizations and in fact

cover a wide variety of object properties. The *Organization* level aggregate object can be easily modified to contain more attributes to encompass more domain specific metrics. Also, the existing attributes of the *Organization* can be modified for differing domain applications.

5.3.3 Associations. The associations of the model cover a variety of uses within the model. They encompass many possible relationships between the object which may be useful for a particular domain. For instance, if an organizational leader wished to identify which *Workers* were *Qualified_On* which *Tools*, then this information is available the the association. A decision maker might also desire to know what *Workers* are *Qualified_For* what *Jobs* along with the *expected_accuracy* and *expected_time* that the *Worker* would have in accomplishing the *Job* to aid in deciding who to assign. Many other similar associations exist in the current model. This may be seen as overkill; however, the associations deemed to be unnecessary for a particular domain can easily be deleted from the model. These associations can be modified or tailored for a particular domain application just as easily as the objects in the model.

5.3.4 Metrics. The metrics specified within the model include *unit_readiness, unit_efficiency,* and *unit_effectiveness.* These metrics are very broad and can have a variety of meanings within different domains. However, the specifications for the metrics can be easily modified to capture the information necessary for the particular domain of interest. Also, any new metrics needed for the domain can be added to the specifications.

5.4 Proposed Reuse of Methodology and Model

Consider the organizational structure at an institute for higher learning such as AFIT. AFIT can be considered an organization composed of *Workers*: faculty, secretaries, administrators, etc., *Jobs*: teach courses, type letters, evaluate instructors, etc., and *Tools*: overhead projectors, typewriters, computers, etc. The *Tools* and *Workers* are used to accomplish the *Jobs* of the *Organization* (AFIT). Implementing the proposed three phase methodology would require minimum participation from an AFIT domain expert. The expert would need to aid the system developer in identifying the domain specific objects along with the attributes necessary to describe them. Next, the domain expert must help identify the pertinent associations between the objects which are needed

for the Rumbaugh object model. After the object model is complete, the dynamic model must be developed to describe the behavior and states the object can exhibit. The final task for the domain expert is in the defining of the actions of the dynamic model in the functional model.

Now, with a complete Rumbaugh model, the methodology can be continued mainly by the system developer with limited interaction with the domain expert. The model developer needs to transform the Rumbaugh model into the equivalent formal Z schemas to progress towards the executable Refine model. Any discrepancies noticed in the Z schemas can be worked out between the domain expert and the system developer. Once the Z specifications are complete, the transformation to Refine can begin.

The incremental development of the Refine model will be helpful in allowing the domain expert to test specifications as they are implemented and thus verify that the specifications are developed validly. The domain expert can also be involved in the incremental testing of the prototype system as it is developed. The developer can add capabilities to the model in an organized incremental manner all the while making sure that the model maintains its behavior from one stage to another (regression testing) in a baselined approach. Once the model is complete, verification and validation of the model behavior can be tested. Once testing is complete, the Z specifications can be documented into final form as the specifications of the executing model and used as a baseline for the development of a system to implement in the organization on the desired platform.

5.5 Conclusion

This chapter examined the domain specific aspects of the methodology and resulting model and showed how these aspects are not as domain specific as they might at first appear. The model and the methodology both appear to be readily extendable beyond the domain of the AF wing. This chapter has also presented a theoretical model development cycle for a more specific AF organization– the Air Force Institute of Technology. This was shown to be a feasible use for the proposed methodology and could be used to develop a model within this domain and probably many others. The next chapter presents the overall research results and provides recommendations for future research.

VI. Conclusions and Recommendations

6.1 Introduction

This chapter summarizes this research's objectives along with a review of the original problem statement. Next is a comparison of this research's objectives to its accomplishments based on the analysis presented in Chapters IV and V. Finally, suggestions for future research are offered.

6.2 Research Summary

Chapter I established that the primary purpose of this research was to demonstrate the feasibility and benefits of applying formal methods of software engineering technology to the Air Force organization structure. This came directly from the original problem statement.

The AF has not made use of existing knowledge based software engineering and formal methods technology which could provide improved information concerning the status of their commands by using the formal representation and assessment of an AF wing.

More specifically, the research goals, along with how they were accomplished, follows:

- *Analyze the previously existing model of the AF wing* - Improve the object model, dynamic model, and functional model using the formal algebraic language Z. The literature review exposed many areas within the model to improve. These range from the added objects, attributes and associations to the ability to calculate enhanced unit metrics. A dynamic model was added. Chapter III described the transformation of all three parts of the Rumbaugh model into the actual Z formal specifications.
- *Verify and, as much as possible, validate the modified model* - At each stage of development, the model was analyzed by local researchers to verify that the model and its behavior were being implemented correctly. Additionally, each iteration of the development process was checked against the previous specifications for consistency. This ensured that the Rumbaugh model, Z specifications, and Refine implementation were all consistent.
- *Translate the Z formal specifications into an executable specification using the Software Refinery Environment*^{TM} - The Z specifications presented in Chapter III were mapped into executable Refine code during the process model development methodology. This was done

through the utilization of the user-friendly Software Refinery Environment^{TM}. Although not complete, the executable model displays much of the capabilities of the model expressed in Z. It also displays the power and efficiency of using formal methods in model development.

- *Generalize the model by identifying the domain specific constraints and dependencies within the model* - Chapter V discusses the generalization of the model and the methodology. Not only is the methodology shown to be reusable across many domains, the domain specific portions of the actual model itself are shown to be easily modifiable.

- *Show that formal methods can improve the process of evaluating and assessing wing metrics* - This research has shown that the model developed might be helpful in the process of assessing an organization. The methodology can be used to develop specific organizational process models whose behavior can be observed through the executable Refine implementation. This model can be developed incrementally while the added or changed requirement specifications can be validated as development continues. The metrics produced by the model can be enhanced during development or at future dates as the organizations changes and grows.

- *Identify the benefits provided by the use of formal methods* - During the development of the model, it was shown that this methodology which uses formal methods and constructs leads directly to a rapid prototype development effort which allows for behavior observation and limited executability during development. With other formal platforms in development and on the horizon, further interfaces with other platforms will certainly lead to a more specification-oriented development environment based on formal approaches. This might possibly be done in conjunction with such platforms as KIDS/SpecWare (5) or through Algebraically-Based Design Refinement environments such as the one by Wabiszewski (31). Also, if the produced model is reused, the Rumbaugh model and Z specifications will be reused also. This leads directly to cost savings in the development process. It also leads to system specifications which have a mathematical basis.

6.3 Recommendations for Future Research

1. *Implement the methodology on an organization outside of the AF* - A proposed implementation of the methodology is presented in Chapter V. An actual attempt to model such an

organization would demonstrate the reuseability of such a methodology and would greatly enhance the value of this research area.

2. *Add metrics in the Refine model* - Many of the metrics described in the Rumbaugh model and in the *Z* specifications were not implemented in the Refine model. Adding these metrics and validating the *Z* specifications for the metrics would make the Refine model closer to achieving the goals of supporting the sponsor's needs.
3. *Use the KIDS and/or SpecWare tools to develop a scheduling algorithm which optimizes organization metrics* - The current model provides a snapshot of an organization with an already provided assignment schedule of *Workers*, *Tools*, and *Jobs*. If this model could be interfaced with a platform such as KIDS or SpecWare, there might exist a manner in which to create an optimized scheduling algorithm which maximizes the specified unit metrics. This would provide feedback for the organization leaders in the area of optimizing schedules and identifying automation impacts.
4. *Take the development methodology one step further to the translation of the Refine model into an implementation in a fieldable language such as Ada95* - The Software Refinery Environment is not an AF approved new system development platform. However, research is leading to the development of transformation systems which will produce executable code of the desired format, Ada95, from Refine code. This would then make possible the placing of the implemented systems at the organization for use by leaders.
5. *Add a new Job, Task, or Project into the model in real-time, while the current Project is being accomplished* - Since real-time taskings have a great impact on leadership decisions, the ability to model such events would greatly enhance the usefulness of the model. This ability would increase the realism of the model.
6. *Add preemptive scheduling* - If real-time taskings were added, it would be desirable to reschedule *Jobs*, *Tools*, and *Workers*. This would be necessary to allow for the real-time tasking to be dealt with in the proper manner.
7. *Continue validation of the model* - The model was not validated by PACAF domain experts. This model and its behavior should be validated not only by the domain experts but by different AF organizations to determine whether they have properly captured the behavior of the organization. This is a necessary step in determining the true reusability and accuracy of this methodology.

6.4 Final Comments

This research is important because wing decision makers do not currently have a way to systematically represent how an organization performs its mission. This has led to the development of some software systems which are short-sighted or very limited. The domain models developed in this research have captured knowledge about the key objects, operations, and relationships inherent to the typical AF organization. The methodology for developing such models has been shown to

possibly be beneficial to organizations outside the AF domain. This formal development process is simpler, more efficient, more maintainable, and more reusable than more traditional methods of development.

This research successfully developed a domain model of a typical AF organization and showed how the use of formal methods of software engineering could lead to a more precise mathematical representation of the system specifications and ultimately to an executable model. This executable Refine model could be used to assist organization leaders in evaluating their organizations. While more work and further research should be accomplished, this research has shown that formal methods can be effectively applied to problems in an AF organization as well as other organizations outside the AF. These results and the domain model developed through this research can continue to be used in increasing the mission effectiveness and readiness of the AF.

```
!! in-package("RU")
!! in-grammar('user)
#||
------------------------------------------------------------------------
-- Object Name: Assigned
-- Specification Date: 09/12/95
-- Filename: assigned.re
-- Specified by: Hibdon
-- Dependencies:
--
-- Based on: Assigned (950510)
-- Design Transforms and Rationale:
--   Associative object design of association assigned
--   Each instance represents a tool-job link.
--
-- History:
-- 09/12/95 (Hibdon): original design.
--
------------------------------------------------------------------------
||#

%% Define class structure:
var Assigned: object-class subtype-of user-object
    var tassigned: map(Assigned, Tool) = {||}
    var jassigned: map(Assigned, Job) = {||}

%% Define class methods:
function NewAssigned() : Assigned =
  make-object('Assigned)

function ZapAssigned(a: Assigned) =
  erase-object(a)

function InitAssigned(a: Assigned) =
  NIL

function ShowAssigned(a: Assigned) =
  format(TRUE, "~\\pp\\ ~%", a)

function SetAssigned(a: Assigned, t1: Tool, j: Job) =
  TRUE --> (tassigned(a) = t1 & jassigned(a) = j)

function GetTAssigned(a: Assigned): Tool =
  tassigned(a)

function GetJAssigned(a: Assigned): Job =
  jassigned(a)

%%---------------- End of Assigned -----------------------------------

!! in-package("RU")
!! in-grammar('user)
#||
------------------------------------------------------------------------
-- Object Name: AssignedSet
-- Specification Date: 09/12/95
-- Filename: assignedset.re
-- Specified by: Hibdon
--
-- Based on: Assigned (920510)
```

```
-- Design Transforms and Rationale:
--
-- History:
-- 09/12/95 (Hibdon): original design.
--
------------------------------------------------------------------------
||#

%% Define base sets (types):

%% Define class structure:
var AssignedSet: object-class subtype-of Container

%% Define class methods:
function NewAssignedSet() : AssignedSet =
  make-object('AssignedSet)

function InitAssignedSet(as: AssignedSet) =
  InitContainer(as)

function ZapAssignedSet(as: AssignedSet) =
  erase-object(as)

function ShowAssignedSet(as: AssignedSet) =
  format(TRUE, "~\\pp\\ ~%", as)

function ListAssignedSet(as: AssignedSet) =
  enumerate aset over range(contents(as)) do
    format(TRUE, "~\\pp\\ ~%", aset)

function InAssignedSet(as: AssignedSet, j: Job): Boolean =
  ex(as1: assigned)(as1 in contents(as) & GetJAssigned(as1) = j)

%%---------------- End of AssignedSet ----------------------------------

!! in-package("RU")
!! in-grammar('user)
#||
------------------------------------------------------------------------
-- Object Name: Assignment
-- Specification Date: 09/05/95
-- Filename: assignment.re
-- Specified by: Hibdon
-- Dependencies:
--  Must compile and load Faculty, Student.
--
-- Based on: Assignment (950516)
-- Design Transforms and Rationale:
--   Associative object design of association assignment
--   Each instance represents a worker-job link.
--
-- History:
-- 09/05/95 (Hibdon): original design.
--
------------------------------------------------------------------------
||#

%% Define class structure:
var Assignment: object-class subtype-of user-object
    var wassignment: map(Assignment, Worker) = {||}
    var jassignment: map(Assignment, Job) = {||}

%% Define class methods:
function NewAssignment() : Assignment =
```

```
  make-object('Assignment)

function ZapAssignment(p: Assignment) =
  erase-object(p)

function InitAssignment(p: Assignment) =
  NIL

function ShowAssignment(p: Assignment) =
  format(TRUE, "~\\pp\\ ~%", p)

function SetAssignment(as: Assignment, w: Worker, j: Job) =
  TRUE --> (wassignment(as) = w & jassignment(as) = j)

%% Might add (j, w) in qualified_for
function SetAssignment(as: Assignment, w: Worker, j: Job) =
  wafsc(w) in jafsc_set(j)  --> (wassignment(as) = w & jassignment(as) = j)

function GetWAssignment(as: Assignment): Worker =
  wassignment(as)

function GetJAssignment(as: Assignment): Job =
  jassignment(as)

%%---------------- End of Assignment ------------------------------------

!! in-package("RU")
!! in-grammar('user)
#||
--------------------------------------------------------------------------
-- Object Name: AssignmentSet
-- Specification Date: 09/12/95
-- Filename: assignmentset.re
-- Specified by: Hibdon
--
-- Based on: Assignment (950516)
-- Design Transforms and Rationale:
--
-- History:
-- 09/12/95 (Hibdon): original design.
--
--------------------------------------------------------------------------
||#

%% Define base sets (types):

%% Define class structure:
var AssignmentSet: object-class subtype-of Container

%% Define class methods:
function NewAssignmentSet() : AssignmentSet =
  make-object('AssignmentSet)

function InitAssignmentSet(as: AssignmentSet) =
  InitContainer(as)

function ZapAssignmentSet(as: AssignmentSet) =
  erase-object(as)

function ShowAssignmentSet(as: AssignmentSet) =
  format(TRUE, "~\\pp\\ ~%", as)

function ListAssignmentSet(as: AssignmentSet) =
  enumerate aset over range(contents(as)) do
```

```
    format(TRUE, "~\\pp\\ ~%", aset)

function GetAssignment(as: AssignmentSet, j: Job): Assignment =
  arb({ass | (ass: Assignment) (ass in contents(as) & jname(GetJAssignment(ass)) = jname(j))})

function JobInAssignmentSet(as: AssignmentSet, j: Job): Boolean =
  ex(as1: assignment)(as1 in contents(as) & GetJAssignment(as1) = j)

%%---------------- End of AssignmentSet ------------------------------------

!! in-package("RU")
!! in-grammar('user)
#||
------------------------------------------------------------------------
-- Object Name: Container
-- Specification Date: 09/04/95
-- Filename: container.re
-- Specified by: Hartrum
--
-- Based on: Container
-- Design Transforms and Rationale:
--
-- History:
-- 09/04/95 (Hartrum): original design.
-- 09/11/95 (Hibdon): Added GetNext.
------------------------------------------------------------------------
||#

%% Define base sets (types):

%% Define class structure:
var Container: object-class subtype-of user-object
    var cont_name: map(Container, string) = {||}
    var contents: map(Container, seq(object)) = {||}
    var cont_index: map(Container, integer) = {||}

%% Define class methods:
function NewContainer() : Container =
  make-object('Container)

function InitContainer(cont: Container) =
  TRUE --> cont_name(cont) = "" & contents(cont) = [] &
    cont_index(cont) = 0

function ZapContainer(cont: Container) =
  erase-object(cont)

function ShowContainer(cont: Container) =
  format(TRUE, "~\\pp\\ ~%", cont)

function SetContName(cont: Container, cname: string) =
  TRUE --> cont_name(cont) = cname

function GetContName(cont: Container): string =
  cont_name(cont)

function AddItem(cont: Container, item: object) =
  oldcont* = contents(cont) --> contents(cont) = append(oldcont*, item)

function RemoveItem(cont: Container, item: object) =
  i in domain(contents(cont)) & oldcont* = contents(cont) &
    (contents(cont))(i) = item  --> contents(cont) = delete(oldcont*, i)

function GetFirst(cont: Container): object =
```

```
  size(contents(cont)) > 0 --> cont_index(cont) = 1;
  (contents(cont))(1)

function GetNext(cont: Container, item: object): object =
  Let (tempindex: integer = 1)
  i in domain(contents(cont)) & (contents(cont))(i) = item  -->
    tempindex <- (i + 1);
  (contents(cont))(tempindex)

%%---------------- End of Container ----------------------------------

!! in-package("RU")
!! in-grammar('user)
#||
------------------------------------------------------------------------
-- Object Name: Job
-- Specification Date: 08/28/95
-- Filename: job.re
-- Specified by: Hibdon
--
-- Based on: Job (950501)
-- Design Transforms and Rationale:
--
-- History:
-- 08/28/95 (Hibdon): original design.
--
------------------------------------------------------------------------
||#

%% Define base sets (types):
constant AFSCS : set(string) = { "33S3B", "33S3C"}
constant JOB_STATUS : set(symbol) = { 'ready, 'waiting, 'in-process, 'finished}
constant DATA : set(symbol) = { 'i0, 'i1, 'i2, 'i3, 'i4, 'i5, 'i6, 'i7, 'i8,
                'i9, 'i10, 'o1, 'o2, 'o3, 'o4, 'o5, 'o6, 'o7, 'o8, 'o9, 'o10}
var tempjtime: integer = 0

%% Define class structure:
var Job: object-class subtype-of user-object
    var jname: map(Job, string) = {||}
    var jpriority: map(Job, integer) = {||}
    var jafsc_set: map(Job, set(string)) = {||}
    var jcomplexity: map(Job, integer) = {||}
    var jskill_level: map(Job, integer) = {||}
    var jstatus: map(Job, symbol) = {||}
    var jinputs: map(Job, set(symbol)) = {||}
    var joutputs: map(Job, set(symbol)) = {||}
    var javerage_time: map(Job, integer) = {||}
    var jbest_time: map(Job, integer) = {||}
    var jworst_time: map(Job, integer) = {||}
    var jautomated_time: map(Job, integer) = {||}
    var javerage_accuracy: map(Job, integer) = {||}
    var jbest_accuracy: map(Job, integer) = {||}
    var jworst_accuracy: map(Job, integer) = {||}
    var jpercent_complete: map(Job, real) = {||}
    var jstart_time: map(Job, integer) = {||}
    var jend_time: map(Job, integer) = {||}
    var jaccuracy: map(Job, integer) = {||}
    var jtime: map(Job, integer) = {||}

%% Define class methods:
function NewJob() : Job =
  make-object('Job)

function ZapJob(j: Job) =
```

```
  erase-object(j)

function ShowJob(j: Job) =
format(TRUE, " Job name: ~S~% Job priority: ~S~% Afsc_set: ~S~% complexity: ~d~%
    skill_level: ~d~% status: ~s~% inputs: ~s~% outputs: ~s~% average_time: ~d~%
    best_time: ~d~% worst_time: ~d~% automated_time: ~d~% average_accuracy: ~d~%
    best_accuracy: ~d~% worst_accuracy: ~d~% percent_complete: ~d~% start_time:
    ~s~% end_time: ~s~% accuracy: ~s~% time: ~d~%~%~%", jname(j), jpriority(j),
    jafsc_set(j), jcomplexity(j), jskill_level(j), jstatus(j), jinputs(j),
    joutputs(j), javerage_time(j), jbest_time(j), jworst_time(j),
    jautomated_time(j), javerage_accuracy(j), jbest_accuracy(j),
    jworst_accuracy(j), jpercent_complete(j), jstart_time(j), jend_time(j),
    jaccuracy(j), jtime(j))

function Setjname(j: Job, n: string) =
  TRUE --> jname(j) = n

function Getjname(j: Job): string =
  jname(j)

function Setjpriority(j: Job, p: integer) =
  (p >= 0) & (p <= 100) --> jpriority(j) = p;
  (p < 0) OR (p > 100) --> (jpriority(j) = undefined)
%  format(TRUE, "~%priority out of range ~%"))


function Getjpriority(j: Job): integer =
  jpriority(j)

function Setjafsc_set(j: Job, a: set(string)) =
  TRUE --> jafsc_set(j) = a

function Getjafsc_set(j: Job): set(string) =
  jafsc_set(j)

function Setjcomplexity(j: Job, c: integer) =
  (c >= 0) & (c <= 100) --> jcomplexity(j) = c;
  (c < 0) & (c > 100) --> jcomplexity(j) = undefined

function Getjcomplexity(j: Job): integer =
  jcomplexity(j)

function Setjskill_level(j: Job, s: integer) =
  (s >= 1) & (s <= 9) --> jskill_level(j) = s;
  (s < 1) OR (s > 9) --> jskill_level(j) = undefined

function Getjskill_level(j: Job): integer =
  jskill_level(j)

function Setjstatus(j: Job, s: symbol) =
  s ~in JOB_STATUS --> jstatus(j) <- undefined;
  s in JOB_STATUS --> jstatus(j) = s
 %(s ~= 'finished) --> (pc < 100)

function Getjstatus(j: Job): symbol =
  jstatus(j)

function Setjinputs(j: Job, i: set(symbol) ) =
  TRUE --> jinputs(j) = i

function SetjIO(j: Job, i: set(symbol), o: set(symbol)) =
 (i intersect o = {}) --> (jinputs(j) = i & joutputs(j) = o)

function Getjinputs(j: Job): set(symbol) =
```

```
  jinputs(j)

function Setjoutputs(j: Job, o: set(symbol)) =
  TRUE --> joutputs(j) = o

function Getjoutputs(j: Job): set(symbol) =
  joutputs(j)

function Setjaverage_time(j: Job, at: integer ) =
  TRUE --> javerage_time(j) = at

function Getjaverage_time(j: Job): integer =
  javerage_time(j)

function Setjbest_time(j: Job, bt: integer ) =
  TRUE --> jbest_time(j) = bt

function Getjbest_time(j: Job): integer =
  jbest_time(j)

function Setjworst_time(j: Job, wt: integer ) =
  TRUE --> jworst_time(j) = wt

function Getjworst_time(j: Job): integer =
  jworst_time(j)

function Setjautomated_time(j: Job, at: integer ) =
  TRUE --> jautomated_time(j) = at

function Getjautomated_time(j: Job): integer =
  jautomated_time(j)

function Setjaverage_accuracy(j: Job, aa: integer ) =
  TRUE --> javerage_accuracy(j) = aa

function Getjaverage_accuracy(j: Job): integer =
  javerage_accuracy(j)

function Setjbest_accuracy(j: Job, ba: integer ) =
  TRUE --> jbest_accuracy(j) = ba

function Getjbest_accuracy(j: Job): integer =
  jbest_accuracy(j)

function Setjworst_accuracy(j: Job, wa: integer ) =
  TRUE --> jworst_accuracy(j) = wa

function Getjworst_accuracy(j: Job): integer =
  jworst_accuracy(j)

function Setjpercent_complete(j: Job, pc: real ) =
  (pc >= 0.0) & (pc <= 1.0) --> (jpercent_complete(j) = pc);
  (pc = 1.0) --> (jstatus(j) = 'finished);
  (pc > 0.0) & (pc < 1.0) --> (jstatus(j) = 'in-process);
  (pc = 0.0) --> (jstatus(j) = 'waiting)

function Getjpercent_complete(j: Job): real =
  jpercent_complete(j)

function Setjstart_time(j: Job, st: integer ) =
  TRUE --> jstart_time(j) = st

function Getjstart_time(j: Job): integer =
  jstart_time(j)
```

```
function Setjend_time(j: Job, et: integer) =
  (et > jstart_time(j)) --> ((jend_time(j) = et) &
    (tempjtime = et - jstart_time(j)));

  (tempjtime <= jworst_time(j)) & (tempjtime >= jautomated_time(j)) -->
    jtime(j) = tempjtime;

  (et <= jstart_time(j)) --> (jend_time(j) = undefined)

function Getjend_time(j: Job): integer =
  jend_time(j)

function Setjaccuracy(j: Job, a:integer ) =
  TRUE --> jaccuracy(j) = a

function Getjaccuracy(j: Job): integer =
  jaccuracy(j)

%function Setjtime(j: Job) =
 % TRUE --> jtime(j) = jend_time(j) - jstart_time(j)

function Getjtime(j: Job): integer =
  jtime(j)


%%---------------- End of Job -----------------------------------



!! in-package("RU")
!! in-grammar('user)
#||
------------------------------------------------------------------------
-- Object Name: JobSet
-- Specification Date: 09/06/95
-- Filename: jobset1.re
-- Specified by: Hartrum
-- Dependencies:
--    Depends on Container.
--
-- Based on: JobSet (920501a)
-- Design Transforms and Rationale:
--
-- History:
-- 09/06/95 (Hibdon): original design.
--
------------------------------------------------------------------------
||#

%% Define base sets (types):

%% Define class structure:
var JobSet: object-class subtype-of Container
    var jobsetuse: map(JobSet, string) = {||}

%% Define class methods:
function NewJobSet() : JobSet =
  make-object('JobSet)

function InitJobSet(js: JobSet) =
  InitContainer(js)

function ZapJobSet(js: JobSet) =
```

```
  erase-object(js)

function ShowJobSet(js: JobSet) =
  format(TRUE, "~\\pp\\ ~%", js)

function SetJobSetUse(js: JobSet, j_use: string) =
  TRUE --> jobsetuse(js) = j_use

function GetJobSetUse(js: JobSet): string =
  jobsetuse(js)

function GetAllJob(js: JobSet): set(Job) =
  range(contents(js))

function ListJobSet(js: JobSet) =
  enumerate j over range(contents(js)) do
    format(TRUE, "~\\pp\\ ~%", GetjName(j))

function CalculateJobTime(j1: Job)=
  TRUE --> jtime(j1) = 75

function SetJobTimes(js: set(Job)) =
  j in js --> CalculateJobTime(j)


%%---------------- End of JobSet ------------------------------------

!! in-package("RU")
!! in-grammar('user)
#||
--------------------------------------------------------------------------
-- Object Name: ManningPlan
-- Specification Date: 09/11/95
-- Filename: manning.re
-- Specified by: Hibdon
--
-- Based on: ManningPlan (950506a)
-- Design Transforms and Rationale:
--
-- History:
-- 09/11/95 (Hibdon): original design.
--
--------------------------------------------------------------------------
||#

%% Define base sets (types):

%% Define class structure:
var ManningPlan: object-class subtype-of user-object
    var positions: map(ManningPlan, set(Position)) = {||}

%% Define class methods:
function NewManningPlan() : ManningPlan =
  make-object('ManningPlan)

function ZapManningPlan(mp: ManningPlan) =
  erase-object(mp)

function ShowManningPlan(mp: ManningPlan) =
 format(TRUE, "ManningPlan~%  Positions: ~S~%~%~%", positions(mp));
  format(TRUE, "~\\pp\\ ~%", mp)

function Setpositions(mp: ManningPlan, ps: set(Position)) =
  positions* = positions(mp) --> positions(mp) = positions* union ps
```

```
%%---------------- End of ToolSet -----------------------------------

!! in-package("RU")
!! in-grammar('user)
#||
------------------------------------------------------------------------
-- Object Name: Occupy
-- Specification Date: 09/12/95
-- Filename: occupy.re
-- Specified by: Hibdon
-- Dependencies:
--
-- Based on: Occupy (950514)
-- Design Transforms and Rationale:
--   Associative object design of occupy
--   Each instance represents a position-worker link.
--
-- History:
-- 09/12/95 (Hibdon): original design.
--
------------------------------------------------------------------------
||#

%% Define class structure:
var Occupy: object-class subtype-of user-object
    var poccupy: map(Occupy, Position) = {||}
    var woccupy: map(Occupy, Worker) = {||}

%% Define class methods:
function NewOccupy() : Occupy =
  make-object('Occupy)

function ZapOccupy(o: Occupy) =
  erase-object(o)

function InitOccupy(o: Occupy) =
  NIL

function ShowOccupy(o: Occupy) =
  format(TRUE, "~\\pp\\ ~%", o)

function SetOccupy(o: Occupy, p: Position, w: Worker) =
  TRUE --> (poccupy(o) = p & woccupy(o) = w)

function GetPOccupy(o: Occupy): Position =
  poccupy(o)

function GetWOccupy(o: Occupy): Worker =
  woccupy(o)

%%---------------- End of Occupy -----------------------------------

!! in-package("RU")
!! in-grammar('user)
#||
------------------------------------------------------------------------
-- Object Name: OccupySet
-- Specification Date: 09/12/95
-- Filename: occupyset.re
-- Specified by: Hibdon
--
-- Based on: Occupy (950514)
-- Design Transforms and Rationale:
```

```
--
-- History:
-- 09/12/95 (Hibdon): original design.
--
-------------------------------------------------------------------------
||#

%% Define base sets (types):

%% Define class structure:
var OccupySet: object-class subtype-of Container

%% Define class methods:
function NewOccupySet() : OccupySet =
  make-object('OccupySet)

function InitOccupySet(os: OccupySet) =
  InitContainer(os)

function ZapOccupySet(os: OccupySet) =
  erase-object(os)

function ShowOccupySet(os: OccupySet) =
  format(TRUE, "~\\pp\\ ~%", os)

function ListOccupySet(os: OccupySet) =
  enumerate osset over range(contents(os)) do
    format(TRUE, "~\\pp\\ ~%", osset)

%% Correctly occupied positions based on afsc only.
function GetCorrectlyOccupiedPositions(os: OccupySet): Set(Occupy) =
  {occ | (occ: Occupy) (occ in contents(os) & posafsc(GetPOccupy(occ)) = wafsc(GetWOccupy(occ)))}

%% Correctly occupied positions based on afsc and skill level.
function GetCorrectlyOccupiedPositions2(os: OccupySet): Set(Occupy) =
   {occ | (occ: Occupy) (occ in contents(os) & posafsc(GetPOccupy(occ)) =
    wafsc(GetWOccupy(occ)) & wskill_level(GetWOccupy(occ)) >=
    posskill_level(GetPOccupy(occ)))}

%%---------------- End of OccupySet -----------------------------------

!! in-package("RU")
!! in-grammar('user)
#||
-------------------------------------------------------------------------
-- Object Name: Organization
-- Specification Date: 09/08/95
-- Filename: org.re
-- Specified by: Hibdon
--
-- Based on: Organization (950507)
-- Design Transforms and Rationale:
--
-- History:
-- 09/08/95 (Hibdon): original design.
--
-------------------------------------------------------------------------
||#

%% Define base sets (types):
constant JOB_STATUS : set(symbol) = { 'ready, 'waiting, 'in-process, 'finished}
constant DATA : set(symbol) = { 'i1, 'i2, 'i3, 'o1, 'o2, 'o3}
var temptime: integer = 0
```

```
%% Define class structure:
var Organization: object-class subtype-of user-object
%% Attributes
    var oname: map(Organization, string) = {||}
    var unit_effectiveness: map(Organization, integer) = {||}
    var unit_efficiency: map(Organization, integer) = {||}
    var unit_readiness: map(Organization, real) = {||}
    var org-toolset: map(Organization, ToolSet) = {||}
    var org-workforce: map(Organization, Workforce) = {||}
    var org-projectset: map(Organization, Set(Project)) = {||}
    var org-taskset: map(Organization, Set(Task)) = {||}
    var org-jobset: map(Organization, Set(Job)) = {||}
%% Associations
    var org-using: map(Organization, usingset) = {||}
    var org-qualified_on: map(Organization, qualified_onset) = {||}
    var org-assigned: map(Organization, assignedset) = {||}
    var org-supports: map(Organization, supportsset) = {||}
    var org-assignment: map(Organization, assignmentset) = {||}
    var org-qualified_for: map(Organization, qualified_forset) = {||}
    var org-precedes: map(Organization, precedesset) = {||}
    var org-occupy: map(Organization, occupyset) = {||}

%% Temp Globals
var tempusing: Using = undefined
var tempqualifiedon: Qualified_On = undefined
var tempoccupy: Occupy = undefined
var tempassigned: Assigned = undefined
var tempsupports: Supports = undefined
var tempqualifiedfor: Qualified_For = undefined
var tempassignment: Assignment = undefined
var tempprecedes: Precedes = undefined
var tempworker: Worker = undefined
var temptool: Tool = undefined
var temptime*: integer = 0
var tempposiblequalifiedon: Qualified_On = undefined
var org-possiblequalified_onset: Qualified_OnSet = undefined

%% Define class methods:
function NewOrganization() : Organization =
  make-object('Organization)

function ZapOrganization(o1: Organization) =
  erase-object(o1)

function InitOrg(o1: Organization) =
%% Initialize attributes
  oname(o1) <- "Air Force Organization";
  unit_effectiveness(o1) <- 0;
  unit_efficiency(o1) <- 0;
  unit_readiness(o1) <- 0.0;
%% Initialize Association sets
  org-using(o1) <- NewUsingSet();
  InitUsingSet(org-using(o1));
  org-qualified_on(o1) <- NewQualified_OnSet();
  InitQualified_OnSet(org-qualified_on(o1));
  org-possiblequalified_onset <- NewQualified_OnSet();
  InitQualified_OnSet(org-possiblequalified_onset);
%  org-qualified_on(o1) <- {};
  org-assigned(o1) <- NewAssignedSet();
  InitAssignedSet(org-assigned(o1));
  org-supports(o1) <- NewSupportsSet();
  InitSupportsSet(org-supports(o1));
  org-assignment(o1) <- NewAssignmentSet();
  InitAssignmentSet(org-assignment(o1));
```

```
  org-qualified_for(o1) <- NewQualified_ForSet();
  InitQualified_ForSet(org-qualified_for(o1));
  org-precedes(o1) <- NewPrecedesSet();
  InitPrecedesSet(org-precedes(o1));
  org-occupy(o1) <- NewOccupySet();
  InitOccupySet(org-occupy(o1));

%% Initialize Project, Task and Job sets, etc

  org-projectset(o1) <- {};
  org-taskset(o1) <- {};
  org-jobset(o1) <- {};
  org-workforce(o1) <- NewWorkforce();
  InitWorkforce(org-workforce(o1));
  org-toolset(o1) <- NewToolSet();
  InitToolSet(org-toolset(o1))

function AddWorkertowf(o1: Organization, w1: Worker)=
  TRUE --> (tempworker <- NewWorker();
            Setwname(tempworker, wname(w1));
            Setwafsc(tempworker, wafsc(w1));
            Setwskill_level(tempworker, wskill_level(w1));
            Setwavailability(tempworker, wavailability(w1));
            Setwexperience(tempworker, wexperience(w1));
            Setweducation(tempworker, weducation(w1));
            Setwaptitude(tempworker, waptitude(w1));
            AddItem(org-workforce(o1), tempworker))

function AddTooltots(o1: Organization, t1: Tool)=
  TRUE --> (temptool <- NewTool();
            Settoolname(temptool, toolname(t1));
            Settoolstatus(temptool, toolstatus(t1));
            Settoolavailability(temptool, toolavailability(t1));
            Settoolafsc_set(temptool, toolafsc_set(t1));
            Settoolskill_level(temptool, toolskill_level(t1));
            AddItem(org-toolset(o1), temptool))

%% Functions to add members to association sets
function AddUsing(o1: Organization, t1: Tool, w1: Worker)=
  TRUE --> (tempusing <- NewUsing();
            SetUsing(tempusing, t1, w1);
            AddItem(org-using(o1), tempusing))

function AddQualified_On(o1: Organization, t1: Tool, w1: Worker)=
  TRUE --> (tempqualifiedon <- NewQualified_On();
            SetQualified_On(tempqualifiedon, t1, w1);
            AddItem(org-qualified_on(o1), tempqualifiedon))

function AddPossibleQualified_On(o1: Organization, t1: Tool, w1: Worker)=
  TRUE --> (temppossiblequalifiedon <- NewQualified_On();
            SetQualified_On(temppossiblequalifiedon, t1, w1);
            AddItem(org-possiblequalified_onset, temppossiblequalifiedon))

function AddOccupy(o1: Organization, p1: Position, w1: Worker)=
  TRUE --> (tempoccupy<- NewOccupy();
            SetOccupy(tempoccupy, p1, w1);
            AddItem(org-occupy(o1), tempoccupy))

function AddAssigned(o1: Organization, t1: Tool, j1: Job)=
  TRUE --> (tempassigned <- NewAssigned();
            SetAssigned(tempassigned, t1, j1);
            AddItem(org-assigned(o1), tempassigned))

function AddSupports(o1: Organization, t1: Tool, j1: Job, time1: integer, percent1: integer)=
```

```
  TRUE --> (tempsupports<- NewSupports();
            SetSupports(tempsupports, t1, j1, time1, percent1);
            AddItem(org-supports(o1), tempsupports))

function AddQualified_For(o1: Organization, w1: Worker, j1: Job, accuracy1: integer, time1: integer)=
  TRUE --> (tempqualifiedfor<- NewQualified_For();
            SetQualified_For(tempqualifiedfor, w1, j1, accuracy1, time1);
            AddItem(org-qualified_for(o1), tempqualifiedfor))

function AddQualified_For2(o1: Organization, w1: Worker, j1: Job)=
  TRUE --> (tempqualifiedfor<- NewQualified_For();
            SetQualified_For4a(tempqualifiedfor, w1, j1);
            AddItem(org-qualified_for(o1), tempqualifiedfor))

function AddAssignment(o1: Organization, w1: Worker, j1: Job)=
  TRUE --> (tempassignment <- NewAssignment();
            SetAssignment(tempassignment, w1, j1);
            AddItem(org-assignment(o1), tempassignment))

function AddAssignment(o1: Organization, w1: Worker, j1: Job)=
  TRUE --> (tempassignment <- NewAssignment();
            SetAssignment(tempassignment, w1, j1);
            AddItem(org-assignment(o1), tempassignment))

function AddPrecedes(o1: Organization, j1: Job, j2: Job)=
%% First add the new member
  TRUE --> (tempprecedes <- NewPrecedes();
            SetPrecedes(tempprecedes, j1, j2, o1);
            AddItem(org-precedes(o1), tempprecedes));
%% Now check for intersection and remove if necessary
  joutputs(j1) intersect jinputs(j2) = {} --> RemoveItem(org-precedes(o1), tempprecedes)


%% This function will not display undefined attributes
function ShowOrg(o1: Organization) =
  format(TRUE, "~\\pp\\~%", o1)

%% This function will display undefined attributes
function ShowOrganization(o1: Organization) =
  format(TRUE, " Organization name: ~S~% unit_effectiveness: ~d~%
   unit_efficiency: ~d~% unit_readiness: ~d~% org-using: ~s~% org-qualified_on:
   ~s~% org-assigned: ~s~% org-supports: ~s~% org-assignment: ~s~%
   org-qualified_for: ~s~% org-precedes: ~s~% org-occupy: ~s~% org-jobset: ~s~%
   org-taskset: ~S~% org-projectset: ~s~%~%~%", oname(o1),
   unit_effectiveness(o1), unit_efficiency(o1), unit_readiness(o1),
   org-using(o1), org-qualified_on(o1), org-assigned(o1), org-supports(o1),
   org-assignment(o1), org-qualified_for(o1), org-precedes(o1), org-occupy(o1),
   org-jobset(o1), org-taskset(o1), org-projectset(o1))

function Setoname(o1: Organization, n: string) =
  TRUE --> oname(o1) = n

function Getoname(o1: Organization): string =
  oname(o1)

%% Checks boundary conditions

%% Checks boundary conditions
function Setunit_effectiveness(o1: Organization, ue: integer) =
  (ue >= 0) & (ue <= 100)  --> unit_effectiveness(o1) = ue;
  (ue < 0) OR (ue > 100) --> unit_effectiveness(o1) = undefined

function Getunit_effectiveness(o1: Organization): integer =
  unit_effectiveness(o1)
```

```
%% Checks boundary conditions
function Setunit_efficiency(o1: Organization, uef: integer) =
  (uef >= 0) & (uef <= 100)  --> unit_efficiency(o1) = uef;
  (uef < 0) OR (uef > 100) --> unit_efficiency(o1) = undefined

function Getunit_effectiveness(o1: Organization): integer =
  unit_effectiveness(o1)

%% Sets unit readiness manually
function Setunit_readiness1(o1: Organization, ur: real) =
  (ur >= 0) & (ur <= 100)  --> unit_readiness(o1) = ur;
  (ur < 0) OR (ur > 100) --> unit_readiness(o1) = undefined

%% Calculates unit readiness based on ToolSet and Workforce
function Setunit_readiness(o1: Organization) =
  TRUE  --> unit_readiness(o1) = GetTool_Readiness(org-toolset(o1)) + GetForce_Readiness(org-workforce(o1))

function Getunit_readiness(o1: Organization): real =
  unit_readiness(o1)

%% Calculates all of the time attributes for jobs in the organization's jobset
   based on the worker-job assignment and the qualified_for association
   expect_time attribute.

function CalculateTimes(o1: Organization, proj1: Project)=
    enumerate p over range(contents(org-assignment(o1))) do
      jtime(GetJAssignment(p)) <-
      GetExpect_Time(GetQualified_For(org-qualified_for(o1), GetJAssignment(p)))

%% Now takes into account the tool-job assigned association and the supports
   association time attribute

function CalculateTimes(o1: Organization, proj1: Project)=
    enumerate p over range(contents(org-assignment(o1))) do
      jtime(GetJAssignment(p)) <-
      GetExpect_Time(GetQualified_For(org-qualified_for(o1), GetJAssignment(p)))
      - suptime(GetSupports(org-supports(o1), GetJAssignment(p)))

function CalculateAccuracys(o1: Organization, proj1: Project)=
    fa(j)((j in org-jobset(o1) & JobInAssignmentSet(org-assignment(o1), j)) =>
      jaccuracy(j) <-
      GetExpect_Accuracy(GetQualified_For(org-qualified_for(o1), j)))

%% Calculates all of the accuracy attributes for jobs in the organization's
   jobset based on the worker-job assignment and the qualified_for association
   expect_accuracy attribute.  Also takes into account the tool-job assigned
   association and the supports association percentage attribute

function CalculateAccuracys(o1: Organization, proj1: Project)=
 enumerate p over range(contents(org-assignment(o1))) do
  jaccuracy(GetJAssignment(p)) <-
  GetExpect_Accuracy(GetQualified_For(org-qualified_for(o1), GetJAssignment(p)))
  + percentage(GetSupports(org-supports(o1), GetJAssignment(p)))

function Setorg-projectset(o1: Organization, ps: Set(Project))=
  True --> org-projectset(o1) <- ps

%% Functions to find and assign all of the tasks and jobs of an organization
function Getorg-taskset(o1: Organization)=
  TRUE --> org-taskset(o1) = reduce(union, {ptask_set(p1) |
    (p1: Project) p1 in org-projectset(o1)})

function Getorg-jobset(o1: Organization): set(Job) =
```

```
  TRUE --> org-jobset(o1) = reduce(union, {pjob_set(p1) |
   (p1: Project) p1 in org-projectset(o1)});
    org-projectset(o1)

function Showorg-jobset(o1: Organization)=
  enumerate j over org-jobset(o1) do
   format(TRUE, "~\\pp\\ ~%", j)

%%---------------- End of Organization ----------------------------------

!! in-package("RU")
!! in-grammar('user)
#||
------------------------------------------------------------------------
-- Object Name: Position
-- Specification Date: 08/28/95
-- Filename: position.re
-- Specified by: Hibdon
--
-- Based on: Position (950506)
-- Design Transforms and Rationale:
--
-- History:
-- 08/28/95 (Hibdon): original design.
--
------------------------------------------------------------------------
||#

%% Define base sets (types):
constant AFSCS : set(string) = { "33S3B", "33S3C", "4921"}

%% Define class structure:
var Position: object-class subtype-of user-object
    var posnumber: map(Position, integer) = {||}
    var posafsc: map(Position, string) = {||}
    var posskill_level: map(Position, integer) = {||}

%% Define class methods:
function NewPosition() : Position =
  make-object('Position)

function ZapPosition(p: Position) =
  erase-object(p)

function ShowPosition(p: Position) =
 format(TRUE, " Position number: ~d~% Afsc: ~S~% skill_level: ~d~%~%~%~%",
   posnumber(p), posafsc(p), posskill_level(p))

function Setposnumber(p: Position, n: integer) =
  TRUE --> posnumber(p) = n

function Getposnumber(p: Position): integer =
  posnumber(p)

%% Checks if afsc is valid
function Setposafsc(p: Position, a: string) =
  a in AFSCS --> posafsc(p) = a;
  a ~in AFSCS --> posafsc(p) = undefined

function Getposafsc(p: Position): string =
  posafsc(p)

%% Checks boundary ranges
function Setposskill_level(p: Position, s: integer) =
```

```
  (s >= 1 & s <= 9) --> posskill_level(p) = s

function Getposskill_level(p: Position): integer =
  posskill_level(p)


%%---------------- End of Position ------------------------------------

!! in-package("RU")
!! in-grammar('user)
#||
--------------------------------------------------------------------------
-- Object Name: Precedes
-- Specification Date: 09/12/95
-- Filename: precedes.re
-- Specified by: Hibdon
-- Dependencies:
--  Must compile and load Tool, Worker.
--
-- Based on: Precedes (950517)
-- Design Transforms and Rationale:
--   Associative object design of precedes
--   Each instance represents a Job-Job link.
--
-- History:
-- 09/12/95 (Hibdon): original design.
--
--------------------------------------------------------------------------
||#

%% Define class structure:
var Precedes: object-class subtype-of user-object
    var j1precedes: map(Precedes, Job) = {||}
    var j2precedes: map(Precedes, Job) = {||}

%% Define class methods:
function NewPrecedes() : Precedes =
  make-object('Precedes)

function ZapPrecedes(p: Precedes) =
  erase-object(p)

function InitPrecedes(p: Precedes) =
  NIL

function ShowPrecedes(p: Precedes) =
  format(TRUE, "~\\pp\\ ~%", p)

function SetPrecedes(p: Precedes, j1: Job, j2: Job) =
  TRUE --> (j1precedes(p) = j1 & j2precedes(p) = j2)

%% The next two functions display incremental development of the same function.
   If compiled as follows, the last function will be the one recognized and used
   by the system.

%% Input and outputs are checked as well as start and stop times.
function SetPrecedes(p: Precedes, j1: Job, j2: Job) =
  joutputs(j1) intersect jinputs(j2) ~= {} & jend_time(j1) <= jstart_time(j2)
   --> (j1precedes(p) = j1 & j2precedes(p) = j2)

%% Safety check: item is removed if intersect is empty.
function SetPrecedes(p: Precedes, j1: Job, j2: Job, o1: Organization) =
  (joutputs(j1) intersect jinputs(j2) ~= {} & jend_time(j1) <= jstart_time(j2))
   --> (j1precedes(p) = j1 & j2precedes(p) = j2);
```

```
  joutputs(j1) intersect jinputs(j2) = {} --> RemoveItem(org-precedes(o1), p)

function Getj1Precedes(p: Precedes): Job =
  j1precedes(p)

function Getj2Precedes(p: Precedes): Job =
  j2precedes(p)

%%---------------- End of Precedes -----------------------------------

!! in-package("RU")
!! in-grammar('user)
#||
------------------------------------------------------------------------
-- Object Name: PrecedesSet
-- Specification Date: 09/12/95
-- Filename: precedesset.re
-- Specified by: Hibdon
--
-- Based on: Precedes (950517)
-- Design Transforms and Rationale:
--
-- History:
-- 09/12/95 (Hibdon): original design.
--
------------------------------------------------------------------------
||#

%% Define base sets (types):

%% Define class structure:
var PrecedesSet: object-class subtype-of Container

%% Define class methods:
function NewPrecedesSet() : PrecedesSet =
  make-object('PrecedesSet)

function InitPrecedesSet(ps: PrecedesSet) =
  InitContainer(ps)

function ZapPrecedesSet(ps: PrecedesSet) =
  erase-object(ps)

function ShowPrecedesSet(ps: PrecedesSet) =
  format(TRUE, "~\\pp\\ ~%", ps)

% Might want to do it this way instead of checking in SetPrecedes
function GetAllPrecedes(ps: PrecedesSet): Set(Precedes) =
  {pre | (pre: Precedes) (pre in contents(ps) & (joutputs(GetJ1Precedes(pre))
   intersect jinputs(GetJ2Precedes(pre)) ~= {}) & jend_time(GetJ1Precedes(pre))
   <= jstart_time(GetJ2Precedes(pre)))}

%function GetAllPrecedes(js: Set(Job)): Set(Precedes) =
%  {pre | (pre: Precedes) (pre in contents(ps) & (joutputs(GetJ1Precedes(pre))
    intersect jinputs(GetJ2Precedes(pre)) ~= {}) & jend_time(GetJ1Precedes(pre))
    <= jstart_time(GetJ2Precedes(pre)))}

function ListPrecedesSet(ps: PrecedesSet) =
  enumerate pset over range(contents(ps)) do
    format(TRUE, "~\\pp\\ ~%", pset)

%%---------------- End of PrecedesSet --------------------------------

!! in-package("RU")
```

```
!! in-grammar('user)
#||
------------------------------------------------------------------------
-- Object Name: Project
-- Specification Date: 08/28/95
-- Filename: project.re
-- Specified by: Hibdon
--
-- Based on: Project (950503)
-- Design Transforms and Rationale:
--
-- History:
-- 08/28/95 (Hibdon): original design.
--
------------------------------------------------------------------------
||#

%% Define base sets (types):
constant JOB_STATUS : set(symbol) = { 'ready, 'waiting, 'in-process, 'finished}
constant DATA : set(symbol) = { 'i1, 'i2, 'i3, 'o1, 'o2, 'o3}
var temptime: integer = 0

%% Define class structure:
var Project: object-class subtype-of user-object
    var pname: map(Project, string) = {||}
    var ppriority: map(Project, integer) = {||}
    var pstatus: map(Project, symbol) = {||}
    var ptime_so_far: map(Project, real) = {||}
    var pnumber_of_tasks: map(Project, integer) = {||}
    var ptasks_complete: map(Project, integer) = {||}
    var ppercent_complete: map(Project, real) = {||}
    var pnumber_of_jobs: map(Project, integer) = {||}
    var pjobs_complete: map(Project, integer) = {||}
    var pstart_time: map(Project, integer) = {||}
    var pend_time: map(Project, integer) = {||}
    var pwall_time: map(Project, integer) = {||}
    var ptotal_time: map(Project, real) = {||}
    var paccuracy: map(Project, integer) = {||}
%% maps from an object to a set of objects without using the container.
   Could have made new objects of container type called JobSet and TaskSet.
    var ptask_set: map(Project, set(Task)) = {||}
    var pjob_set: map(Project, set(Job)) = {||}

%% Define class methods:
function NewProject() : Project =
  make-object('Project)

function ZapProject(p1: Project) =
  erase-object(p1)

function ShowProject(p1: Project) =
  format(TRUE, " Project name: ~S~% Project priority: ~S~% status: ~s~%
   time_so_far: ~d~% number_of_tasks: ~d~% tasks_complete: ~d~%
   percent_complete: ~d~% number_of_jobs: ~d~% jobs_complete: ~d~% start_time:
   ~s~% end_time: ~s~% wall_time: ~d~% total_time: ~d~% accuracy: ~s~% task_set:
   ~s~% job_set: ~s~%~%~%", pname(p1), ppriority(p1), pstatus(p1),
   ptime_so_far(p1), pnumber_of_tasks(p1), ptasks_complete(p1),
   ppercent_complete(p1), pnumber_of_jobs(p1), pjobs_complete(p1),
   pstart_time(p1), pend_time(p1), pwall_time(p1), ptotal_time(p1),
   paccuracy(p1), ptask_set(p1), pjob_set(p1))

function Setpname(p1: Project, n: string) =
  TRUE --> pname(p1) = n
```

```
function Getpname(p1: Project): string =
  pname(p1)

%% Checking boundary ranges
function Setppriority(p1: Project, p: integer) =
  (p >= 0) & (p <= 100) --> ppriority(p1) = p;
  (p < 0) OR (p > 100) --> (ppriority(p1) = undefined)

function Getppriority(p1: Project): integer =
  ppriority(p1)

function Setpstatus(p1: Project) =
  fa(t1)((t1 in ptask_set(p1) & tstatus(t1) = 'finished) =>
    pstatus(p1) = 'finished);
  fa(t1)((t1 in ptask_set(p1) & tstatus(t1) = 'waiting) =>
    pstatus(p1) = 'waiting);
  fa(t1)((t1 in ptask_set(p1) & tstatus(t1) = 'ready) =>
    pstatus(p1) = 'ready);
  ex(t1)(t1 in ptask_set(p1) & tstatus(t1) = 'in-process) =>
    pstatus(p1) <- 'in-process

%% Retrieves the project status based on the status of the tasks in the
   project's taskset.

function Getpstatus(p1: Project): symbol =
  fa(t1)((t1 in ptask_set(p1) & tstatus(t1) = 'finished) =>
    pstatus(p1) = 'finished);
  fa(t1)((t1 in ptask_set(p1) & tstatus(t1) = 'waiting) =>
    pstatus(p1) = 'waiting);
  fa(t1)((t1 in ptask_set(p1) & tstatus(t1) = 'ready) =>
    pstatus(p1) = 'ready);
  ex(t1)(t1 in ptask_set(p1) & tstatus(t1) = 'in-process) =>
    pstatus(p1) <- 'in-process;
  pstatus(p1)

%% Calculates project's time so far based on the tasks
function Setptime_so_far(p1: Project) =
  Let (temptask: Task = arb({t1 | (t1: Task) t1 in ptask_set(p1) &
       tstatus(t1) = 'in-process}))
  TRUE --> ptime_so_far(p1) <- tstart_time(temptask) + twall_time(temptask) *
           tpercent_complete(temptask)

function Getptime_so_far(p1: Project): real =
  ptime_so_far(p1)

%% Project percent complete based on the amount of time for finished tasks
   divided by the total time for all tasks.  Next step is to add range
   checking (if necessary).

function Setppercent_complete(p1: Project) =
  size([ttotal_time(t1) | (t1: Task) t1 in ptask_set(p1) &
   tstatus(t1) = 'finished]) = 0 --> ppercent_complete(p1) = 0.0;
  TRUE --> ppercent_complete(p1) <-
   ((reduce(+,[ttotal_time(t1) | (t1: Task) t1 in ptask_set(p1) &
   tstatus(t1) = 'finished]))/
   (reduce(+,[ttotal_time(t1) | (t1: Task) t1 in ptask_set(p1)])))

function Getppercent_complete(p1: Project): real =
  ppercent_complete(p1)

function Setpnumber_of_jobs(p1: Project) =
  TRUE --> pnumber_of_jobs(p1) <- size(pjob_set(p1))

function Getpnumber_of_jobs(p1: Project): integer =
```

```
  pnumber_of_jobs(p1)

function Setpnumber_of_tasks(p1: Project) =
  TRUE --> pnumber_of_tasks(p1) <- size(ptask_set(p1))

function Getpnumber_of_tasks(p1: Project): integer =
  pnumber_of_tasks(p1)

%% Finds number of jobs complete through the tasks in the taskset.
   Also checks to determine if the project status is finished.

function Setpjobs_complete(p1: Project) =
  TRUE --> pjobs_complete(p1) =
    size({j1|(j1: Job) j1 in pjob_set(p1) & jstatus(j1) = 'finished});
  pnumber_of_jobs(p1) = size({j1 | (j1: Job) j1 in pjob_set(p1) &
   jstatus(j1) = 'finished}) --> pjobs_complete(p1) =
   size({j1 | (j1: Job) j1 in pjob_set(p1) & jstatus(j1) = 'finished});
   pjobs_complete(p1) = pnumber_of_jobs(p1) -->
   ppercent_complete(p1) = 1.0 & pstatus(p1) = 'finished

function Getpjobs_complete(p1: Project): integer =
  pjobs_complete(p1)

%% Finds the number of tasks complete in the project and checks if the project
   is finished.

function Setptasks_complete(p1: Project) =
  TRUE --> ptasks_complete(p1) =
    size({t1|(t1: Task) t1 in ptask_set(p1) & tstatus(t1) = 'finished});
  pnumber_of_tasks(p1) = size({t1 | (t1: Task) t1 in ptask_set(p1) &
   tstatus(t1) = 'finished}) --> ptasks_complete(p1) =
   size({t1 | (t1: Task) t1 in ptask_set(p1) & tstatus(t1) = 'finished});
   ptasks_complete(p1) = pnumber_of_jobs(p1) -->
   ppercent_complete(p1) = 1.0 & pstatus(p1) = 'finished

function Getptasks_complete(p1: Project): integer =
  pjobs_complete(p1)

%% Project start time is determined by the lowest task start time.
function SetpStart_Time(p1: Project) =
  TRUE --> pstart_time(p1) <-
   tstart_time(arb({t1 | (t1: Task) t1 in ptask_set(p1) &
   (fa(t2) (t2 in ptask_set(p1) => tstart_time(t1) <= tstart_time(t2)))}))

function Getpstart_time(p1: Project): integer =
  pstart_time(p1)

%% Project end time is determined by the last task end time.
function Setpend_time(p1: Project) =
  TRUE --> pend_time(p1) <-
   tend_time( arb({t1 | (t1: Task) t1 in ptask_set(p1) &
   (fa(t2) (t2 in ptask_set(p1) => tend_time(t1) >= tend_time(t2)))}))

function Getpend_time(p1: Project): integer =
  pend_time(p1)

function Setpwall_time(p1: Project) =
  TRUE --> pwall_time(p1) = pend_time(p1) - pstart_time(p1)

function Getpwall_time(p1: Project): integer =
  pwall_time(p1)

%% Project total time is based on the tasks' times.
function Setptotal_time(p1: Project) =
```

```
  TRUE --> ptotal_time(p1) <-
   reduce(+, [ttotal_time(t1) | (t1: Task) t1 in ptask_set(p1)])

function Getptotal_time(p1: Project): real =
  ptotal_time(p1)

%% Project accuracy is based on the tasks' accuracies.
function Setpaccuracy(p1: Project) =
  TRUE --> paccuracy(p1) <-
  real-to-nearest-integer(reduce(+,[taccuracy(t1)|(t1: Task) t1 in ptask_set(p1)])/size(ptask_set(p1)))

function Getpaccuracy(p1: Project): integer =
  paccuracy(p1)

function Setptask_set(p1: Project, ts: set(Task)) =
  ptask_set* = ptask_set(p1) --> ptask_set(p1) = ptask_set* union ts

function Getptask_set(p1: Project): set(Task) =
  ptask_set(p1)

function Getpjob_set(p1: Project): set(Job) =
  TRUE --> pjob_set(p1) = reduce(union, {tjob_set(t1) | (t1: Task) t1 in ptask_set(p1)});
  pjob_set(p1)

function FindJob(p1: Project, jobname: string):Job =
  arb({j |(j:Job) j in pjob_set(p1) & jname(j) = jobname})



%%---------------- End of Project -----------------------------------

!! in-package("RU")
!! in-grammar('user)
#||
------------------------------------------------------------------------
-- Object Name: Qualified_For
-- Specification Date: 09/12/95
-- Filename: qualfor.re
-- Specified by: Hibdon
-- Dependencies:
--
-- Based on: Qualified_For (950515)
-- Design Transforms and Rationale:
--   Associative object design of association assignment
--   Each instance represents a worker-job link.
--
-- History:
-- 09/05/95 (Hartrum): original design.
--
------------------------------------------------------------------------
||#

%% Define class structure:
var Qualified_For: object-class subtype-of user-object
    var wqualified_for: map(Qualified_For, Worker) = {||}
    var jqualified_for: map(Qualified_For, Job) = {||}
    var expect_accuracy: map(Qualified_for, integer) = {||}
    var expect_time: map(Qualified_for, integer) = {||}

%% Define class methods:
function NewQualified_For() : Qualified_For =
  make-object('Qualified_For)

function ZapQualified_For(qf: Qualified_For) =
```

```
  erase-object(qf)

function InitQualified_For(qf: Qualified_For) =
  NIL

function ShowQualified_For(qf: Qualified_For) =
  format(TRUE, "~\\pp\\ ~%", qf)


%% The next six functions display incremental development of SetQualified_For.

%% No range checks just assign
function SetQualified_For(qf: Qualified_For, w: Worker, j: Job, ex_accuracy:
                          integer, ex_time: integer) =
  TRUE --> (wqualified_for(qf) = w & jqualified_for(qf) = j &
            expect_accuracy(qf) = ex_accuracy & expect_time(qf) = ex_time)

%% Check afsc and skill level and use worker and job attributes to establish
   qualified_fro association's expect_time attribute for best_time only.

function SetQualified_For2(qf: Qualified_For, w: Worker, j: Job) =
  wafsc(w) in jafsc_set(j) & wskill_level(w) >= jskill_level(j) -->
  (wqualified_for(qf) = w & jqualified_for(qf) = j & expect_time(qf) =
   jbest_time(j))

%% Add average_time check and assignment.
function SetQualified_For3(qf: Qualified_For, w: Worker, j: Job) =
  wafsc(w) in jafsc_set(j) & wskill_level(w) > jskill_level(j) -->
  (wqualified_for(qf) = w & jqualified_for(qf) = j & expect_time(qf) =
   jbest_time(j));
  wafsc(w) in jafsc_set(j) & wskill_level(w) = jskill_level(j) -->
  (wqualified_for(qf) = w & jqualified_for(qf) = j & expect_time(qf) =
   javerage_time(j))

%% Add worst_time check and assignment.
function SetQualified_For4(qf: Qualified_For, w: Worker, j: Job) =
  wafsc(w) in jafsc_set(j) & wskill_level(w) > jskill_level(j) -->
  (wqualified_for(qf) = w & jqualified_for(qf) = j & expect_time(qf) =
   jbest_time(j));
  wafsc(w) in jafsc_set(j) & wskill_level(w) = jskill_level(j) -->
  (wqualified_for(qf) = w & jqualified_for(qf) = j & expect_time(qf) =
   javerage_time(j));
  wafsc(w) in jafsc_set(j) & wskill_level(w) < jskill_level(j) -->
  (wqualified_for(qf) = w & jqualified_for(qf) = j & expect_time(qf) =
   jworst_time(j))

%% Add qualified_for expect_accuracy checks and assignments.
function SetQualified_For4a(qf: Qualified_For, w: Worker, j: Job) =
  wafsc(w) in jafsc_set(j) & wskill_level(w) > jskill_level(j) -->
  (wqualified_for(qf) = w & jqualified_for(qf) = j & expect_time(qf) =
   jbest_time(j) & expect_accuracy(qf) = jbest_accuracy(j));
   wafsc(w) in jafsc_set(j) & wskill_level(w) = jskill_level(j) -->
  (wqualified_for(qf) = w & jqualified_for(qf) = j & expect_time(qf) =
   javerage_time(j) & expect_accuracy(qf) = javerage_accuracy(j));
   wafsc(w) in jafsc_set(j) & wskill_level(w) < jskill_level(j) -->
  (wqualified_for(qf) = w & jqualified_for(qf) = j & expect_time(qf) =
   jbest_time(j) & expect_accuracy(qf) = jworst_accuracy(j))

%% Added axioms to allow other attributes of the worker and job to affect the
   expect_time and expect_accuracy attributes of the qualified_for association.

function SetQualified_For5(qf: Qualified_For, w: Worker, j: Job) =
% Need to ensure that accuracy <= 100 and time <= besttime
% Could easily add more axioms to affect the expect_time and
```

```
  expect_accuracy attributes

  wafsc(w) in jafsc_set(j) & wskill_level(w) > jskill_level(j) -->
 (wqualified_for(qf) = w & jqualified_for(qf) = j & expect_time(qf) =
  jbest_time(j));
  wafsc(w) in jafsc_set(j) & wskill_level(w) = jskill_level(j) -->
  (wqualified_for(qf) = w & jqualified_for(qf) = j & expect_time(qf) =
   javerage_time(j));
  wafsc(w) in jafsc_set(j) & wskill_level(w) < jskill_level(j) -->
  (wqualified_for(qf) = w & jqualified_for(qf) = j & expect_time(qf) =
   jbest_time(j));
  wafsc(w) in jafsc_set(j) & weducation(w) >= 10 & jcomplexity(j) <= 75 -->
  ((expect_time* = expect_time(qf) --> expect_time(qf) = expect_time* + 20) &
  (expect_accuracy* = expect_accuracy(qf) --> expect_accuracy(qf) =
   expect_accuracy* + 20))

function GetWQualified_For(qf: Qualified_For): Worker =
  wqualified_for(qf)

function GetJQualified_For(qf: Qualified_For): Job =
  jqualified_for(qf)

function GetExpect_Time(qf: Qualified_For): integer =
  expect_time(qf)

function GetExpect_Accuracy(qf: Qualified_For): integer =
  expect_accuracy(qf)
%%---------------- End of Qualified_For ------------------------------------

!! in-package("RU")
!! in-grammar('user)
#||
------------------------------------------------------------------------
-- Object Name: Qualified_ForSet
-- Specification Date: 09/12/95
-- Filename: qualforset.re
-- Specified by: Hibdon
--
-- Based on: Qualified_For (950515)
-- Design Transforms and Rationale:
--
-- History:
-- 09/12/95 (Hibdon): original design.
--
------------------------------------------------------------------------
||#

%% Define base sets (types):

%% Define class structure:
var Qualified_ForSet: object-class subtype-of Container

%% Define class methods:
function NewQualified_ForSet() : Qualified_ForSet =
  make-object('Qualified_ForSet)

function InitQualified_ForSet(qf: Qualified_ForSet) =
  InitContainer(qf)

function ZapQualified_ForSet(qf: Qualified_ForSet) =
  erase-object(qf)

function ShowQualified_ForSet(qf: Qualified_ForSet) =
  format(TRUE, "~\\pp\\ ~%", qf)
```

```
function ListQualified_ForSet(qf: Qualified_ForSet) =
  enumerate qset over range(contents(qf)) do
    format(TRUE, "~\\pp\\ ~%", qset)

%% Given a job and a qualified_forset, return the qualified_for association.
   Might need to return a set of these.

function GetQualified_For(qs: Qualified_ForSet, j: Job): Qualified_For =
  arb({qf | (qf: Qualified_For) (qf in contents(qs) &
  jname(GetJQualified_For(qf)) = jname(j))})

%%---------------- End of Qualified_ForSet ------------------------------------

!! in-package("RU")
!! in-grammar('user)
#||
--------------------------------------------------------------------------
-- Object Name: Qualified_On
-- Specification Date: 09/12/95
-- Filename: qualifiedon.re
-- Specified by: Hibdon
-- Dependencies:
--  Must compile and load Tool, Worker.
--
-- Based on: Qualified_On (9204XX)
-- Design Transforms and Rationale:
--   Associative object design of qualified_on
--   Each instance represents a tool-worker link.
--
-- History:
-- 09/12/95 (Hibdon): original design.
--
--------------------------------------------------------------------------
||#

%% Define class structure:
var Qualified_On: object-class subtype-of user-object
    var tqualified_on: map(Qualified_On, Tool) = {||}
    var wqualified_on: map(Qualified_On, Worker) = {||}

%% Define class methods:
function NewQualified_On() : Qualified_On =
  make-object('Qualified_On)

function ZapQualified_On(qo: Qualified_On) =
  erase-object(qo)

function InitQualified_On(qo: Qualified_On) =
  NIL

function ShowQualified_On(qo: Qualified_On) =
  format(TRUE, "~\\pp\\ ~%", qo)

function SetQualified_On(qo: Qualified_On, t1: Tool, w: Worker) =
  TRUE --> (tqualified_on(qo) = t1 & wqualified_on(qo) = w)

function SetQualified_On(qo: Qualified_On, t1: Tool, w: Worker) =
  wafsc(w) in toolafsc_set(t1) & wskill_level(w) >= toolskill_level(t1) -->
  (tqualified_on(qo) = t1 & wqualified_on(qo) = w)

function GetTQualified_On(qo: Qualified_On): Tool =
  tqualified_on(qo)
```

```
function GetWQualified_On(qo: Qualified_On): Worker =
  wqualified_on(qo)

%%---------------- End of Qualified_On ------------------------------------

!! in-package("RU")
!! in-grammar('user)
#||
------------------------------------------------------------------------
-- Object Name: Qualified_OnSet
-- Specification Date: 09/12/95
-- Filename: qualifiedonset.re
-- Specified by: Hibdon
--
-- Based on: Qualified_On (950513)
-- Design Transforms and Rationale:
--
-- History:
-- 09/12/95 (Hibdon): original design.
--
------------------------------------------------------------------------
||#

%% Define base sets (types):

%% Define class structure:
var Qualified_OnSet: object-class subtype-of Container

%% Define class methods:
function NewQualified_OnSet() : Qualified_OnSet =
  make-object('Qualified_OnSet)

function InitQualified_OnSet(qo: Qualified_OnSet) =
  InitContainer(qo)

function ZapQualified_OnSet(qo: Qualified_OnSet) =
  erase-object(qo)

function ShowQualified_OnSet(qo: Qualified_OnSet) =
  format(TRUE, "~\\pp\\ ~%", qo)

function ListQualified_OnSet(qo: Qualified_OnSet) =
  enumerate qoset over range(contents(qo)) do
    format(TRUE, "~\\pp\\ ~%", qoset)

function GetQualifiedOnSet(o1: Organization): Set(<Worker, Tool>)=
  {<t1, w> | (t1: Tool, w: Worker) t1 in contents(org-toolset(o1)) &
   w in contents(org-workforce(o1)) & wafsc(w) in toolafsc_set(t1) &
   wskill_level(w) >= toolskill_level(t1)}

%% Correctly occupied positions based on afsc only.
function GetAllQualifiedOn(o1: Organization) =
  fa(w: Worker, t1: Tool)(w in worker_set(org-workforce(o1)) &
     t1 in tool_set(org-toolset(o1)) => AddQualified_On(o1, t1, w))

function GetAllQualifiedOn(qs: Qualified_OnSet): Set(Qualified_On) =
  {qo | (qo: Qualified_On) (qo in contents(qs) & wafsc(GetWQualified_On(qo))
   in toolafsc_set(GetTQualified_On(qo)))}

function SetPossibleQualifiedOn(o1: Organization) =
  fa(w: Worker, t1: Tool)((w in contents(org-workforce(o1))) &
    (t1 in contents(org-toolset(o1))) => AddPossibleQualified_On(o1, t1, w))
```

```
%% Correctly occupied positions based on afsc and skill level.
  wafsc(GetWOccupy(occ)) & wskill_level(GetWOccupy(occ)) >=
  posskill_level(GetPOccupy(occ)))}

%%---------------- End of Qualified_OnSet ------------------------------------

!! in-package("RU")
!! in-grammar('user)
#||
--------------------------------------------------------------------------
-- Object Name: Supports
-- Specification Date: 09/12/95
-- Filename: supports.re
-- Specified by: Hibdon
-- Dependencies:
--
-- Based on: Supports (950511)
-- Design Transforms and Rationale:
--   Associative object design of association supports
--   Each instance represents a tool-job link.
--
-- History:
-- 09/12/95 (Hibdon): original design.
--
--------------------------------------------------------------------------
||#

%% Define class structure:
var Supports: object-class subtype-of user-object
    var tsupports: map(Supports, Tool) = {||}
    var jsupports: map(Supports, Job) = {||}
    var suptime: map(Supports, integer) = {||}
    var percentage: map(Supports, integer) = {||}

%% Define class methods:
function NewSupports() : Supports =
  make-object('Supports)

function ZapSupports(s: Supports) =
  erase-object(s)

function InitSupports(s: Supports) =
  NIL

function ShowSupports(s: Supports) =
  format(TRUE, "~\\pp\\ ~%", s)

%% Assumes that tool supports job.
function SetSupports(s: Supports, t1: Tool, j: Job, time1: integer,
                     percent1: integer) =
  TRUE --> (tsupports(s) = t1 & jsupports(s) = j & time(s) = time1 &
            percentage(s) = percent1)

%% Checks to ensure that the tool can support the job through the intersection
   of their respective afsc sets.

function SetSupports(s: Supports, t1: Tool, j: Job, time1: integer,
                     percent1: integer) =
  toolafsc_set(t1) intersect jafsc_set(j1) ~= {} --> (tsupports(s) = t1 &
  jsupports(s) = j & suptime(s) = time1 & percentage(s) = percent1)

function GetTSupports(s: Supports): Tool =
  tsupports(s)
```

```
function GetSupportsTime(s: Supports): Integer =
  suptime(s)

function GetSupportsPercent(s: Supports): Integer =
  percentage(s)

function GetJSupports(s: Supports): Job =
  jsupports(s)

%%---------------- End of Supports ------------------------------------

!! in-package("RU")
!! in-grammar('user)
#||
--------------------------------------------------------------------------
-- Object Name: SupportsSet
-- Specification Date: 09/12/95
-- Filename: supportsset.re
-- Specified by: Hibdon
--
-- Based on: Supports (950511)
-- Design Transforms and Rationale:
--
-- History:
-- 09/12/95 (Hibdon): original design.
--
--------------------------------------------------------------------------
||#

%% Define base sets (types):

%% Define class structure:
var SupportsSet: object-class subtype-of Container

%% Define class methods:
function NewSupportsSet() : SupportsSet =
  make-object('SupportsSet)

function InitSupportsSet(s: SupportsSet) =
  InitContainer(s)

function ZapSupportsSet(s: SupportsSet) =
  erase-object(s)

function ShowSupportsSet(s: SupportsSet) =
  format(TRUE, "~\\pp\\ ~%", s)

function ListSupportsSet(s: SupportsSet) =
  enumerate sset over range(contents(s)) do
    format(TRUE, "~\\pp\\ ~%", sset)

%% Given a job and a supportsset, return the supports association that
   corresponds.  Might need to return a set.

function GetSupports(ss: SupportsSet, j: Job): Supports =
  arb({s | (s: Supports) (s in contents(ss) &
       jname(GetJSupports(s)) = jname(j))})


%%---------------- End of SupportsSet ---------------------------------


!! in-package("RU")
```

```
!! in-grammar('user)
#||
------------------------------------------------------------------------
-- Object Name: Task
-- Specification Date: 08/28/95
-- Filename: task.re
-- Specified by: Hibdon
--
-- Based on: Task (950502)
-- Design Transforms and Rationale:
--
-- History:
-- 08/28/95 (Hibdon): original design.
--
------------------------------------------------------------------------
||#

%% Define base sets (types):
constant JOB_STATUS : set(symbol) = { 'ready, 'waiting, 'in-process, 'finished}
var temptime: integer = 0
var tempjob: Job = NewJob()

%% Define class structure:
var Task: object-class subtype-of user-object
    var tname: map(Task, string) = {||}
    var tpriority: map(Task, integer) = {||}
    var tstatus: map(Task, symbol) = {||}
    var tinputs: map(Task, set(symbol)) = {||}
    var toutputs: map(Task, set(symbol)) = {||}
    var ttime_so_far: map(Task, real) = {||}
    var tpercent_complete: map(Task, real) = {||}
    var tnumber_of_jobs: map(Task, integer) = {||}
    var tjobs_complete: map(Task, integer) = {||}
    var tstart_time: map(Task, integer) = {||}
    var tend_time: map(Task, integer) = {||}
    var twall_time: map(Task, integer) = {||}
    var ttotal_time: map(Task, real) = {||}
    var taccuracy: map(Task, integer) = {||}
    var tjob_set: map(Task, set(Job)) = {||}

%% Define class methods:
function NewTask() : Task =
  make-object('Task)

function ZapTask(t1: Task) =
  erase-object(t1)

function ShowTask(t1: Task) =
  format(TRUE, " Task name: ~S~% Task priority: ~S~% status: ~s~% inputs: ~s~%
   outputs: ~s~% time_so_far: ~d~% percent_complete: ~d~% number_of_jobs: ~d~%
   jobs_complete: ~d~% start_time: ~s~% end_time: ~s~% wall_time: ~d~%
   total_time: ~d~% accuracy: ~s~% job_set: ~s~%~%~%", tname(t1), tpriority(t1),
   tstatus(t1), tinputs(t1), toutputs(t1), ttime_so_far(t1),
   tpercent_complete(t1), tnumber_of_jobs(t1), tjobs_complete(t1),
   tstart_time(t1), tend_time(t1), twall_time(t1), ttotal_time(t1),
   taccuracy(t1), tjob_set(t1))

function Settname(t1: Task, n: string) =
  TRUE --> tname(t1) = n

function Gettname(t1: Task): string =
  tname(t1)

%% Checks boundary ranges
```

```
function Settpriority(t1: Task, p: integer) =
  (p >= 0) & (p <= 100) --> tpriority(t1) = p;
  (p < 0) OR (p > 100) --> (tpriority(t1) = undefined)


function Gettpriority(t1: Task): integer =
  tpriority(t1)

%% Manually set a task's status
function Settstatus(t1: Task) =
  s ~in JOB_STATUS --> tstatus(t1) <- undefined;
  s in JOB_STATUS --> tstatus(t1) = s

function Settstatus(t1: Task) =
  fa(j)((j in tjob_set(t1) & jstatus(j) = 'finished) =>
    tstatus(t1) = 'finished);
  fa(j)((j in tjob_set(t1) & jstatus(j) = 'waiting) =>
    tstatus(t1) = 'waiting);
  fa(j)((j in tjob_set(t1) & jstatus(j) = 'ready) =>
    tstatus(t1) = 'ready);
  ex(j)(j in tjob_set(t1) & jstatus(j) = 'in-process) =>
    tstatus(t1) <- 'in-process

%% Refine cannot determine how to do the following:
%  (s ~= 'finished) --> (pc < 100)
%  (ex j in tjob_set(t1))(jstatus(j) ~= 'finished) =>
%   tstatus(t1) ~= 'finished


%% Derive the status of a task based on the jobs in its jobset.
function Gettstatus(t1: Task): symbol =
  fa(j)((j in tjob_set(t1) & jstatus(j) = 'finished) =>
   (tstatus(t1) = 'finished & ttime_so_far(t1) = Getttotal_time(t1)));
  fa(j)((j in tjob_set(t1) & jstatus(j) = 'waiting) =>
    tstatus(t1) = 'waiting);
  fa(j)((j in tjob_set(t1) & jstatus(j) = 'ready) =>
    tstatus(t1) = 'ready);
  ex(j)(j in tjob_set(t1) & jstatus(j) = 'in-process) =>
    tstatus(t1) = 'in-process;
  tstatus(t1)

%% Manually set inputs
function Settinputs(t1: Task, i: set(symbol) ) =
  TRUE --> tinputs(t1) = i

%% Determines the inputs and outputs of a task based on the jobs in its jobset.

function SettIO(t1: Task) =
  TRUE --> tinputs(t1) =
    arb(setdiff({jinputs(j1) | (j1: Job) j1 in tjob_set(t1)},
    ({jinputs(j2) | (j2: Job) j2 in tjob_set(t1)} intersect
    {joutputs(j3) | (j3: Job) j3 in tjob_set(t1)}));

  TRUE --> toutputs(t1) =
    arb(setdiff({joutputs(j1) | (j1: Job) j1 in tjob_set(t1)},
    ({jinputs(j2) | (j2: Job) j2 in tjob_set(t1)} intersect
    {joutputs(j3) | (j3: Job) j3 in tjob_set(t1)}))

function Gettinputs(t1: Task): set(symbol) =
  tinputs(t1)

%% Manually set outputs
function Settoutputs(t1: Task, o: set(symbol)) =
  TRUE --> toutputs(t1) = o
```

```
function Gettoutputs(t1: Task): set(symbol) =
  toutputs(t1)

%% Time so far of a task is determined by the amount of time spent so far on the jobs in its jobset.
% Need to add tstatus(t1) = 'ready and tstatus(t1) = waiting

  tstatus(t1) = 'finished --> ttime_so_far(t1) <- ttotal_time(t1);
  tstatus(t1) = 'in-process --> ttime_so_far(t1) <-
   jstart_time(arb({j1 | (j1: Job) j1 in tjob_set(t1) & jstatus(j1) =
   'in-process})) * jpercent_complete(arb({j1 | (j1: Job) j1 in tjob_set(t1) &
   jstatus(j1) = 'in-process}))

function Getttime_so_far(t1: Task): real =
  ttime_so_far(t1)

%% This function calcualtes percent complete based on the number of jobs
   complete divided by the total number of jobs.  Not very realistic.
%function Settpercent_complete(t1: Task) =
%  size({j1 | (j1: Job) j1 in tjob_set(t1) & jstatus(j1) = 'finished})/
%   size(tjob_set(t1)) >= 0.0 & size({j1 | (j1: Job) j1 in tjob_set(t1) &
%   jstatus(j1) = 'finished})/size(tjob_set(t1)) <= 1.0 -->
%   tpercent_complete(t1) = size({j1 | (j1: Job) j1 in tjob_set(t1) &
%   jstatus(j1) = 'finished})/size(tjob_set(t1))

%% Based on the amount of time spent so far divided by the total amount of time
   for all jobs.

function Settpercent_complete(t1: Task) =
  TRUE --> tpercent_complete(t1) <-
   ((reduce(+,[jtime(j1) | (j1: Job) j1 in tjob_set(t1) &
   jstatus(j1) = 'finished])))/
   (reduce(+,[jtime(j1) | (j1: Job) j1 in tjob_set(t1)])))

function Gettpercent_complete(t1: Task): real =
  tpercent_complete(t1)

function Settnumber_of_jobs(t1: Task) =
  TRUE --> tnumber_of_jobs(t1) <- size(tjob_set(t1))

function Gettnumber_of_jobs(t1: Task): integer =
  tnumber_of_jobs(t1)

function Settjobs_complete(t1: Task) =

  TRUE --> tjobs_complete(t1) =
    size({j1|(j1: Job) j1 in tjob_set(t1) & jstatus(j1) = 'finished});
  tnumber_of_jobs(t1) = size({j1 | (j1: Job) j1 in tjob_set(t1) &
   jstatus(j1) = 'finished}) --> tjobs_complete(t1) =
   size({j1 | (j1: Job) j1 in tjob_set(t1) & jstatus(j1) = 'finished});
   tjobs_complete(t1) = tnumber_of_jobs(t1) -->
   tpercent_complete(t1) = 1.0 & tstatus(t1) = 'finished

function Gettjobs_complete(t1: Task): integer =
  tjobs_complete(t1)

function SettStart_Time(t1: Task) =
  TRUE --> tstart_time(t1) <-
   jstart_time(arb({j1 | (j1: Job) j1 in tjob_set(t1) &
   (fa(j2) (j2 in tjob_set(t1) => jstart_time(j1) <= jstart_time(j2))())))

function Gettstart_time(t1: Task): integer =
  tstart_time(t1)
```

```
function Settend_time(t1: Task) =
  TRUE --> tend_time(t1) <-
   jend_time( arb({j1 | (j1: Job) j1 in tjob_set(t1) &
   (fa(j2) (j2 in tjob_set(t1) => jend_time(j1) >= jend_time(j2)))}))

function Gettend_time(t1: Task): integer =
  tend_time(t1)

function Settwall_time(t1: Task) =
  TRUE --> twall_time(t1) = tend_time(t1) - tstart_time(t1)

function Gettwall_time(t1: Task): integer =
  twall_time(t1)

function Setttotal_time(t1: Task) =
  TRUE --> ttotal_time(t1) <-
   integer-to-real(reduce(+, [jtime(j1) | (j1: Job) j1 in tjob_set(t1)]))

function Getttotal_time(t1: Task): real =
  ttotal_time(t1)

function Settaccuracy(t1: Task) =
  TRUE --> taccuracy(t1) <-
  real-to-nearest-integer(reduce(+,[jaccuracy(j1)|(j1: Job) j1 in
   tjob_set(t1)])/size(tjob_set(t1)))

function Gettaccuracy(t1: Task): integer =
  taccuracy(t1)

function SettJob_Set(t1: Task, js: set(Job)) =
  tjob_set* = tjob_set(t1) --> tjob_set(t1) = tjob_set* union js

function GettJob_Set(t1: Task): set(Job) =
  tjob_set(t1)

function FindJob(t1: Task, jobname: string):Job =
  arb({j |(j:Job) j in tjob_set(t1) & jname(j) = jobname})


%%---------------- End of Task -----------------------------------

!! in-package("RU")
!! in-grammar('user)
#||
--------------------------------------------------------------------------
-- Object Name: Tool
-- Specification Date: 08/28/95
-- Filename: tool.re
-- Specified by: Hibdon
--
-- Based on: Tool (950504)
-- Design Transforms and Rationale:
--
-- History:
-- 08/28/95 (Hibdon): original design.
--
--------------------------------------------------------------------------
||#

%% Define base sets (types):
constant AFSCS : set(string) = { "33S3B", "33S3C", "4921"}
constant STATUS_TYPE : set(symbol) = { 'go, 'nogo}

%% Define class structure:
```

```
var Tool: object-class subtype-of user-object
    var toolname: map(Tool, string) = {||}
    var toolstatus: map(Tool, symbol) = {||}
    var toolavailability: map(Tool, boolean) = {||}
    var toolafsc_set: map(Tool, set(string)) = {||}
    var toolskill_level: map(Tool, integer) = {||}

%% Define class methods:
function NewTool() : Tool =
  make-object('Tool)

function ZapTool(t1: Tool) =
  erase-object(t1)

function ShowTool(t1: Tool) =
 format(TRUE, " Tool name: ~S~% Status: ~s~% Availability: ~b~% Afsc_set: ~S~%
   skill_level: ~d~%~%~%~%", toolname(t1), toolstatus(t1), toolavailability(t1),
   toolafsc_set(t1), toolskill_level(t1))

function Settoolname(t1: Tool, n: string) =
  TRUE --> toolname(t1) = n

function Gettoolname(t1: Tool): string =
  toolname(t1)

%% Checks for valid afsc.
function Settoolstatus(t1: Tool, s: symbol) =
  s in STATUS_TYPE --> toolstatus(t1) = s;
  s ~in STATUS_TYPE --> toolstatus(t1) = undefined

function Gettoolstatus(t1: Tool): symbol =
  toolstatus(t1)

function Settoolavailability(t1: Tool, a: boolean) =
  TRUE --> toolavailability(t1) = a

function Gettoolavailability(t1: Tool): boolean =
  toolavailability(t1)

function Settoolafsc_set(t1: Tool, aset: set(string) ) =
  aset subset AFSCS --> toolafsc_set(t1) = aset

function Gettoolafsc_set(t1: Tool): set(string) =
  toolafsc_set(t1)

function Settoolskill_level(t1: Tool, s: integer) =
  (s >= 1 & s <= 9) --> toolskill_level(t1) = s

function Gettoolskill_level(t1: Tool): integer =
  toolskill_level(t1)

%%---------------- End of Tool -----------------------------------

!! in-package("RU")
!! in-grammar('user)
#||
------------------------------------------------------------------------
-- Object Name: ToolSet
-- Specification Date: 09/11/95
-- Filename: toolset.re
-- Specified by: Hibdon
--
-- Based on: ToolSet (950508)
-- Design Transforms and Rationale:
```

```
--
-- History:
-- 09/11/95 (Hibdon): original design.
--
-------------------------------------------------------------------------
||#

%% Define base sets (types):

%% Define class structure:
var ToolSet: object-class subtype-of Container
    var tool_readiness: map(ToolSet, real) = {||}
    var tool_set: map(ToolSet, set(Tool)) = {||}


%% Define class methods:
function NewToolSet() : ToolSet =
  make-object('ToolSet)

function InitToolSet(ts: ToolSet) =
  InitContainer(ts)

function ZapToolSet(ts1: ToolSet) =
  erase-object(ts1)

function ShowToolSet(ts1: ToolSet) =
 format(TRUE, "ToolSet~%  tool_readiness: ~d~% Tools: ~S~%~%~%",
   tool_readiness(ts1), tool_set(ts1))

%% Determine the readiness of the toolset based on the number of tools in the
   go status divided by the number of tools in the toolset.

function Settool_readiness(ts1: ToolSet, o1: Organization) =
  TRUE --> tool_readiness(ts1) = size({t1 | (t1: Tool) t1 in
           contents(org-toolset(o1)) & toolstatus(t1) = 'go})/
           size({t1 | (t1: Tool) t1 in contents(org-toolset(o1))})

function ListToolSet(ts: ToolSet) =
  enumerate t1 over range(contents(ts)) do
    format(TRUE, "~\\pp\\ ~%", GetToolName(t1))

function Gettool_readiness(ts1: ToolSet): real =
  tool_readiness(ts1)

function SetToolSet(ts1: ToolSet, ts: set(Tool)) =
  tool_set* = tool_set(ts1) --> tool_set(ts1) = tool_set* union ts

%%---------------- End of ToolSet -----------------------------------

!! in-package("RU")
!! in-grammar('user)
#||
-------------------------------------------------------------------------
-- Object Name: Using
-- Specification Date: 09/12/95
-- Filename: using.re
-- Specified by: Hibdon
-- Dependencies:
--
-- Based on: Using (950512)
-- Design Transforms and Rationale:
--   Associative object design of using
--   Each instance represents a tool-worker link.
--
```

```
-- History:
-- 09/12/95 (Hibdon): original design.
--
------------------------------------------------------------------------
||#

%% Define class structure:
var Using: object-class subtype-of user-object
    var tusing: map(Using, Tool) = {||}
    var wusing: map(Using, Worker) = {||}

%% Define class methods:
function NewUsing() : Using =
  make-object('Using)

function ZapUsing(u: Using) =
  erase-object(u)

function InitUsing(u: Using) =
  NIL

function ShowUsing(u: Using) =
  format(TRUE, "~\\pp\\ ~%", u)

function SetUsing(u: Using, t1: Tool, w: Worker) =
  TRUE --> (tusing(u) = t1 & wusing(u) = w)

function GetTUsing(u: Using): Tool =
  tusing(u)

function GetWUsing(u: Using): Worker =
  wusing(u)

%%---------------- End of Using -----------------------------------

!! in-package("RU")
!! in-grammar('user)
#||
------------------------------------------------------------------------
-- Object Name: UsingSet
-- Specification Date: 09/12/95
-- Filename: usingset.re
-- Specified by: Hibdon
--
-- Based on: Using (950512)
-- Design Transforms and Rationale:
--
-- History:
-- 09/12/95 (Hibdon): original design.
--
------------------------------------------------------------------------
||#

%% Define base sets (types):

%% Define class structure:
var UsingSet: object-class subtype-of Container

%% Define class methods:
function NewUsingSet() : UsingSet =
  make-object('UsingSet)

function InitUsingSet(us: UsingSet) =
  InitContainer(us)
```

```
function ZapUsingSet(us: UsingSet) =
  erase-object(us)

function ShowUsingSet(us: UsingSet) =
  format(TRUE, "~\\pp\\ ~%", us)

function ListUsingSet(us: UsingSet) =
  enumerate uset over range(contents(us)) do
    format(TRUE, "~\\pp\\ ~%", uset)

%%---------------- End of UsingSet -----------------------------------

!! in-package("RU")
!! in-grammar('user)
#||
------------------------------------------------------------------------
-- Object Name: Worker
-- Specification Date: 08/28/95
-- Filename: worker.re
-- Specified by: Hibdon
--
-- Based on: Worker (950505)
-- Design Transforms and Rationale:
--
-- History:
-- 08/28/95 (Hibdon): original design.
--
------------------------------------------------------------------------
||#

%% Define base sets (types):
constant AFSCS : set(string) = { "33S3B", "33S3C", "4921"}

%% Define class structure:
var Worker: object-class subtype-of user-object
    var wname: map(Worker, string) = {||}
    var wafsc: map(Worker, string) = {||}
    var wskill_level: map(Worker, integer) = {||}
    var wavailability: map(Worker, boolean) = {||}
    var wexperience: map(Worker, integer) = {||}
    var weducation: map(Worker, integer) = {||}
    var waptitude: map(Worker, integer) = {||}

%% Define class methods:
function NewWorker() : Worker =
  make-object('Worker)

function ZapWorker(w: Worker) =
  erase-object(w)

function ShowWorker(w: Worker) =
 format(TRUE, " Worker name: ~S~% Afsc: ~S~% skill_level: ~d~% Availability:
        ~b~% Experience: ~d~% Education: ~d~% Aptitude: ~d~%~%~%", wname(w),
        wafsc(w), wskill_level(w), wavailability(w), wexperience(w),
        weducation(w), waptitude(w))

function Setwname(w: Worker, n: string) =
  TRUE --> wname(w) = n

function Getwname(w: Worker): string =
  wname(w)

%% Checks for valid afsc
```

```
function Setwafsc(w: Worker, a: string) =
  a in AFSCS --> wafsc(w) = a;
  a ~in AFSCS --> wafsc(w) = undefined

function Getwafsc(w: Worker): string =
  wafsc(w)

%% Checks boundary ranges.
function Setwskill_level(w: Worker, s: integer) =
  (s >= 1 & s <= 9) --> wskill_level(w) = s

function Getwskill_level(w: Worker): integer =
  wskill_level(w)

function Setwavailability(w: Worker, a: boolean) =
  TRUE --> wavailability(w) = a

function Getwavailability(w: Worker): boolean =
 wavailability(w)

%% Checks boundary ranges.
function Setwexperience(w: Worker, e: integer ) =
  (e >= 0 & e <= 30) --> wexperience(w) = e

function Getwexperience(w: Worker): integer =
  wexperience(w)

%% Checks boundary ranges.
function Setweducation(w: Worker, e: integer) =
  (e >= 0 & e <= 18) --> weducation(w) = e

function Getweducation(w: Worker): integer =
  weducation(w)

%% Checks boundary ranges.
function Setwaptitude(w: Worker, a: integer) =
  (a >= 0 & a <= 100) --> waptitude(w) = a

function Getwaptitude(w: Worker): integer =
  waptitude(w)

%%---------------- End of Worker ------------------------------------

!! in-package("RU")
!! in-grammar('user)
#||
------------------------------------------------------------------------
-- Object Name: Workforce
-- Specification Date: 09/11/95
-- Filename: workforce1.re
-- Specified by: Hibdon
--
-- Based on: Worker (950509)
-- Design Transforms and Rationale:
--
-- History:
-- 08/28/95 (Hibdon): original design.
--
------------------------------------------------------------------------
||#

%% Define base sets (types):

%% Define class structure:
```

```
var Workforce: object-class subtype-of Container
    var workforceuse: map(Workforce, string) = {||}
    var force_readiness: map(Workforce, real) = {||}
    var worker_set: map(Workforce, set(Worker)) = {||}

%% Define class methods:
function NewWorkforce() : Workforce =
  make-object('Workforce)

function InitWorkforce(wf: Workforce) =
  InitContainer(wf)

function ZapWorkforce(wf: Workforce) =
  erase-object(wf)

function ShowWorkforce(wf: Workforce) =
 format(TRUE, "Workforce~% force_readiness: ~d~% Workers: ~S~%~%~%",
        force_readiness(wf), worker_set(wf))

%% Force readiness is the number of correctly occupied positions in the manning
   plan divided by the total number of positions in the manning plan.

function Setforce_readiness(wf: Workforce, o1: Organization, mp: ManningPlan) =
  TRUE --> force_readiness(wf) =
   size(GetCorrectlyOccupiedPositions2(org-occupy(o1)))/size(positions(mp))

function Getforce_readiness(wf: Workforce): real =
  force_readiness(wf)

function Setworkerset(wf: Workforce, ws: set(Worker)) =
  worker_set* = worker_set(wf) --> worker_set(wf) = worker_set* union ws

function SetWorkforceUse(wf: Workforce, s_use: string) =
  TRUE --> workforceuse(wf) = s_use

function GetWorkforceUse(wf: Workforce): string =
  workforceuse(wf)

function GetAllWorkers(wf: Workforce): set(Worker) =
  range(contents(wf))

function ListWorkforce(wf: Workforce) =
  enumerate w1 over range(contents(wf)) do
    format(TRUE, "~\\pp\\ ~%", GetWName(w1))

function GetWorker(wf: Workforce, tempname: string): Worker =
  arb({x |(x:Worker) x in range(contents(wf)) & wname(x) = tempname})

%%--------------- End of Workforce -----------------------------------
```

Bibliography

1. "The Handbook of Artificial Intelligence," *IV* (1989).
2. Balzer, Robert, et al. "Software Techmology in the 1990's: Using a New Paradigm," *IEEE Computer*, 16–22 (November 1983).
3. Bandinelli, Sergio, et al. "Modeling and Improving an Industrial Software Process," *IEEE Transactions on Software Engineering*, *21*(5):440–453 (May 1995).
4. Bjork, Jim. "Screening for Success," *Texas Banking*, *82*(11) (November 1993).
5. Blaine, Lee, et al. *Specware^{TM} User Manual*, October 1994. Specware^{TM} Version Core4.
6. Borman, W., et al. *Productive Capacity: The Concept, Research, and Applications*. Technical Report AL/HR-TP-1994-0021, Brooks AFB, TX 78235-5601: Human Resources Directorate, August 1994.
7. Choi, Dong, et al. *Systems Modeling*. Technical Report NAVSWC TR 91-592, Silver Spring, Maryland 20903-5000: Naval Surface Warfare Center, February 1992.
8. Faneuff, Robert S. *Predicting the Productive Capacity of Air Force Aerospace Ground Equipment Personnel Using Aptitude and Experience Measures*. MS thesis, AFIT/GOR/ENS/93M-05, 1993.
9. Hall, Anthony. "Seven Myths of Formal Methods," *IEEE Software*, 11–19 (September 1990).
10. Hartrum, Thomas C., "CSCE 594 - Object-Oriented Design and Analysis." Class Notes, 1993.
11. Hartrum, Thomas C., et al. "Modeling Wing Level Operations Using Formal Object Models." *National Aerospace and Electronics Conference Proceedings*. NAECON, May 1995.
12. Hunt, Captain Robert J. *Modeling Operational Task Assignment in Air Force Wing Command and Control*. AD-A289319, Graduate School of Engineering, Air Force Institute of Technology (AU), 1994.
13. Huo, Y. Paul and Jack Kearns. "Optimizing the Job-person Match with Computerized Human Resource Information Systems," *Personnel Review*, *21*(2):3–18 (1992).
14. Kavanaugh, Michael J., et al. *Job Performance Measurement in the Military: A Classification Scheme, Literature Review, and Directions for Research*. Technical Report AFHRL-TR-87-15, AFSC Brooks AFB, TX 78235-5601: Air Force Human Resources Laboratory, September 1987.
15. Kralji, Mary M., et al. *Definition and Measures of Induvidual and Unit Readiness and Family Phenomena Affecting it*. Technical Report 91-32, Research Triangle Park, NC: Research Triangle Institute, February 1991.
16. Kusiak, Andrew, et al. "Reengineering of Design and Manufacturing Processes," *Computers and Industrial Engineering*, *26*(3):521–536 (July 1994).
17. 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.

18. Leonhard, Corey A. and J Steve Davis. "Job-Shop Development Model: A Case Study," *IEEE Software*, *12*(2):86–92 (March 1995).

19. Lichter, Horst, et al. "Prototyping in Industrial Software Projects-Bridging the Gap Between Theory and Practice," *IEEE Transactions on Software Engineering*, *20*(11):825–832 (November 1994).

20. 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.

21. Lung, C., et al. "Computer Simulation Software Reuse by Generic/Specific Domain Modeling Approach," *International Journal of Software Engineering and Knowledge Engineering*, *4*(1):81–102 (mar 1994).

22. McCain, Ron. "Reusable Software Component Construction: A Product-Oriented Paradigm," *AIAA/ACM/NASA/IEEE Computers in Aerospace V Conference*, 125–135 (October 1985).

23. McDowell, Phillip W. and David W. Morgan. *Business Process Improvement Applied to Written Temporary Duty Travel Orders within the United States Air Force*. MS thesis, Graduate School of Logistics and Acquisition Management, Air Force Institute of Technology (AU), December 1993.

24. Place, Patrick R. H. and William G. Wood. *Formal Development of Ada Programs Using Z and Anna: A Case Study*. Technical Report SEI-91-TR-1, Carnegie Mellon University, Pittsburgh, Pennsylvania 15213: Software Engineering Instute, February 1991.

25. 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.

26. Reasoning Systems Inc., 3260 Hillview Avenue, Palo Alto, CA. *Refine User's Guide*, May 1990. Version 3.0.

27. Rumbaugh, J., et al. *Object-Oriented Modeling and Design*. Prentice-Hall, Inc., 1991.

28. Sarchet, Captain Michael D. *Modeling Workload Effectiveness and Efficiency of Air Force Wing Command and Control*. AD-A289217, Graduate School of Engineering, Air Force Institute of Technology (AU), 1994.

29. Spivey, J.M. *The Z Notation*. London: Prentice Hall International, 1989.

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

31. Wabiszewski, Kathleen May. *Unification of Larch and Z-Based Object Models to Support Algebraically-Based Design Refinement: The Z Perspective*. AD-A289234, Graduate School of Engineering, Air Force Institute of Technology, December 1994.

32. 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).

33. Welgan, Robert L. *Domain Analysis and Modeling of a Model-Based Software Executive*. AD-A274087, Graduate School of Engineering, Air Force Institute of Technology,, Wright-Patterson AFB, Ohio, December 1993.

Vita

Captain Vincent S. Hibdon [redacted], and graduated from East Central High School, Tulsa, Oklahoma, in May 1982. On July 7, 1987, Vince enlisted in the US Air Force. Upon graduation from Basic Military Training School, Vince was sent to Keesler AFB, Mississippi, where he completed Avionics Communications Systems training and was stationed for his first assignment as a Communication-Navigation Systems Maintenance technician working on C-130 aircraft. While stationed at Keesler AFB, Vince was accepted into the Airman's Education and Commissioning Program to complete his Bachelor of Science in Computer Science at the Univeristy of Missouri-Rolla. Graduation from UMR led Vince to Officer Training School. Vince became Second Lieutenant Hibdon on September 25, 1991 and returned to Keesler AFB to complete Basic Communication-Computer Officer Training.

Upon graduation from the Communication-Computer Systems Officer Core course at Keesler AFB, Mississippi, in March, 1992, Lt Hibdon was assigned to the 552nd Air Control Wing. His duties there included the support and operational testing of Airborne Warning and Control Systems (AWACS) software. In May 1994, Lt Hibdon entered the Air Force Institute of Technology (AFIT) at Wright-Patterson AFB, Ohio, to pursue a Master of Science degree with a Computer Science major and concentration in Software Engineering. Upon graduation from AFIT in December 1995, Captain Hibdon was re-assigned to Global Weather Central, Offutt AFB, Nebraska.

Permanent address: [redacted]

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 1995	Technical Report; Thesis

4. TITLE AND SUBTITLE
AN OBJECT-ORIENTED, FORMAL METHODS APPROACH TO ORGANIZATIONAL PROCESS MODELING

5. FUNDING NUMBERS

6. AUTHOR(S)
Captain Vincent S. Hibdon

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/95D-06

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
HQ PACAF
25 E Street, Suite C-206
Hickam AFB, HI 96853

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; Distribution Unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This document presents a methodology for developing an organizational process model which is based on the principles of object-oriented design and formal software engineering methods. The methodology begins with the development of an object-oriented Rumbaugh model. The Rumbaugh model is then formally specified in Z (Zed) schemas. Finally, the Z specifications are translated into an executable model in the Software Refinery Environment^{TM}. This model is described based on the AF wing domain and developed in this domain. The proposed methodology is then shown to produce a very general model which is extendable across almost any domain. The proposed methodology is also shown to be very general and tailorable for specific domain applications.

14. SUBJECT TERMS
Object-Oriented, Formal Methods, Process Model

15. NUMBER OF PAGES
157

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

GENERAL INSTRUCTIONS FOR COMPLETING SF 298

The Report Documentation Page (RDP) is used in announcing and cataloging reports. It is important that this information be consistent with the rest of the report, particularly the cover and title page. Instructions for filling in each block of the form follow. It is important to ***stay within the lines*** to meet ***optical scanning requirements***.

Block 1. Agency Use Only *(Leave blank)*.

Block 2. Report Date. Full publication date including day, month, and year, if available (e.g. 1 Jan 88). Must cite at least the year.

Block 3. Type of Report and Dates Covered. State whether report is interim, final, etc. If applicable, enter inclusive report dates (e.g. 10 Jun 87 - 30 Jun 88).

Block 4. Title and Subtitle. A title is taken from the part of the report that provides the most meaningful and complete information. When a report is prepared in more than one volume, repeat the primary title, add volume number, and include subtitle for the specific volume. On classified documents enter the title classification in parentheses.

Block 5. Funding Numbers. To include contract and grant numbers; may include program element number(s), project number(s), task number(s), and work unit number(s). Use the following labels:

C	- Contract	**PR**	- Project
G	- Grant	**TA**	- Task
PE	- Program Element	**WU**	- Work Unit Accession No.

Block 6. Author(s). Name(s) of person(s) responsible for writing the report, performing the research, or credited with the content of the report. If editor or compiler, this should follow the name(s).

Block 7. Performing Organization Name(s) and Address(es). Self-explanatory.

Block 8. Performing Organization Report Number. Enter the unique alphanumeric report number(s) assigned by the organization performing the report.

Block 9. Sponsoring/Monitoring Agency Name(s) and Address(es). Self-explanatory.

Block 10. Sponsoring/Monitoring Agency Report Number. *(If known)*

Block 11. Supplementary Notes. Enter information not included elsewhere such as: Prepared in cooperation with...; Trans. of...; To be published in.... When a report is revised, include a statement whether the new report supersedes or supplements the older report.

Block 12a. Distribution/Availability Statement. Denotes public availability or limitations. Cite any availability to the public. Enter additional limitations or special markings in all capitals (e.g. NOFORN, REL, ITAR).

- **DOD** - See DoDD 5230.24, "Distribution Statements on Technical Documents."
- **DOE** - See authorities.
- **NASA** - See Handbook NHB 2200.2.
- **NTIS** - Leave blank.

Block 12b. Distribution Code.

- **DOD** - Leave blank.
- **DOE** - Enter DOE distribution categories from the Standard Distribution for Unclassified Scientific and Technical Reports.
- **NASA** - Leave blank.
- **NTIS** - Leave blank.

Block 13. Abstract. Include a brief *(Maximum 200 words)* factual summary of the most significant information contained in the report.

Block 14. Subject Terms. Keywords or phrases identifying major subjects in the report.

Block 15. Number of Pages. Enter the total number of pages.

Block 16. Price Code. Enter appropriate price code *(NTIS only)*.

Blocks 17. - 19. Security Classifications. Self-explanatory. Enter U.S. Security Classification in accordance with U.S. Security Regulations (i.e., UNCLASSIFIED). If form contains classified information, stamp classification on the top and bottom of the page.

Block 20. Limitation of Abstract. This block must be completed to assign a limitation to the abstract. Enter either UL (unlimited) or SAR (same as report). An entry in this block is necessary if the abstract is to be limited. If blank, the abstract is assumed to be unlimited.