AN AIR FORCE ORGANIZATION PROCESS MODEL USING FORMAL SOFTWARE ENGINEERING TECHNIQUES

Vincent S. Hibdon and Thomas C. Hartrum

Air Force Institute of Technology
Department of Electrical and Computer Engineering
Wright-Patterson Air Force Base, OH 45433-7765

Abstract

This paper presents a methodology for developing an organizational process model based on the principles of object-oriented design and formal software engineering methods. The methodology begins with the development of an object-oriented domain model which consists of an informal Rumbaugh model [14] formally specified in the Z [Zed] formal specification language. The Z specifications are then translated into an executable model in the Software Refinery EnvironmentTM. The Refine environment provides an executable language that allows behavior to be expressed declaratively, giving the developer an executable specification. This results in an observable domain model with attributes to describe organizational level metrics. While the model described in this paper is based on the Air Force wing C31 domain, both the methodology and the resulting model are shown to be very general and tailorable for other domain applications. The use of formal methods and an object-oriented approach leads to a more mathematically verifiable and more easily maintained executable process model.

1 Introduction

Process modeling is playing an increasingly important role in the Air Force in many diverse areas. One example is the recent interest in business process reengineering (BPR). The fundamental goal of business process reengineering is to identify and to correct fundamental deficiencies in the business process [11]. Once the deficiencies have been corrected, automating appropriate tasks will further improve the process. This is approached by developing a model of the business process and using that model to properly redefine the way business is conducted.

Another area of current interest is that of total quality management. Here the goal is to improve quality by analyzing and improving the way a task is accomplished. A critical step in this approach is to develop a model of how the process is currently carried out, along with a set of metrics for measuring how well the job is done.

In a more technical vein, software development in the Air Force is being focused along the lines of the Software Engineering Institute's Capability Maturity Model [12]. Here software developers match their development of software to one of five levels, depending on how well the software development process is modeled and controlled. The emphasis is on improving software quality by improving the development process.

The initial goal of the research discussed in this paper was to develop a formal domain model of the mission of an Air Force fighter wing and to demonstrate that such a model can be used to predict the effect that automation would have on unit readiness and unit effectiveness. This paper describes the application of Knowledge Based Software Engineering (KBSE) techniques to model a unit's mission (process) performance under differing personnel assignments and levels of automation [5]. The approach consists of formally modeling both the unit mission (behavior) and resources, including existing and proposed software tools, and defining the mappings needed to allow reasoning about the effects of the latter on mission performance.

2 Prior Work

Several approaches to process modeling have been pursued over the years. Perhaps the most mature discipline is that of job shop scheduling [9]. This usually involves modeling the process as a set of tasks flowing through a network of workstations and is typically solved using simulation or queuing theory models.

Some process modeling techniques have been suggested for the specific domain of software development. Petri net models have been proposed to allow prediction of development times and of changes in the process [10]. A variation of petri nets is the Software Process Analysis, Design, and Enactment (SPADE) environment [1]. SPADE includes the SPADE Language (SLANG), a domain-specific language for modeling software processes.

Another modeling approach, developed under Air Force support, is the ICAM Definition Methodology (IDEF), based on Integrated Computer Aided Manufacturing (ICAM). Several levels of models have been defined. Of particular interest here is IDEF3, created to model the sequence of activities in a manufacturing system. Two main components, the process flow description and the object...
state transition network description, define a sequence of activities and the relationships between them [8]. The IDEF3 model uses object state transition network diagrams to model object state changes relative to the process flow description. Some formal methods exist for analyzing the structure of an IDEF model.

The research described in this paper is an extension of the work by Hunt and Sarchet [6][15][4]. They developed an initial object model for processing an air tasking order in an operational Air Force wing using Rumbaugh's Object Modeling Technique (OMT) [14], including the dynamic (state) and functional (data flow) model. Although some formal Z schemas were written, they derived unit effectiveness and performance measures by mapping the dynamic model into a discrete event simulation written in Ada, and collecting those measures from the executing simulation. One of the problems that became evident from this approach was the difficulty in modifying the model or the collected unit metrics, along with the inherent problem of potentially long simulation execution time.

3 Methodology Overview

The methodology proposed in this research is different from other approaches in two ways. Formal modeling and domain analysis techniques, usually associated with product modeling, e.g. the software to be developed, are used extensively in a process-oriented model. The goal here was to take advantage of the ability to check correctness allowed by formal representations, and to allow for machine assistance as early in the modeling process as possible. A second difference is that while process modeling is usually done using a dynamic representation, such as queueing model or simulation, our approach derives performance measures from a static representation of formal relationships between objects. By using a declarative form of representing behavior, the goal was to develop a model that is easy to modify, allowing the executability of the formal language to provide the reasoning necessary to derive the metrics of interest.

3.1 Model Definition

A methodology can be defined at various levels of abstraction. Although the research effort was directed at modeling air tasking order preparation in an Air Force wing, the methodology itself was kept as general as possible in order to be applicable to a wide range of domains. The general model is that of a set of tasks (processes) to be performed, and a set of resources (workers and tools) to perform those tasks.

3.2 Define Goals

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 goal of model is to assess overall organization metrics such as unit readiness, unit effectiveness, and unit efficiency. Other uses of the model include evaluating the impact of automation tools on performing the tasks, and evaluating the status of the tasks. The model is also useful in the assessment of various proposed schedules for individual worker and tool assignments and their impact on the organization. The model is also intended to be easily modified for use by other organizations.

3.3 Object Model Development

The first modeling step is to define all of the objects in the domain, including attributes which describe their properties and states. Of particular interest here are attributes that reflect or impact the desired process performance metrics. These would include such things as task level of difficulty and average time to perform, worker skill level and experience level, and such state variables as task start time and tool idle/busy status.

The next step is to identify all relationships which exist between the objects. These relationships or associations are further analyzed to determine attributes of the associations where necessary. Again, emphasis is on those relationships and their attributes that impact the organization metrics. Examples would include the association between a task and the workers who are qualified to perform it as well as the association between a task and the specific worker assigned to perform it. The latter association might have an attribute to reflect the estimated time required for the specific worker/task pair. A special class of association is that of aggregation, sometimes referred to as the “composed-of” association, which relates a more complex object to the simpler objects of which it is composed. Significant aggregates in our process model include complex processes that are composed of simpler tasks, and organizational elements composed of workers, tools, and tasks to be performed.

In our methodology only the object model, or structural representation, is used. The dynamic (state) and functional (data flow) models are not defined. While our research used Rumbaugh's OMT, any informal object-oriented modeling technique could be used for this step.

3.4 Formalization of the Model

The object model is then converted into the formal language Z [3]. Although it is not directly executable, the Z language was chosen because it allows describing the model through the use of first-order predicate logic and mathematical symbol descriptions. Since it is not executable, its expressive ability is not limited by finite sets or other artificially imposed restrictions. In addition, its pure math notation is more familiar to practitioners than the syntax of another language. This allows the modeler to concentrate on the correctness of the representation without the distraction of syntax issues.
Z is used to directly represent the objects, attributes, and associations developed in the previous step. However, the real value of the formal representation is in defining constraints on and between objects, and in relating aspects of the objects to the organizational metrics. These are difficult to capture using the more informal modeling techniques. The formal representation allows a level of automatic reasoning about the system. For example, if task definitions include attributes for estimated time required and constraints concerning any tasks that must precede others, then given a start time, a completion time for all tasks can be deduced from the formal representation of that information.

3.5 Transforming Z to Refine

The next step is the transformation of the Z specifications into executable Refine constructs, the actual language of the final product [13]. The Software Refinery Environment (SRE) was chosen for its executability of behavior that is expressed as declarative first-order predicate logic. Refine directly supports most of the data types encountered in Z, including integer, real, boolean, symbol, sets, sequences, maps, and objects.

A class is represented in Refine as an object type. Each attribute is defined as a map from the object to the attribute domain. Functions can be defined over the objects and their attributes using Refine's wide-spectrum ability to execute any mixture of declarative and imperative commands.

In object-oriented design, associations are typically implemented either as embedded pointers in one or both of the associated classes, or as an associative object. Our model follows the latter approach by defining a link object class in Refine. Each link has attributes which designate the associated objects along with any link attributes attached to the association. Then a container object is declared to hold the set of these links. Although the built-in set type could be used for this, defining a specific container class for each association allows for specialized operations unique to each particular association. These association container objects are declared as subclasses of a Container superclass and inherit such general operations as AddItem, RemoveItem, and GetItem. Finally, the container is defined as an attribute of the aggregate which is composed of the associated objects. Our philosophy is that the association is only meaningful in the context of the aggregate. In some cases one may have to traverse several levels up the aggregate tree to find a common ancestor aggregate of the associated objects.

Constraints and metrics expressed as predicates in Z are mapped to functions in Refine. Constraints are typically mapped to pre-conditions for any functions that would change any of the involved variables. Metrics are treated as derived attributes, and become post-conditions of functions defined to return their value. Note that with Refine this is a straightforward mapping of predicates to declarative pre- and post-conditions, and does not require the design of procedural algorithms. This is directly related to allowing easy modification of the executable model to reflect changes in constraints or metrics, including tailoring them to another organization.

3.6 Executing the Model

Once the model has been coded in Refine it can be executed. This involves first entering any parameterized data (such as a set of tasks to be performed) and invoking the appropriate functions. This is facilitated in Refine by incremental compilation, allowing new functions (e.g. implementing new metrics) "on the fly" without recompiling the rest of the model; and the ability to invoke a function at any level in the model interactively without requiring a top level user-interface function. On the other hand, support is also available for defining user-friendly graphic interfaces suitable for use in a management environment.

4 Example Executable Model

This section discusses the specific domain model developed for this project. Although a specific domain was being considered, our objective was to keep the resulting model as general as possible (see discussion in Section 5). The informal model is presented first, followed by examples of the formalism in both Z and Refine. A more detailed description can be found in Hibdon [5].

4.1 Structural Model

The informal structural model is an extension of the work by Hunt and Sarchet [4]. The graphical representation is shown in Figure 1. An organization (the AF wing) consists of a workforce, a toolset, and a set of projects to be accomplished. The organization meaning plan is included in order to reason about unit readiness as related to unfilled positions and positions filled with the wrong
"type" of worker. Each project is broken down into a set of tasks, each of which is further decomposed into a set of jobs. Workers, positions, and tools comprise the remaining object classes in the model.

4.1.1 Attributes
Attributes of specific interest are the following:

**Worker:**
- name: unique identifier
- afsc: the worker's Air Force Specialty Code
- skill_level: the skill level held by the worker

**Job:**
- name: the identifier of the particular job
- afsc_set: set of afscs able to perform the job
- skill_level: worker skill level required for the job
- inputs: set of data items needed before the job can start
- outputs: set of items produced by the job
- average_time: the expected value for time required to accomplish the job
- best_time: the least amount of time required to accomplish the job
- worst_time: the most amount of time required to accomplish the job
- start_time: time the job started or will start
- end_time: time the job actually finished or will finish
- time: amount of time required to accomplish the job

**Task:**
- name: identifier of the task
- inputs: set of items which must be available for the task to begin
- outputs: set of items produced by the task
- number_of_jobs: total number of jobs that make up the task
- start_time: time when the task started
- end_time: time when the task finished
- wall_time: total time elapsed to complete the task
- total_time: sum of all Job times in the task

**Organization:**
- 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.

One of the problems with the informal model is the inability to precisely define aspects of the model, for example, the fact that a task's inputs are the inputs of its jobs that are not outputs from other jobs within the task.

4.1.2 Associations
The interrelationships among the objects are defined by the use of associations, which may or may not have associated attributes. Associations of specific interest are the following:

**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 is dependent on (derived from) the inputs and outputs of the Job objects.

**Assignment Association:**
The assignment association links a job to a worker who is to perform that job. There are no attributes attached to the assignment association, but the attributes of the worker and the job do impact the time attribute of the job object.

It may be necessary to determine the estimated time required to perform the job with the assignment of a particular worker. The average time to complete a job, as well as the best and worst times, is available as attributes of the job. The estimated time is determined as follows. If the assigned worker does not possess an afsc which is in the job's afsc_set, then the assignment is not allowed. If the worker and job skill levels match, then the average time value is used. If the worker's skill level is lower than the skill level desired by the job, the estimated time will be assigned the worst time value. If the person selected has a higher skill level than the job requires, the best time value will be used. These calculations for determining the time to perform the job could be changed based on domain specific expected time values if available. Note that it is difficult to accurately capture this constraint without the use of formal expressions.

4.2 Z Formalisms
The use of the Z formal language allows constraints and relationships to be accurately specified in a form that can be reasoned about automatically. Consider the following Z static schemas representing the Worker and Job classes. Z specifications consist of a graphical box called a schema. The upper half of the schema, the signature, contains variable (attribute) declarations, and the lower half, the predicate, contains multiple predicate logic statements conjuncted together which define constraints.


Note the ability to specify constraints, such as the fact that a job’s input data set must be different than its output data set. Most of these constraints, while necessary, are somewhat simple. The real value becomes more apparent when associations and aggregate objects are modeled.

### 4.2.1 Task Time Metric

For example, consider the `Precedes` association.

$\forall j_1, j_2 : \text{Job} \bullet (j_1, j_2) \in \text{precedes} \Rightarrow j_1.\text{end-time} \leq j_2.\text{start-time}$

The $\Rightarrow$ symbol indicates that `precedes` is a mathematical relation, an $m : n$ mapping between jobs. The first predicate indicates that `precedes` maps all job pairs where there is an overlap between the first job’s outputs and the second job’s inputs. The second predicate indicates that for any two such jobs, the first job must finish before the second job starts. While the `Precedes` schema defined above defines a general association between jobs, it can be further tailored in the context of the aggregate `Task` class, shown in the next column. The appearance of `Precedes` in the `Task` schema is an example of schema inclusion, similar to “includes” in C or “with” in Ada. Thus the `precedes` declaration and its two predicates are included in the `Task` schema. The first two predicates in `Task` indicate that only jobs in the task’s job set and considered in this association.

Note also the declarative representation of the task performance metric `wall-time` based ultimately on the individual job start and end times, which are themselves constrained by the `Precedes` association. Finally, note the declaration defining the inputs needed for the task based on the inputs and outputs of the jobs in its job set. In the full system, these concepts are repeated at the `Project` level.

### 4.2.2 Estimated Job Time

As another example, consider the `Assignment` association defined in the following Z schema.

$\forall (j, w) \in \text{assignment} \bullet w.\text{afsc} \in j.\text{afsc-set}$

$\forall (j, w) \in \text{assignment} \bullet w.\text{skill-level} < j.\text{skill-level} \Rightarrow j.\text{time} = j.\text{worst-time}$

$\forall (j, w) \in \text{assignment} \bullet w.\text{skill-level} > j.\text{skill-level} \Rightarrow j.\text{time} = j.\text{best-time}$

The $\bullet$ symbol represents a mathematical partial function, specifying formally that not all jobs need to be assigned a worker (partial), but those that are have a single worker assigned (function). In addition, the first predicate indicates that only job-worker pairs with matching afsc codes can be in the assignment. The remaining three predicates specify the relationship between the job and worker skill levels and the estimated job execution time.
4.3 Refine Representation

While Z has the advantages of unrestricted mathematical expression and the ability to define predicate constraints that “must always be true,” the inability to compile or execute the resulting model limits its usefulness. On the other hand, conversion to a classical procedural programming language is undesirable because it forces algorithm design choices, mixing the “how” of design and implementation with the “what” of pure specifications. What is needed is an executable language that supports declarative first-order predicate logic and set theoretic notation. The Refine language [13], as part of the Software Refinery Environment, provides this capability. Because it is a computer language, its grammar is more limited than the mathematical expression possible in Z, so some modification is needed. Objects and attributes in Z map to Refine as objects and attributes in Z map to Refine. They must be mapped to pre-and post-conditions of Refine functions. Consider, for example, the Refine implementation of the Worker class.

%---------------------------------------------------------
% Object Name: Worker
%---------------------------------------------------------
% Define base sets (types):
% constant AFSCS : set(string) = { "3332B", "3333C"}

% 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) = {};

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

function Getwname(w: Worker): string =
  wname(w);

function Setwafsc(w: Worker, afsc: string) =
  a in AFSCS --> wafsc(w) = afsc;

function Getwafsc(w: Worker): string =
  wafsc(w);

function Setwskill_level(w: Worker,
    skill: integer) =
  (s >= 1 & s <= 9 --> wskill_level(w) = skill;

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

%------------------- End of Worker ------------------------

The “-->” construct in Refine is similar to logical implication. It basically says that if the left hand expression is true, somehow make the right hand side true also. (Note that the "=" on the right hand side is not an assignment operator, but logical comparison.) Thus the left hand side forms a pre-condition for the operation and the right hand side forms a post-condition. In Setwname the wname attribute is always made equal to the input value since the left hand side is TRUE, while in Setwafsc the wafsc attribute is changed only if the input value is in the specified set, and in Setwskill_level the input value must lie in the specified range.

While the approach illustrated in Worker works fine for constraints on a single attribute, constraints involving multiple attributes must be handled differently. Usually this requires making one of the involved attributes a derived attribute whose value is calculated “on the fly” when requested. For example, in the Job schema is a constraint relating start_time, end_time, and time. Here end_time is determined to be the derived attribute, and in the Refine model is not declared as an attribute. As a result there is no Setjend_time function, and Getjend_time is defined as follows.

function Getjend_time(j: Job): integer =
  jstart_time(j) + jtime(j);

Next, consider the constraints in the Assignment association. These would be enforced whenever a job-worker assignment is attempted as follows.

function Setjob-worker(a:Assignment,
  j: Job, w:worker) =
  Getwafsc in Getjafscset -->
    (Getwskill_level < Getjskill_level -->
      Setjtime(Getjworst_time) &
      Getwskill_level = Getjskill_level -->
      Setjtime(Getjavg_time) &
      Getwskill_level > Getjskill_level -->
      Setjtime(Getjbest_time) )

Finally, consider the specification of the aggregate metric wall_time in class Task.

function WallTime(tsk: Task) =
  SettStart_Time(tsk) =
  Gettend_time(tsk) - Gettstart_time(tsk);

function SettStart_Time(tsk: Task) =
  TRUE --> tstart_time(tsk) <=
  jstart_time(arb(
    (j1 | (j1: Job) j1 in tjob_set(tsk) &
      (fa(j2) (j2 in tjob_set(tsk) =>
        jstart_time(j1) <= jstart_time(j2)))));

Here SettStart_Time uses set-former notation to first specify the set of all jobs whose start time is less than or equal to the start time of all other jobs, then select an arbitrary one and use its start time for the task start time.

4.4 Domain Data

Once the Refine model has been defined it is useable by executing it in the Refine environment. What is needed,
however, is to populate it with the necessary data. This is domain specific and will require varying levels of difficulty to obtain. A set of workers and tools must be specified. These can be determined from the organization’s manpower documents and equipment inventory. Also needed is a set of jobs. This may require more effort in analyzing the projects to be done and functionally decomposing them into the necessary sets of tasks and jobs.

Once all of these object instances have been identified, values for their attributes are needed. Again some, such as afs and skill level, can be determined directly from personnel records. Others, such as the estimated best, average, and worst times for job completion, may require more research. In some cases existing studies may be used. For example, the Air Force Human Resources Laboratory has developed a classification of job performance measurements for jobs and AFSCs throughout the Air Force [7]. Once these values have been specified, executing the model will generate the other derived attribute values as well as values for the specified performance metrics.

Our model currently treats the assignment of workers to jobs (as well as tools) as input data which it then uses to deduce other attribute values. However, the model as defined contains sufficient knowledge to allow specification of a scheduling function that would (given enough time) create an assignment of workers to jobs to satisfy a given criteria (such as minimum total completion time). However, Refine is not set up to generate an efficient algorithm, so for such a search problem an efficient algorithm would also need to be provided.

5 Reuse

Reuse is applicable at two levels in this project. Our original goal was to define a “reusable” methodology that any organization could follow to build a formal specification of its process. The methodology defined in Section 3 provides that level of reuse. By following the steps of defining an object model, formalizing it in Z, and transforming it to an executable model in Refine an organization can create an executable model of its specific processes.

However, the model developed in Section 4 is itself reusable in many organizations and can be easily tailored without going through all of the development spelled out in the methodology. An examination of Figure 1 clearly shows the generality of the overall architecture of our model. Most organizations consist of a set of workers assigned to execute a set of tasks using a set of available tools. This is similar to many job-shop scheduling models [9]. Furthermore, examination of object attributes reveals a similar generality. Although the specific names of the attributes are reflective of an Air Force organization (e.g., afs, skill level) these are simply domain-specific names for more general concepts reflecting classification of jobs and worker skills. Perhaps the biggest area of difference would be in the metrics definition. Unit-readiness might not be a common performance metric in a business organization. However, adding new metric specification is fairly easy to do in the Refine environment. For example, for a task it might be of interest to know the number of jobs that are dependent on external tasks before they can begin. All that would be needed is a Refine function to return this new derived variable as in the following.

\[
\text{function GetWaiters}(\text{tast: Task}): \text{set(Job)} = \\
\{ j | \ (j:Job) \ j \ \text{in} \ \text{tjob set}(\text{tast}) & \\
\text{ex}(i:DATA)(i \ \text{in GetJobInputs}(j) & \\
\text{i} \ \text{in GetTaskInputs}(\text{tast}) ) \}
\]

This specifies returning the set of jobs \( j \) where there exists a data item that is both in the job’s input set and the task’s input set. This is much simpler than having to write an Ada procedure to do the corresponding searching as well as modifying a higher level procedure to invoke the new one.

6 Conclusions

This paper describes an application of formal knowledge-based software engineering techniques to developing an organizational process model. A general methodology was described which basically consists of defining an informal object model of the organization’s resources (workers and tools) and requirements (processes to be performed): writing a formal Z specification capturing constraints, relationships, and derived performance metrics; converting the \( Z \) specification into an executable specification in Refine: defining the object instances and attribute values; and executing the specification to derive performance metric values. The methodology was then applied to modeling the processing of an air tasking order by an Air Force wing C3I section, and the resulting model was found to itself be reusable, not only for other Air Force organizations but also in many business environments.

While the general model is similar to many job shop scheduling models, our approach stresses formal representation of the model in declarative terms. This not only provides the ability to catch errors earlier due to the formal nature of the model, but allows the model to be directly executed without the need to design and debug procedural code. This leads to the ability to easily add metrics and modify constraints. For metrics where the Refine execution time could be lengthy, the wide-spectrum nature of the Refine language allows more efficient procedural code to be integrated within the declarative model.

The primary area of difficulty with this approach is the necessity to be able to express the model in formal predicate logic and set notation. However, the gains from having this formal model are felt to outweigh the extra effort at the front. In addition, research efforts are under way to provide a more informal interface to formal specifications [2]. The biggest nuisance in the current methodology is the need to transform from \( Z \) to Refine. One approach would be to eliminate \( Z \) and go directly from the OMT object model to Refine. However, our experience has indicated that the more pure math notation of \( Z \) along with the ability to specify invariant constraints
instead of immediately forcing one to write functions with pre-conditions warrant writing the initial formal model in Z. The other approach, which we are following, is to automate as much as possible the transformation from Z to Refine. An initial Z parser that creates a Refine abstract syntax tree (AST) from Z has been built [16]. With further development we expect this transformation system to actually transform the Z specification into Refine code.

Based on this project and other related efforts, we are optimistic about the value of executable formal models. Although there is a learning curve associated with writing formal specifications, the use of such formalization has an immediate impact in forcing the modeler to examine the domain in more detail and not overlook critical issues. The ability to directly execute these specifications without having to write and debug procedural code is an exciting approach to model development.

References


