Three Dimensions of Design Development

Neil M. Goldman

USC/Information Sciences Institute
4676 Admiralty Way
Marina del Rey, CA 90291

Abstract

Formal specifications are difficult to understand for a number of reasons. When the developer of a large specification explains it to another person, he typically includes information in his explanation that is is not present, even implicitly, in the specification itself. One useful form of information presents the specification in terms of an evolution from simpler specifications. Typically a specification was actually *produced* by a series of evolutionary steps reflected in the explanation. This paper suggests three dimensions of evolution that can be used to structure specification developments: *structural granularity*, *temporal granularity*, and *coverage*. Their use in a particular example is demonstrated.

1. Introduction

When we describe system behaviors to other people outside the confines of a formal language, it is common to find an evolutionary vein in the description. The final behavior can be viewed as an elaboration of some simpler behavior, itself the elaboration of a yet simpler behavior, etc., back to some behavior deemed sufficiently simple to be comprehended from a non-evolutionary description.

Formal specifications can likewise be described in an evolutionary vein. More importantly, the evolutionary steps can be characterized in terms of the kind of change they make to the specification. Three orthogonal dimensions of evolution have been identified.

- structural granularity -- deals with the amount of detail the specification reveals about each individual state of the process.
- temporal granularity -- deals with the amount of change between successive states revealed by the specification.
- coverage -- deals with the range of possible behaviors permitted by a specification.

Development steps along these dimensions can be composed to produce desired changes to a specification. This gives some hope that the steps themselves can be formalized -- i.e., that a language of change can be developed that permits a formal specification to be viewed and analyzed from its evolutionary perspective.

This research was supported by Defense Advanced Research Projects Agency contract MDA903-81-C-0335. Views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official opinion or policy of DARPA, the U.S. Government, or any other person or agency connected with them.

2. Specifications and Processes

A "formal specification" is something which denotes, according to well-defined rules, a *set of behaviors*. This set will be termed the *process* denoted by the specification. A behavior is a *sequence of states*. Each state comprises a set of *objects* and a finite set of *relations*. A relation is a set of n-tuples over the objects. A state models an instantaneous snapshot of a situation occurring during a particular execution of the process. A behavior then models the temporally ordered sequence of situations that constitutes a particular execution of the denoted process.

2.1. Initial Decisions

In order to specify some desired or existing activity within this framework, it is necessary to make certain (tentative) decisions about what aspects of the activity are to be specified. These decisions need not encompass all aspects of the activity. The goal at this initial stage is to specify enough of an abstraction of the activity to distinguish some of the relevant objects and actions involved, but not so much detail that the abstraction itself is difficult to write or comprehend.

Let us explore these considerations in the context of a particular example, a specification of the game of baseball. To structure our initial design, we consider three dimensions along which initial decisions must be made.

One decision to be made concerns an initial *structural granularity* for the specification. This amounts to deciding what information about a particular situation is to be encoded in a state -- i.e., what objects and relations are to be represented. The jargon of any particular domain generally gives many possibilities. In baseball, we talk about concrete objects like players, coaches, managers, umpires, bases, balls, bats, as well as more abstract concepts like teams, positions, and scores. Since baseball is a game, a minimally interesting formalization will have to capture the notion of the participants and scoring. So let us make our initial goal be to encode in each state only the score of each of the two teams in the game.

A second consideration concerns an initial *temporal granularity* for the specification. In this case, the jargon of baseball suggests several possibilities. One could take a snapshot after every pitch, or batter, or out, or half-inning, or inning, or just at the start and finish of the game. Our goal of keeping things simple suggests a

rule of thumb: start with the coarsest temporal granularity that suggests itself. In this case, that corresponds to making our initial goal be to let each state transition in a behavior cover a single inning of the game.

The third consideration concerns the behavioral *coverage* of the specification. One might aim for all possible complete games of baseball, or might choose to include possible cancelled or suspended games as well. The simplicity goal suggests another rule of thumb: start with coverage of normal cases only. In this case, that corresponds to making our initial goal be to include in the denoted process only complete, nine-inning games.

2.2. Development Decisions

One's initial version of a specification (or program) is an approximation, often a very crude one, to what is desired or achievable [3]. Traditionally, one refines the denoted process by altering the specification -- inserting, deleting, and replacing textual units of specification. In recent years, researchers have argued that viewing the development of a specification as a flat sequence of textual changes hides structure that directs those changes. Waters [4] and Barstow [2] show that many seemingly complex alterations are implementation idioms of a given language. Wile [5] allows the alterations to be grouped within a hierarchical goal structure reflecting meaningful concerns of the implementer. Structured developments are easier to comprehend and, it is argued, raise the level of expression available to the designer in a way that leads to gains in productivity.

This paper demonstrates that the same three dimensions of decision used to obtain an initial specification can be used to structure the development of that specification. A decision to specify greater detail about each state of a process is a *refinement* in the structural granularity dimension. Such a refinement leads to a new process and a many-to-1 mapping from the behaviors of the new process to those of the old. A new behavior has states in 1-1 correspondence with the states of the corresponding old behavior, but containing more objects and/or relationships. Conversely, a decision to specify less ("hide") detail about each state is an *abstraction* in the structural granularity.

A decision to reveal more of the individual state transitions of a process is a refinement in the temporal granularity dimension. Such a refinement leads to a new process and a many-to-1 mapping from the behaviors of the new process to those of the old. A new behavior contains its corresponding old behavior as a subsequence. Conversely, a decision to specify less ("hide") detail about state transitions is an abstraction in the temporal granularity.

A decision to add additional possible behaviors to a specification is an expansion in the *coverage* dimension. Such a refinement leads to a new process whose behaviors are a superset of those specified previously. Conversely, a decision to remove possible behaviors is a contraction in the coverage.

It will become apparent that, even in the simple example used in this paper, meaningful (in domain terms) changes to a specification do not constitute changes along a single one of these dimensions. However, it is demonstrated that meaningful changes can be represented as sequences of changes each along a single dimension, and that these representations could be useful both as explanations of specification development and as the means of specification development.

3. Baseball: an example

The initial specification of baseball appears in figure 3-1. Paraphrased, this specification says:

> There are two teams, called *home* and *visitor*. Each team has exactly one *Score*, which is 0 or more. Each team starts with a score of 0. A game consists of nine sequential innings. An inning is the net effect of each team batting once. The effect of a team batting is to leave the score unchanged or to increment the team's *Score* by some (integral) amount.

In this specification, the particular granularity decisions discussed above have been made explicit.

The notation used here to represent formal specifications is called Gist [1]. Understanding this notation in detail is not important. The point of the development that follows is that the textual changes to this notation *effect*, but do not clearly *convey* to a person, the nature of the change being made to the denoted process -- in this case, baseball. The changes to the process are described informally in terms of the three dimensions discussed above. This provides an alternative, and, it is claimed, preferable view of the development.

There are several directions in which one might choose to elaborate this definition of baseball. In particular, we might choose either to provide a more detailed definition of those baseball games which are included in the collection just defined, or we might try to define a more accurate set at the same level of detail. Let us start in the latter direction.

```
Behaviors from
   STATE ==> visitor:Score = home:Score = 0
   ACTIVITY ==> PlayBall()

agent TEAM(Score | non_negative_integer)
 definition {home, visitor}
 where
 action Bat()  definition
   self:Score :+= a non_negativeinteger ;

action PlayBall ()
   definition   9 times do PlayInning()  ;

action PlayInning ()
   definition begin atomic
                Bat() by visitor;
                Bat() by home
              end atomic
```

Figure 3-1: Initial Specification -- 9-inning game

3.1. Rule Out Tie Games

One thing we want to convey is that baseball games do not end in a tie. In development terms, this is a contraction in coverage -- we wish to restrict our specification to a subset of the currently specified behaviors. In Gist, contraction is often accomplished by adding a constraint. In this case, a postcondition on the overall activity will suffice (see figure 3-2). This might be paraphrased by:

The game must end with the two teams having different scores.

```
Behaviors from
   STATE ==> visitor:Score = home:Score = 0
   ACTIVITY ==> PlayBall()
                postcondition visitor:Score ≠
                              home:Score
```

Figure 3-2: Rule Out Tie Games

3.2. Extra Inning Games

We now have a more accurate, though not yet "correct", specification of nine-inning games. As a next step we could further refine this set, or, what seems more natural, introduce the concept of extra-inning games. This is a coverage expansion step, which might be paraphrased by:

> Actually, a game consists of *at least* nine innings, but will continue after that until an inning terminates with the score not tied.

One way to achieve this in Gist is with the change depicted in figure 3-3.

```
action PlayBall ()
   definition  until NormalTermination()
               do PlayInning();

relation NormalTermination()
  definition Inning(*) ≥ 9
             and visitor:Score ≠ home:Score;

relation Inning(I|non_negative_integer)
  definition I = count start PlayInning()
```

Figure 3-3: Extra Inning Games

3.3. One Team Bats at a Time

The behaviors defined now include all "normal" (barring war, riot, and natural disasters) complete baseball games, but still constitute too large a set. Among games that must be excluded are those in which the home team scores in the ninth inning when it had enough runs to win after eight innings. For example, a home team cannot lead 2-1 after the eighth inning and end up winning the game by a 5-1 score. The desired coverage contraction on the currently defined process could be accomplished by strengthening the termination condition. An English paraphrase of the condition would be:

> ... and, if the home team wins, then either (a) it scored no runs in the ninth inning, or (b) its score at the start of the ninth inning was no greater than the visiting team's score at the end of the ninth inning.

It seems absurd that anyone would choose to refine the description this way. The reason is that this is not a rule (axiom) of baseball as most people understand the game; rather it is a property (theorem) that follows from rules that are far more simply stated. To state these rules, however, requires a major refinement in the description; it will no longer suffice to think of an inning as an atomic event.

Here, then, is a case of *subgoaling* in the development. The primary goal is to contract the specification to a subset of behaviors. The means for achieving this requires refining the temporal granularity of the description. As a first step, we make an inning a sequential event (see figure 3-4), paraphrased as:

> Actually, an inning consists of *first* having the visiting team bat, and *then* having the home team bat.

```
action PlayInning ()
  definition    begin sequential
                   Bat() by visitor;
                   Bat() by home
                end sequential
```

Figure 3-4: Half Inning Granularity

Following this refinement, each behavior has two state transitions per inning rather than one. The key effect of this refinement is to provide half-inning updates of the score, making it simple to state the condition under which the home team does not bat, the key to achieving the needed contraction of the behavior. This can be accomplished in the specification text by conditionalizing the event (see figure 3-5), paraphrased as:

> The home team's turn at bat is skipped in the ninth inning if it is ahead.

```
action PlayInning ()
 definition  begin sequential
               Bat() by visitor;
               if Inning(9) and
                  home:Score > visitor:Score
                then null
                else Bat() by home
             end sequential
```

Figure 3-5: Skip Home Ninth

3.4. One Player Bats at a Time

Alas, the revised description still contains too many games. The problem is that, when the home team bats in the ninth inning or thereafter, it is not allowed to score an arbitrary number of runs, as it is in a normal inning. But now we are up against a problem we have just faced. The desired subset of behaviors is difficult to describe at the half-inning granularity; we would have to say something akin to:

> In the ninth inning and following innings, the home team may not score more runs than required to give it a margin of victory of four.

As before, this property sounds absurd as stated. It is more naturally thought of as a theorem following from rules stated more simply at a finer temporal granularity. *Half-innings* are not atomic events to someone who understands baseball, but are composed of an iteration of smaller events. In each of these smaller events, the batting team's score is incremented by at most four. Within this finer temporal granularity, the desired behavior subset is achieved by adding a simple mandatory termination exception, depicted in figure 3-6, to the home team's *Bat* event:

If the inning is 9 or greater, and the home team is ahead, the half-inning (and thus the game) terminates.

```
action PlayInning ()
 definition
  begin sequential
    Bat() by visitor;
    if Inning(9) and
       home:Score >  visitor:Score
       then null
       else Bat() by home
            exceptions
            (Inning(*) ≥ 9 and
            home:Score > visitor:Score)
            ==> null
  end sequential

action Bat()
   definition repeatedly Play()

action Play()
 definition  self:Score :+= choose{0,1,2,3,4}
```

Figure 3-6: Terminate Game When Home Team Has Won

The "smaller events" used to achieve a sufficiently fine temporal granularity within a half-inning do not correspond to any unit of activity suggested by the jargon of baseball. They are an unnatural abstraction that serve the purpose of having the score incremented in suitable units. In fact, the next finer unit of activity suggested by baseball is the "player's-turn-at-bat". To suitably model this event, it is necessary to introduce a *structural refinement* to the specification. That is, to avoid introducing odd-sounding theorems as axioms, we need more detail about a state than just the score. We need to introduce the concept of "men on-base" . There are a number of ways one might state the effect of an at-bat. For example:

> At the start of a half-inning, there are zero men on-base and zero outs. There may never be more than three men on-base at once. The effect of a single player's at-bat is: Let M be the number of men on-base. The team's score and the number of outs are each increased by between 0 and 1 + M. The number of men-on-base is incremented by between +1 and -M. The number of outs may not exceed three. When the number of outs reaches three, the half-inning terminates. At the end of each player's at-bat, the following invariant must hold:
>
> The number of players having batted in the half-inning equals the sum of the net increment to the batting team's score, the current number of men on-base, and the current number of outs.

It can now be seen that the seemingly arbitrary four runs maximum per scoring play follows from the (equally arbitrary) three men on-base maximum and 1 + M score increment limit. These in turn follow from factors that could, but won't, be defined by going to finer levels of structural and temporal granularity.

4. Conclusions

We have identified three dimensions along which process specifications change as they are elaborated. The dimension of *structural granularity* determines the level of detail the specification reveals about each state of the process. The dimension of *temporal granularity* determines the time slices at which the specification models the process. If both temporal and structural granularity are fixed, a specification may be changed so as to expand or contract the *coverage* of behaviors. It appears that the best way to achieve a change along one dimension may involve making a change along one of the other dimensions.

A view of a specification in terms of its structured development differs significantly from a view of a specification that expresses only the net result of that development. In the example presented in section 3, each development step was formally expressed in terms of a change to the textual representation of the specification. But, as is almost always the case, there are numerous textual changes that would have the same semantic effect. The intent of the development step, however, is not to describe a change to the specification per se, but to change the process denoted by the specification. Although we have not yet done so, it seems plausible that the development steps stated in English in this paper could be formalized directly in terms of functions mapping processes to processes rather than as mappings from specifications to specifications. In that case, the initial specification and sequence of structured modifications would present a complete definition of the final process and would arguably be both easier to produce and easier (for a person) to comprehend.

Acknowledgements

I wish to thank Robert Balzer, Don Cohen, Martin Feather, Jack Mostow, Bill Swartout and Dave Wile for many discussions on the understandability (or lack thereof) of software and specifications and for comments on this paper.

References

1. Balzer, R., Goldman, N. & Wile, D. Operational specification as the basis for rapid prototyping. Proceedings of the Second Software Engineering Symposium: Workshop on Rapid Prototyping, ACM SIGSOFT, April, 1982.

2. Barstow, D.R.. *"Knowledge-Based Program Construction"*. Elsevier North-Holland, 1979.

3. Swartout, W. and Balzer, R. "On the Inevitable Intertwining of Specification and Implementation." *Communications of the ACM 25*, 7 (July 1982), 438:440.

4. Waters, R. C. "The Programmer's Apprentice: Knowledge Based Program Editing." *IEEE Transactions on Software Engineering SE-8*, 1 (1982), 1-12.

5. Wile, D. S. Program Developments: Formal Explanations of Implementations. Tech. Rept. RR-82-99, ISI, August, 1982.