Track Assignment in an Air Traffic Control System: A Rational Reconstruction of System Design

Douglas R. Smith
Kestrel Institute
3260 Hillview Avenue
Palo Alto, CA. 94304

Abstract

This paper summarizes our application of the KIDS system to a larger-scale problem drawn from the domain of Air Traffic Control. Hughes Aircraft supplied us with a natural language specification of the Track Assignment (TA) portion of an Air Traffic Control System. We formalized this specification as a domain theory and derived programs within this theory. The derived code is comparable to ADA code that was produced by conventional means.

1. Introduction

This paper summarizes our application of the KIDS system to a larger-scale problem drawn from the domain of Air Traffic Control (ATC). A complete description of this work may be found in [13]. Previously, KIDS has been used mainly to recapitulate the design of core Computer Science algorithms as defined by standard textbooks (although we have also been sucessful in synthesizing algorithms that were previously unknown [9, 11]). Applying KIDS to an ATC problem was a test of (i) the model of software development that KIDS supports, (ii) the adequacy of the programming knowledge in KIDS, and (iii) the adequacy of its inference mechanisms.

Air Traffic Control (ATC) is a realtime monitoring problem. The problem is to maintain a faithful representation of objects and events in the real world based on incomplete information provided by sensors. The objects of this world include airspaces, aircraft, airfields, positions, and signals; their attributes include velocity, range, altitude, etc.; events include entry of aircraft into airspace at a particular time and place, etc.

Our project focused on the Track Assignment (TA) portion of an ATC system. Periodically, signals are received at the ATC center from aircraft. These signals must be correlated with the tracks (representations of flight paths) of aircraft in the airspace. The track assignment problem (for the purposes of this paper) is: given a signal received from an aircraft, correlate it with a track and if it does not correlate with any known track then create a new track for it. The purpose of this part of ATC is of course to maintain fidelity between the track database and the aircraft currently in the airspace.

KIDS [14] is a knowledge-based system for developing correct programs from formal specifications. We have used KIDS to design and optimize algorithms for over fifty problems. Examples have been drawn from diverse problem domains such as scheduling, combinatorial design, sorting and searching, linear programming, and graph theory. The KIDS system and the model of software development that it supports are described in Section 2. Generally, the approach is to build a formal model of the application domain, encoded as a domain theory, including a specification of the target system's desired behavior, then to develop detailed code by applying transformations to the specification. The transformations encode design and optimization knowledge and allow the user to mechanically make high-level design decisions with confidence that the system will apply them without error. A system for developing consistent extensions to domain theories is discussed in Section 3.

The starting point for our derivation was a structured English specification of the Track Assignment portion of an ATC. This specification was developed by Hughes Aircraft preliminary to their development of code for a customer. In Section 4 we present some representative fragments of this specification interleaved with our formalization of a domain theory for TA and commentary on the specification and the pro-

0-8186-2880-4/92 $03.00 © 1992 IEEE

cess of formalization. Section 5 describes our development of code within this theory. This derived code is then compared with hand-generated ADA code developed manually by Hughes personnel.

2. KIDS Model of Specification and Development

KIDS is a program transformation system – one applies a sequence of consistency-preserving transformations to an initial specification and achieves a correct and hopefully efficient program. The system emphasizes the application of complex high-level transformations that perform significant and meaningful actions. The user typically goes through the following steps in using KIDS for program development.

1. *Develop a domain theory* – The user builds up a domain theory by formalizing the concepts of the domain in terms of appropriate types, functions, predicates, and laws. The laws allow high-level reasoning about the defined functions. In Section 3 we describe a theory development component that supports the development of consistent theories.
2. *Create a specification* – The user enters a specification stated in terms of the underlying domain theory.
3. *Apply a design tactic* – The user selects an algorithm design tactic from a menu and applies it to a specification. Currently KIDS has tactics for simple problem reduction (reducing a specification to a library routine) [8], divide-and-conquer [8], global search (binary search, backtrack, branch-and-bound) [10], problem reduction generators (generalized branch-and-bound, dynamic programming, game tree search) [12], local search (hillclimbing) [6], and others.
4. *Apply optimizations* – The KIDS system allows the application of optimization techniques such as simplification, partial evaluation, finite differencing, and other transformations. Each of the optimization methods are fully automatic and, with the exception of simplification (which is arbitrarily hard), take only a few seconds.
5. *Apply data type refinements* – The user can select implementations for the high-level data types in the program. Data type refinement rules carry out the details of constructing the implementation.
6. *Compile* – The resulting code is compiled to executable form. In a sense, KIDS can be regarded as a front-end to a conventional compiler.

KIDS is written in the Refine language and it transforms a slightly extended variant of Refine. The language used in this paper has set-theoretic datatypes (such as finite sets, sequences, and maps) together with boolean operators from first-order logic, and functional programming constructs. We hope that the notation is close enough to conventional usage that readers unfamiliar with Refine can get by. Novel constructs are explained as they arise.

To help new users understand the system, KIDS includes recorded derivations that can be replayed "player-piano" style under user control. The KIDS manual lists twelve derivations step-by-step including their complete domain theories.

3. Theory Development

A domain model defines a language that is appropriate for describing and reasoning about an application domain. A formal specification is expressed in terms of some such domain language and describes the desired behavior of a program or system. KIDS uses *theories* as the means for formally modelling the concepts and laws of various application domains. The basic concepts of a theory are used to in specifying desired behaviors. The laws are used by the inference system in support of design, optimization, and refinement steps. A *theory presentation* (or simply a *theory*) includes a list of type definitions, function and predicate symbols, laws, and specialized rules. Rules are obtained from laws by specifying how to control their use; e.g. an equation might be turned into a rewrite rule by specifying that it be used in a left-to-right orientation and applied in an irrevocable manner. For purposes of the TA derivation, we extended our theory presentation syntax to allow definition of a hierarchy of object classes and their attributes.

As of mid-1992 we have built over 80 theories in KIDS. There are theories for basic datatypes and mathematical structures such as binary-trees, bags, maps, sequences, segments, sequences-as-linear-orderings, binary-relations, basic-arithmetic, and others. These basic theories are imported by more application- and problem-specific domain theories such as sorting, cyclic-difference-sets, costas-array-theory, ATC-Track-Assignment, and scheduling.

We have also implemented various operations on theories that support the user in developing consistent theories.[1] There are a collection of tools for managing the theory library – the user can load, create, save, or compile theory objects. Loading a theory has the effect of installing the types, operations, laws, and rules in the REFINE object base. The rules are then available to the inference system. One advantage to this approach is that only those concepts and rules are loaded that are needed for the problem at hand. This modularization of domain knowledge is an important efficiency consideration as we build knowledge bases with thousands of rules.

There are also tools to introduce new functions, laws, and rules so as to preserve consistency of a theory. KIDS provides an Abstract operation that allows users to select a subexpression on the screen and to parameterize it into a new function definition. Another KIDS operation allows users to derive various kinds of laws for a given function, principally distributive laws. For example, consider the following concept definition that arises in the Track Assignment problem discussed later. It is not necessary to understand the meaning of the terms in order to understand the derivation of a distributive law. This concept defines the set of established tracks that can be associated with a given radar-datum and which having a matching beacon of SIF return:

```
function ASSOCIABLE-ESTABLISHED-TRACKS-MATCHING-SIF
  (PLOT : RADAR-DATUM, CURRENT-TRACKS:set(TRACK)
    | VALID-TRACK-SET(CURRENT-TRACKS))
  : set(TRACK-DATUM-ASSOC)
  = {CORRECT-SLANT-RANGE-AND-TIME-DELAY
              (MAKE-TRACK-DATUM-ASSOCIATION(plot, trk))
    | TRK in CURRENT-TRACKS & ESTABLISHED-TRACK(TRK)
    &R-D-SIMULATION-STATUS(PLOT) = TRACK-SIMULATION- STATUS(TRK)
     & MATCHING-SIF(PLOT, TRK) & PROXIMITY(PLOT, TRK) ≤ 16
     & (TRACK-SIMULATION-STATUS(TRK) ≠ EXERCISE
              ⟹ MODE-3-SIF(PLOT))}
```

In order to reason about this concept (and to design code for problems that involve this concept) we often need to have laws describing certain properties of it. For functional concepts such as ASSOCIABLE-ESTABLISHED-TRACKS–MATCHING-SIF we typically need to derive distributive laws for it – e.g. how ASSOCIABLE-ESTABLISHED-TRACKS–MATCHING-SIF distributes over constructors of its arguments. In [13] we show how KIDS derives laws for distributing ASSOCIABLE-ESTABLISHED-TRACKS–MATCHING-SIF over set constructors — empty-set, singleton-set, and union of current-tracks. The derivation is completely automatic and takes less than a minute on a SUN4 workstation. The Derive-Distributive-Law tactic uses a structured unfold/fold strategy that is predisposed to the desired fold steps. The resulting law for union is

```
∀(PLOT,CURRENT-TRACKS-1,CURRENT-TRACKS-2)
  AET-MATCHING-SIF(PLOT,
                   CURRENT-TRACKS-1 ∪ CURRENT-TRACKS-2)
  = AET-MATCHING-SIF(PLOT,CURRENT-TRACKS-1)
    ∪
    AET-MATCHING-SIF(PLOT,CURRENT-TRACKS-2).
```

In this section we discuss Air Traffic Control (ATC) and the Track Assignment (TA) problem and show how a domain theory was built up for this application. Most of the laws in this domain theory were derived using KIDS. In the next section we summarize the derivation of code from the TA specification in this theory.

A review of some background concepts will help to understand the specification. An ATC system typically receives two kinds of radar signals, called radar data or plots. A radar signal reflected off an aircraft is called a *search radar datum*. A signal obtained by stimulating the transponder on an aircraft is called a *beacon return* or *SIF radar datum*. Beacon returns come in a variety of modes – the specification below mentions modes 2, 3, and 4. Each mode has a code that uniquely identifies the aircraft.

A *track* represents an identified aircraft and its trajectory. The *Established tracks* are the collection of currently active tracks. Tracks can have a *simulation status* – either live or exercise. A *track-datum association* is a tentative association of a radar datum and a track for the purposes of correlation processing.

To formalize this domain, we defined various object classes, attributes, and types in TA-theory. Object classes included radar-datum, tracks, established-tracks, scrambled-tracks, airborne-tracks, ans track datum-associations. Each class has various attributes that charcterize the structure and properties of individual objects in the class. For example, track objects have a velocity, predicted position, altitude, identification, SIF-mode code, and others.

We now interleave some fragments of the Hughes TA Specification with discussion and presentation of our formalization of a KIDS domain theory for TA. In this presentation, the Hughes specification will be presented in a `typewriter` font and the KIDS domain theory will appear in a *math-italic* font. Commentary will appear in ordinary roman font.

[1] A theory is *consistent* if there is no formula A such that A and $\neg A$ are provable in the theory.

The data correlation process shall consist of the following:

1. Gross distance test,
2. Slant range correction,
3. Time delay correction,
4. Search areas, and
5. Track selection.

This initial fragment of the specification already indicates its operational nature. Specifications generally are intended as a way to describe/prescribe the overall behavior of software while omitting unnecessary operational detail. This specification prescribes a sequences of five steps. By not presenting a more precise description of what constitutes a successfully correlated track for a given plot, we may not end up with the best possible code. On the other hand, there may be several reasons for specifying a system in an operational manner. One is that there may be corporate, national, or international standards that must be adhered to (protocols that must be followed). Another is that experienced ATC system designers may wish to provide high-level guidance on the structure of successful ATC systems. Or it may simply be the case that an operational specification is the simplest or most natural way to present system behavior. The difficulty with operational specifications is the lack of well-established tools for reasoning about them and overcommittment to a solution method. Does the above structure lend itself to an efficient parallel solution?

Before moving on, it may help to explain the intent of the above fragment. The "gross distance test" is a means to limit the number of candidate tracks to consider for correlation to a given plot. As seen below, the candidate tracks are filtered to those whose last position was within a certain distance of the position of the given plot. With respect to those candidates, the plot undergoes Slant range and Time delay correction. Next the plot-track candidates are classified into Search areas, and finally the track that best correlates with the plot is selected. For brevity, we only present and discuss fragments of the Gross Distance test and Track selection steps.

3.2.4.3.2.1.1 Gross Distance Test. Prior to the attempt to correlate radar data with Established tracks, a gross distance test shall be performed for each eligible track-datum pair in order to reduce the number of track-datum comparisons to be made during correlation. Only tracks having the same simulation status as the radar datum shall be eligible for data association. The gross distance test processing will consist of the following.

a. If SIF code stored with an Established track matches exactly the datum's code of the same mode, and if the track is within 16 nmi of the datum, the datum shall be considered associated with the track and shall undergo slant range and time delay corrections as specified in 3.2.4.3.2.1.2 and 3.2.4.3.2.1.3, respectively.

Our KIDS domain theory for this part of the specification assumes that there is a track database called *current-tracks*. The function *ASSOCIABLE-ESTABLISHED-TRACKS-MATCHING-SIF* returns a set of plot-track pairs that are candidates for correlation.

```
function MATCHING-SIF
  (PLOT : RADAR-DATUM, TRK : TRACK) : boolean
  = R-D-SIF(PLOT) = TRACK-SIF(TRK)

function ASSOCIABLE-ESTABLISHED-TRACKS-MATCHING-SIF
  (PLOT : RADAR-DATUM, CURRENT-TRACKS:set(TRACK)
     | VALID-TRACK-SET(CURRENT-TRACKS))
  : set(TRACK-DATUM-ASSOC)
  = {CORRECT-SLANT-RANGE-AND-TIME-DELAY
            (MAKE-TRACK-DATUM-ASSOCIATION(plot, trk))
     | TRK in CURRENT-TRACKS & ESTABLISHED-TRACK(TRK)
     &R-D-SIMULATION-STATUS(PLOT) = TRACK-SIMULATION- STATUS(TRK)
      & MATCHING-SIF(PLOT, TRK) & PROXIMITY(PLOT, TRK) ≤ 16}
```

b. A search radar datum or SIF radar datum which did not match a code with any track shall be tested for proximity to Established tracks. If a track with a speed of 600 knots or less is found within 10nmi, or track with a speed greater than 600 knots is found within 11nmi of the datum, the datum shall be considered associated and shall undergo slant range and time delay corrections as specified in 3.2.4.3.2.1.2 and 3.2.4.3.2.1.3.

```
function NONMATCHING-SIF
  (PLOT : RADAR-DATUM, TRK : TRACK) : boolean
  = R-D-SIF(PLOT) ~= TRACK-SIF(TRK)

function ASSOCIABLE-ESTABLISHED-TRACKS-NONMATCHING-SIF- SLOW
  (PLOT : RADAR-DATUM, CURRENT-TRACKS:set(TRACK)
     | VALID-TRACK-SET(CURRENT-TRACKS))
  : set(TRACK-DATUM-ASSOC)
  = {CORRECT-SLANT-RANGE-AND-TIME-DELAY
            (MAKE-TRACK-DATUM-ASSOCIATION(plot, trk))
     | (TRK : TRACK)
       TRK in CURRENT-TRACKS & ESTABLISHED-TRACK(TRK)
       &R-D-SIMULATION-STATUS(PLOT) = TRACK-SIMULATION- STATUS(TRK)
        & NONMATCHING-SIF(PLOT, TRK) & TRACK-SPEED(TRK) ≤ 600
        & PROXIMITY(PLOT, TRK) ≤ 10}
```

```
function ASSOCIABLE-ESTABLISHED-TRACKS-NONMATCHING-SIF-FAST
  (PLOT : RADAR-DATUM, CURRENT-TRACKS:set(TRACK)
    | VALID-TRACK-SET(CURRENT-TRACKS))
  : set(TRACK-DATUM-ASSOC)
  = {CORRECT-SLANT-RANGE-AND-TIME-DELAY
          (MAKE-TRACK-DATUM-ASSOCIATION(plot, trk))
    | (TRK : TRACK)
       TRK in CURRENT-TRACKS & ESTABLISHED:TRACK(TRK)
      &R-D-SIMULATION-STATUS(PLOT) = TRACK-SIMULATION- STATUS(TRK)
       & NONMATCHING-SIF(PLOT, TRK) & TRACK-SPEED(TRK) > 600
       & PROXIMITY(PLOT, TRK) ≤ 11}
```

The above formalization was straightforward. These concepts are aggregated in the concept of *ASSOCIABLE-ESTABLISHED-TRACKS*:

```
function ASSOCIABLE-ESTABLISHED-TRACKS
  (PLOT : RADAR-DATUM, CURRENT-TRACKS:set(TRACK)
    | VALID-TRACK-SET(CURRENT-TRACKS))
  : set(TRACK-DATUM-ASSOC)
  = ASSOCIABLE-ESTABLISHED-TRACKS-MATCHING-SIF
      (PLOT, CURRENT-TRACKS)
     union ASSOCIABLE-ESTABLISHED-TRACKS-NONMATCHING-SIF-SLOW
           (PLOT, CURRENT-TRACKS)
     union ASSOCIABLE-ESTABLISHED-TRACKS-NONMATCHING-SIF-FAST
           (PLOT, CURRENT-TRACKS)
```

For reasons of space, we skip a section of specification that prescribes three refinements of the gross distance test.

The following sections prescribe how to select the best of the plot-track pairs for final correlation and update of the track database.

3.2.4.3.2.1.5 Track Selection. Each radar datum shall be tentatively correlated with one eligible Established track at most. If second eligible track becomes a candidate for correlation, one of the two tracks shall be discarded and the datum shall be tentatively correlated with the selected track. The tests employed to elect one of the two tracks shall be as follows, in order of priority:

a. The track that matches the SIF radar datum in a Mode 2 SIF code shall be selected.

b. The track that matches the SIF radar datum in a Mode 3 SIF code shall be selected.

c. For a SIF radar datum, if one track is a search track and the other is a beacon track, the beacon track shall be selected.

d. For a search radar datum, or if the two tracks are of the same type, or if the two tracks have the same matching SIF code, the closer of the two tracks shall be selected, or if equidistant, the first track correlated with the datum shall be selected.

We formalized the ordering above as a priority function.

```
Type CORRELATION-STATUS = symbol

Constant priority : map(CORRELATION-LEVEL, integer)
  = {| 'MATCHING-MODE-2 -> 0,
       'MATCHING-MODE-3 -> 1,
       'NON-MATCHING-SIF -> 2,
       'SEARCH -> 3,
       'NONE -> 4 |}

Type CORRELATION-LEVEL = symbol

function PLOT-TRACK-CORRELATION-LEVEL
  (PLOT : RADAR-DATUM, TRK : TRACK) : CORRELATION-LEVEL
  = if MODE-2-SIF(PLOT) & MATCHING-SIF(PLOT, TRK)
       then 'MATCHING-MODE-2,
    elseif MODE-3-SIF(PLOT) & MATCHING-SIF(PLOT, TRK)
       then 'MATCHING-MODE-3,
    elseif SIF-RADAR-DATUM(PLOT) & NONMATCHING-SIF(PLOT, TRK)
       then 'NON-MATCHING-SIF,
    elseif SEARCH-RADAR-DATUM(PLOT)
       then 'SEARCH,
    else 'NONE
```

Track-datum associations are compared via

```
function COMPARE-PLOT-TRACK-ASSOCIATIONS
  (TDA1 : TRACK-DATUM-ASSOC, TDA2 : TRACK-DATUM-ASSOC
   | T-D-A-RADAR-DATUM(TDA1) = T-D-A-RADAR-DATUM(TDA2))
  : boolean
  = (let (PLT : RADAR-DATUM = T-D-A-RADAR-DATUM(TDA1),
          TRK1 : TRACK = T-D-A-TRACK(TDA1),
          TRK2 : TRACK = T-D-A-TRACK(TDA2))
       PRIORITY(PLOT-TRACK-CORRELATION-LEVEL(PLT, TRK1))
     < PRIORITY(PLOT-TRACK-CORRELATION-LEVEL(PLT, TRK2))
     or (PRIORITY(PLOT-TRACK-CORRELATION-LEVEL(PLT, TRK1))
         = PRIORITY(PLOT-TRACK-CORRELATION-LEVEL(PLT, TRK2))
         & PROXIMITY(PLT, TRK1) ≤ PROXIMITY(PLT, TRK2)))
```

The concepts defined above allow us to specify track correlation as the problem of selecting the best candidate track[2]

```
function CORRELATE
  (PLOT : RADAR-DATUM, CURRENT-TRACKS:set(TRACK)
    | VALID-TRACK-SET(CURRENT-TRACKS))
  returns
   (TDA : TRACK-DATUM-ASSOC
     | EXTREMAL
        (TDA,
         lambda(TDA1, TDA2)
           COMPARE-PLOT-TRACK-ASSOCIATIONS(TDA1, TDA2),
         CORRELATABLE-TRACKS(PLOT, CURRENT-TRACKS)))
```

For the purposes of this paper, the function *CORRELATABLE-TRACKS* can be thought of as being the same as the function *ASSOCIABLE-ESTABLISHED-TRACKS* defined earlier, although in fact it is more complicated. Here *extremal(x,p,S)* means that x is an extremal element of S with respect to the linear ordering relation p. Formally we define

$$extremal(x, p, S) = (x \in S \wedge \forall(y)(y \in S \implies p(x, y))).$$

Thus we specify the track assignment problem as an optimization problem – of all the correlatable tracks

[2]The returns clause gives the input-output condition on a specification. The absence of a body here indicates that we need to design code that meets the input-output condition.

select the one with highest priority, or if several have the same priority then select the closest.

This completes our description of the domain theory for the Track Assignment problem. We now turn to a few comments on the Hughes specification.

Our first observation is that the specification is fairly operational in nature. It may be that this is the best way to specify the desired system within the current Hughes environment. However, I found the specification to suggest a sequential computation. If the specification were more abstract, then a parallel system might also be allowed. In principle I feel that any sequentialities in the problem domain should fall out of dependency analysis between concepts. Otherwise it may not be obvious which sequentiality committments are necessary, which are suggestive of a good solution, and which are simply arbitrary.

Another observation is that the specification seems to dictate a wasteful solution method. After the gross distance test, the specification goes on to prescribe three other more refined distance tests used to filter out implausible plot-track associations. Since we already resolve ties in SIF-mode matching on the basis of distance, the other tests seem cumbersome and unnecessary. Furthermore, the gross distance test itself seems naive. At modern computing speeds the distance calculations are measured in microseconds and it is almost inconceivable that a system could be overwhelmed with plot track pairs to consider. My impression is that this specification prescribes methods that were developed during a period when manual correlation was performed. These methods have been written into the specification without rethinking the essence of the correlation process.

Another observation is that the specification is somewhat disorganized. Several subsections present constraints on the correlation process that should have appeared when the association process was specified. The danger is that these seemingly out-of-place constraints may result in "patchy" code. Conceptual coherence at the specification level is necessary to obtaining clean well-structured code.

Finally we comment on the development of domain theories. In our experience with KIDS, the effort to develop domain theories is much greater than the effort to develop code, when the requisite programming knowledge is available. For the TA derivation, the difficulty in developing a theory was increased by the (self-imposed) requirement of working from the ambiguous and incomplete natural language specification. At times we had to inspect the ADA code in order to disambiguate the specification or to fill in gaps.

For several reasons the development of domain theories is worth the effort. Domain theories provide rigorous foundations for stating, reusing, and communicating precise knowledge about the application domain. Of course they also are critical for supporting the development of correct, well-structured code with a formal record of design decisions as documentation. This information is necessary for subsequent evolution of the system.

The theories in KIDS are arranged in a hierarchy with lower-level more general theories imported by higher-level more problem-specific theories. Lower-level theories are routinely reused for new problem domains. The need to systematically build domain theories motivated our work on the theory development system. There is a great deal more to do on this task, including the ability to evolve specifications as in ARIES [4].

4. Derivation of Code

As described in the previous two sections, theory development precedes code development. We formalized the basic concepts of the TA problem based on the Hughes English specification and used KIDS to derive most of the laws and inference rules in the Track-Assignment theory. The resulting TA-theory is about 21 pages long. It took several months to understand the Hughes specification well enough to produce this domain theory. With that theory in hand we turned to deriving code.

The structure of the derivation is fairly straightforward. During the course of the derivation, we unfold some concepts in the CORRELATE specification, and soon derive some specialized subproblems; in particular the problem of finding the best candidate track among the ASSOCIABLE-ESTABLISHED-TRACKS. We solve this problem by applying divide-and-conquer and choose a simple decomposition operation: split the set of associable tracks in half. Each half is processed to obtain the track that correlates best with the given radar-datum, and then the better of these is returned. The usual choice is to split into one track and the remaining tracks, resulting in a simple looping program. Our choice allows the possibility of parallel computation.

The result is a logarithmic time parallel solution versus a linear time solution. Of course this improvement may only be significant if there are large numbers of tracks that could associate with the radar-datum. Several other subproblems are solvable via divide-and-conquer also.

Simplification is interspersed throughout in order to clean-up the code and reduce expressions to canonical or simplest possible form. The derivation could be continued in a number of ways — finite differencing could be used to save proximity of a track-datum-association; loop fusion could merge the loops for finding extremal established, scrambled, and airborne tracks; and so on.

5. Comparison with Hughes ADA Code

There are a few differences between the underlying domain theories of the Hughes ADA specification/code and the Hughes Structured English (HSE) specification. These differences make the KIDS-generated code and the Hughes code somewhat incomparable. Nevertheless, the KIDS code and the Hughes ADA spec do solve almost the same problem and in essentially the same way. Much of the code arises fairly directly from the underlying domain theory. Design and optimization tools were applied, but not in a very deep or surprising way.

The KIDS code is functional; one effect is that the implicit computational requirement that the best currently correlated track is associated with the plot is violated. In an imperative setting one can record the best current solution. If the system should be interrupted, due to error or running out of time etc, then the best correlated track is available. In the KIDS code the *Correlate* function must run to completion before the correlation can be made. Conversion to imperative code and subsequent program optimizations would take care of this problem.

The most obvious structural difference between the two bodies of code is that the Hughes ADA code abstracts out several routines that are compactly stated in the KIDS code. For example, the Hughes ADA code has separate correlation routines for established tracks with matching SIF and nonmatching SIF. This could be achieved in KIDS by unfolding ASSOCIABLE-ESTABLISHED-TRACKS which unions together these concepts.

As part of the foundations of KIDS, we have attempted to identify basic patterns of computation and to formalize their design. Our belief is that these patterns arise as all scales of programs. We expect to find that larger-scale systems are composed a great quantity of programs with simple algorithmic and data structure content (e.g. most of the loops that one builds are simple instances of the divide-and-conquer pattern). In the TA derivation, much of the program detail comes from the underlying domain theory — there is less of a distance between specification and code than in deeper algorithms previously derived by KIDS. This does not obviate the need for KIDS-like design tools, it just means that they are pervasively applicable in fairly simple and tractable ways.

6. Domain-Specific Software Synthesis

Several current systems emphasize domain-specific software synthesis: Sinapse [5], ELF [7], GENESIS [1], LEAP [3], and others. Each tries to gain an advantage by building domain-specific concepts and programming knowledge into the structure of the specification language, knowledge representations, and program generator.

We believe that it is important to combine general-purpose tools for algorithm design, program optimization, and refinement with domain-specific knowledge. For example, Batory [1] claims that a key weakness of the GENESIS system (which specializes in the generation of database systems) is its lack of tools for optimizing the code and choosing different datatype refinements. The ELF system [7] comes closest to providing both general-purpose tools (data structure selection) and domain-specific synthesis (algorithm templates for wire routers for VLSI). Without these general-purpose tools the designer of a domain-specific system must anticipate the performance issues when representing domain knowledge. This is at odds with current methodology which emphasizes the value of a separation of concerns.

KIDS aims for domain-specific performance in two ways. First, by definition domain theories encode a model of an application domain, capturing the basic concepts and laws for reasoning about them. Second, we also use parameterized theories to represent knowledge about algorithms and algorithm design [15] and datatype refinement [2]. The algorithm theories are arranged in a refinement hierarchy. The deeper one goes in the hierarchy, the more structure is as-

sumed in problems for which the corresponding algorithm (problem-solving method) applies. Recently we have expanded the algorithm hierarchy to include specialized problem-solving methods from Operations Research (e.g. specializations of linear programming) and applied them to the design of scheduling systems.

In summary, KIDS allows users to add domain-specific models to the system. Soon there will be a uniform capability for extending its programming knowledge by adding programming theories to the refinement hierarchy. We believe that these capabilities bring KIDS closer to being the "EMYCIN" of automatic programming than other systems.

7. Concluding Remarks

One general observation is that the complexity of handling this application arose from its bulk, not its depth. KIDS had to be upgraded in several ways to accomodate this bulk. For example, the inference problems that arose during development could be solved in a few steps, but the sheer quantity of contextual detail that was presented to the inference system overwhelmed it at first. In one early attempt at development we encountered branching factors of 65 in the inference system! Incorporating a simple congruence closure mechanism to handle equivalence classes of terms brought the branching factor down to under two on average. Similarly, we needed to implement generalized versions of the programming knowledge in KIDS, such as the divide-and-conquer tactic.

Our experience supports the notion that building good domain theories is the dominant-cost activity in developing conventional software. KIDS provides some support for this theory-development activity and provides state-of-the-art support for subsequent steps for developing specifications in the domain theory into code. Our experience with the Track Assignment problem has provided further evidence of the essential soundness of the KIDS model of software design (as described in Section 2) and that the KIDS operations support the development of conventional code at a usable level of automation. This derivation also provides yet more evidence that theory and specification development will dominate future software development activity.

Acknowledgements

I would like to thank Tom Pressburger for his many contributions to the evolving KIDS system. I would like to also thank LiMei Gilham for her efforts in implementing the theory development subsystem and a generalized divide-and-conquer tactic. This research was supported in part by the RADC Development Assistant Contract F30602-88-C-0127 and in part by the Office of Naval Research under Contract N00014-87-K-0550.

References

[1] BATORY, D. S., ET. AL. GENESIS: An extensible database management system. *IEEE Transactions on Software Engineering 14*, 11 (November 1988), 1711–1730.

[2] BLAINE, L., AND GOLDBERG, A. DTRE – a semi-automatic transformation system. In *Constructing Programs from Specifications*, B. Möller, Ed. North-Holland, Amsterdam, 1991, pp. 165–204.

[3] GRAVES, H. Lockheed environment for automatic programming. Tech. rep., Lockheed Palo Alto Research Center, Palo Alto, CA, 1990.

[4] JOHNSON, W. L., AND FEATHER, M. S. Using evolution transforms to construct specifications. In *Automating Software Design*, M. L. Lowry and R. McCartney, Eds. AAAI Press, Menlo Park, CA, 1991, pp. 65–92.

[5] KANT, E., DAUBE, F., MACGREGOR, W., AND WALD, J. Scientific programming by automated synthesis. In *Automating Software Design*, M. L. Lowry and R. McCartney, Eds. AAAI Press, Menlo Park, CA, 1991.

[6] LOWRY, M. R. Automating the design of local search algorithms. In *Automating Software Design*, M. Lowry and R. McCartney, Eds. AAAI/MIT Press, Menlo Park, 1991, pp. 515–546.

[7] SETLIFF, D. E. On the automatic selection of data structures and algorithms. In *Automating Software Design*, M. L. Lowry and R. McCartney, Eds. AAAI Press, Menlo Park, CA, 1991.

[8] SMITH, D. R. Top-down synthesis of divide-and-conquer algorithms. *Artificial Intelligence 27*, 1 (September 1985), 43–96. (Reprinted in *Readings*

in Artificial Intelligence and Software Engineering, C. Rich and R. Waters, Eds., Los Altos, CA, Morgan Kaufmann, 1986.).

[9] SMITH, D. R. Applications of a strategy for designing divide-and-conquer algorithms. *Science of Computer Programming 8*, 3 (June 1987), 213–229.

[10] SMITH, D. R. Structure and design of global search algorithms. Tech. Rep. KES.U.87.12, Kestrel Institute, November 1987. to appear in *Acta Informatica*.

[11] SMITH, D. R. KIDS: A knowledge-based software development system. In *Automating Software Design*, M. Lowry and R. McCartney, Eds. MIT Press, Menlo Park, 1991, pp. 483–514.

[12] SMITH, D. R. Structure and design of problem reduction generators. In *Constructing Programs from Specifications*, B. Möller, Ed. North-Holland, Amsterdam, 1991, pp. 91–124.

[13] SMITH, D. R. Track assignment in an air traffic control system: A rational reconstruction of system design. Tech. Rep. KES.U.91.9, Kestrel Institute, July 1991.

[14] SMITH, D. R. KIDS – a semi-automatic program development system. *IEEE Transactions on Software Engineering Special Issue on Formal Methods in Software Engineering 16*, 9 (September 1990), 1024–1043.

[15] SMITH, D. R., AND LOWRY, M. R. Algorithm theories and design tactics. *Science of Computer Programming 14*, 2-3 (October 1990), 305–321.