

## Specification and Analysis of Data for Safety-Critical Systems

J. C. Knight; University of Virginia; Charlottesville, Virginia

E. A. Strunk; University of Virginia; Charlottesville, Virginia

W. S. Greenwell; University of Virginia; Charlottesville, Virginia

K. S. Wasson; University of Virginia; Charlottesville, Virginia

Keywords: data specification, safety-critical systems, data-driven systems

### Abstract

Data-driven systems are those that make use of large quantities of data to perform their function. For a data-driven system that must also be dependable, ensuring the validity of data is just as crucial as ensuring that of code. Data for such systems are often developed separately from the systems themselves and, if not subjected to careful scrutiny, might not adhere to all of the assumptions made by the systems' developers. This paper describes a specification and verification technique for data that employs both formal and informal specification methods. This technique is illustrated using part of a real-world system: the Minimum Safe Altitude Warning (MSAW) system, a data-driven system that is used by the U.S. Federal Aviation Administration to detect and alert air traffic controllers to low-flying aircraft.

### Introduction

Many safety-critical systems make heavy use of predefined configuration data as part of the definition of system function. Storey calls these "data-driven systems" because such systems use large quantities of complex configuration data to perform their functions, and the data can be significantly different at each system installation [refs. 11, 12]. Ensuring the validity of data is just as crucial as ensuring that of code. The data are often developed separately from the systems themselves and, if not subjected to careful scrutiny, might not adhere to all of the assumptions made by the systems' developers. Interestingly, typical standards such as RTCA DO-178B [ref. 7] and RTCA DO-278 [ref. 9] do not make explicit provision for data assurance but focus entirely on code preparation.

As an example of a data-driven system, we look at the software systems installed in air-traffic-control towers. These systems perform essentially the same set of functions at each facility, but the specific calculations they make are dependent on local geographic and airport information and thus vary with each installation. Serious consequences can arise if this information is inaccurate, as evidenced by the accidents surrounding the U.S. Federal Aviation Administration's (FAA) Minimum Safe Altitude Warning (MSAW) system. This system detects and alerts air traffic controllers to aircraft either flying below a safe altitude or in danger of doing so; it is designed to help prevent what are referred to as controlled-flight-into-terrain accidents. Several such accidents have occurred in which the failure of MSAW to alert controllers was a causal factor [ref. 6], and in some of these cases, the problem was attributed to incorrect configuration data.

Typically, the specification of the data that a system will use is entangled with functional specification of the software itself. As Storey has pointed out, however, for systems where instantiation involves the preparation of extensive local data, a complete lifecycle for data is needed [ref. 11]. For safety-critical systems, this might include a variety of specialized techniques adapted for data such as hazard analysis in which particular data defects of special significance might be identified. In this paper we describe a technique for the specification phase of the data lifecycle. The purpose of data specification is to provide the data developer with all of the information needed to construct a properly-formatted data instantiation and to ensure that the instantiation is valid. By properly formatted, we mean that the fields of the data are in the proper location, of the right size, and use the proper encoding. By valid, we mean that the data represent accurately the real-world concepts that are intended.

Format can be described formally, and so in our technique we use formal languages for this purpose. These languages are amenable to mechanical analysis, and we have developed a tool that implements a wide variety of analyses. We refer to the tool as the data verifier since it establishes a number of important properties about a specific data instantiation. Validity, on the other hand, requires that real-world concepts and objects be defined precisely, and anything to do with meaning in the real world has to be documented in natural language. For safety-critical systems where precision and accuracy are paramount, this requires considerable care and attention because of the known difficulties with natural language. As part of a separate research project, we are developing general techniques for the elicitation and representation of context information with the goal of exploiting these techniques in a variety of areas, including software requirements specification [ref. 13]. In our data specification technique, we use some of these techniques to help define the data context.

We begin this paper by reviewing what is involved in data specification, and we then present our approach and the tool that we have developed for analysis of the resulting artifacts. To illustrate our technique, we have applied it to a small part of the current form of the configuration data for the FAA's MSAW system and we show how several critical properties of data can be defined and data instantiations can be checked. Finally, we present our conclusions.

### Data in Complex Systems

The type of data-driven system with which we are concerned is shown in figure 1. In such a system, the same software operates in different instantiations, and differences between instantiations are determined by the data that the software uses. Such systems are common, but we are concerned in this paper with such systems in safety-critical applications where erroneous data could have serious consequences.

The data used by any software system are a binary encoding of certain aspects of the artifact(s) of interest to the system's users. The binary encoding has a structure that defines the organization of the data, i.e., how the data are decomposed into fields, sections of files, and files. This structure is entirely artificial and is merely an agreement which ensures that different users of the data, i.e., different programs that manipulate the data, access the bits in a consistent way. A data field holding a runway length, for example, might be a 16-bit field, a 64-bit field, or anything that can contain the necessary scale. This notion can be described in mathematics in a way that is amenable to effective communication and mechanical analysis.

As well as the need to ensure that different software systems parse the binary content of a file identically, the binary encoding must reflect precisely what the system's users expect the bits to encode. Any misunderstanding between human users of the data (through programs that manipulate the data) could lead to software functionality that is not what is expected or desired. In the encoding of a runway length, for example, if one user thinks the encoding represents meters between two specified reference points and another thinks that the value represents yards or is based on different reference points, then the results are unlikely to be what was desired. This will be true irrespective of where within the data file the length value is stored and irrespective of its binary representation. The fact that the runway length is specified in meters cannot be formalized because the details of the scale along with its name, *metric*, are merely conventions that we all share. As such, all of the real-world meaning of the data, including details of the measurement scale used for runway length, must be defined carefully and completely and this has to be done using natu-

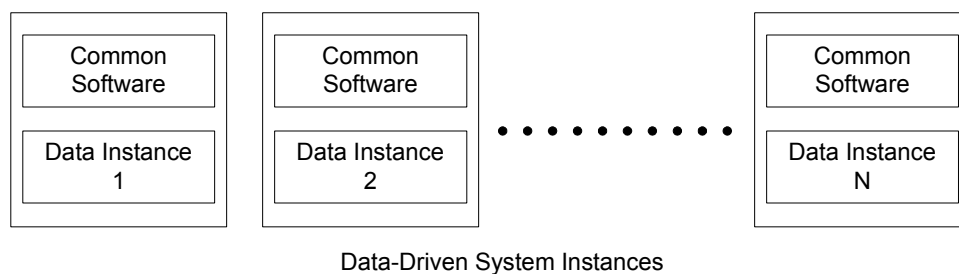


Figure 1 — Data-driven System Concept

ral language. Issues related to this topic have been discussed extensively by Jackson [refs. 4, 5]. In particular, Jackson's notion of a designation provides a mechanism for defining domain terms.

An encoding can map a real-world entity to its data representation using any transformation that is desired; for example, the mapping of a runway length measured in meters could be to a binary integer, a decimal integer or a character string, and could include a translation in which a constant value is added to actual values as a convenience. As with the structure of the data, the mapping is entirely artificial and, again, is merely an agreement between the different programs that use the data. In the runway length example, the length might be encoded in a data file as a 16-bit unsigned integer value corresponding to the measured length in meters divided by ten.

In some cases, real-world characteristics that restrict possible data values are present. The ending point of a runway, for example, must be at least some minimum distance from its starting point, and, on a broader scale, any runway that is specified to be monitored for some form of airspace violations must be included in the overall area being monitored. Such supplementary information facilitates reasonableness checks on data, and are important elements of a data specification.

### An Approach to Data Specification

In order to facilitate precise and accurate data specification and thereby avoid many of the mistakes that can occur with ad hoc approaches, we have developed a comprehensive data-specification technique. Our technique includes three elements in a specification:

*Data Syntax:* formal specification of the structure of the data.

*Data Context:* rigorous specification of the real-world entities that the data encode and the relationship between the real-world entities and their encoding.

*Data Semantics:* formal specification of the rules that limit data specifications to those which are acceptable to the application.

Data Syntax: Errors in the syntax of data can cause data values to be associated with wrong fields. While the syntax is formal and conceptually simple relative to the other specification components, its enumeration can be quite lengthy due to a potentially large number of fields that can be ordered in many ways. Its length can make it difficult to comprehend and analyze without the aid of a systematic method for doing so. Formally defining the syntax and checking it mechanically permits a high degree of assurance that all those involved with a data specification will have the same understanding of the structure of the data. We define the syntax of data using Backus-Naur Form (BNF) for the context-free structure of the data. This provides a precise and unambiguous formal notation<sup>1</sup> for the statement of data syntax.

Data Context: Errors in selecting and describing elements of the real world with which data are to be associated can allow misinterpretation and resulting misuse of such data. We assert that, in order to be able to make a valid association between a syntactic element and an element of the real world, there must be measures in place to ensure that the elements of the real world in question are uniformly accessible and understood properly by *all* required parties. To this end, we incorporate into our data specification technique some of the results of ongoing research aimed at increasing the validity of natural-language requirements and specifications in general. This research exploits linguistic and psychological theory to identify faults in the specification that result from miscommunication in its preparation and weaknesses in the specification that might lead to its being misunderstood [refs. 2, 3, 13].

As well as providing the rigorous documentation of the real-world entities to which the data applies, a specific mapping is needed from data elements to the real-world entities that the data encodes. This is provided in the specification of the data context by associating field names (symbols) from the syntactic specification with natural language explanations of what the field encodes.

---

1. Although it might unambiguously describe ambiguous grammars.

Data Semantics: In a similar vein to programming language semantics, the data semantics tell which syntactically valid specifications are to be allowed. Syntactically valid specifications might be disallowed because of restrictions inferred from the data's context; a data field might be restricted to certain value ranges, for example. There is no way to represent all of the data's contextual restrictions as formal semantics; there is not even a way to ensure that all of the safety-related restrictions are captured.

We specify the data's formal semantics in the PVS specification notation [ref. 10]. The PVS type mechanism allows us to define the properties of the data and rules about constraints through type instantiations and theorems requiring proof in an instantiation.

The syntax and formal semantics of a specification can be combined logically. The advantage of separating them is separating the concerns of how the machine will read the data and what can be expressed by the data. In other words, the formal semantics can be used to model some elements of interest from the context, whereas the syntax documents only the patterns of bits of the data file.

Overall Structure: All the elements of definition are present, *of necessity*, in the documentation associated with any data; if any one were missing, the data could not be used. The presence of these elements is usually in ad hoc documents that arise from necessity rather than design. Mistakes, therefore, are both likely and common, and mistakes in any element of the data specification can lead to failures of a system even if the software being used is, in fact, correct.

Although all three are required, the separate components of the specification can be combined in a number of different ways, depending on a user's needs. While writing the formalism, for instance, a specifier might not want to see the informal contextual elements—or the elements of the formalism that are not directly applicable, for that matter. In this case only the formalism of interest might be displayed. An inspector of the specification, on the other hand, would want to see all of the elements together to check their validity.

### Tool Support For Data Analysis

The formal languages used in our specification technique enable mechanical analysis, and the data verifier shown in figure 2 implements this analysis. The purpose of the data verifier is to read a data file that should comply with a particular specification and check the data syntax and formal data semantics of the file.

The primary component of the data verifier is the data parser. The parser reads a data file and checks compliance with the syntactic rules. For a syntactically correct data file, the parser outputs PVS text that documents a set of links between variables in the semantic component of the specification and values from the file. This is then appended to a PVS semantic specification containing a series of type declarations and theorems. Recall that the constraints in the semantic specification are documented as theorems in PVS. The theorems are read by the PVS theorem-proving system, which then attempts to prove them. If the proofs are created successfully, they provide assurance that the data in the file meets the constraints. Limited human support for the proof process is needed at the moment although we are in the process of increasing the degree of automation. The output of the data verifier is a report that documents the theorems that were proved. The user may then locate any remaining unproven theorems and either correct the data file or modify the proof strategy to prove them.

An important practical aspect of the design of the data verifier is that it operates with the same data specification that is used as a reference when creating data files. This presents the difficulty of providing convenient access for human specifiers and also convenient access for automated tools. This issue is dealt with by providing editing tools that facilitate human access. The two editors shown in figure 2 are presently ad hoc, and we intend to provide custom facilities for these functions in the future.

The data verifier itself must be customized for each specification. The overall architecture of the data parser is produced by Yacc and Lex directly from the BNF component of the specification. The code for the creation of the PVS semantics is produced by modifying a generic template. The PVS theorem-proving system is the same for each specification. The use of Yacc, Lex, and the PVS system provides the added benefit that extensive syntax and type check-

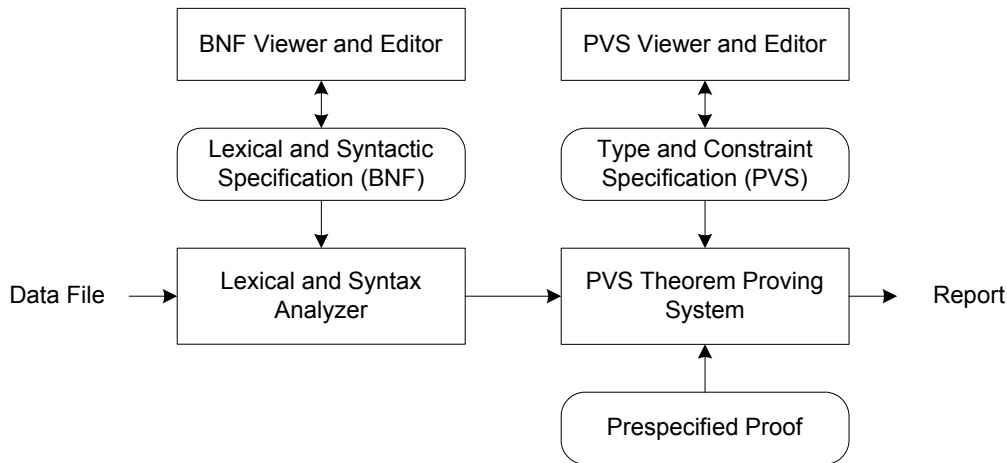


Figure 2 — Data Analysis Toolset

ing is performed on the syntactic and semantic components of a data specification. Yacc, for example, performs many checks on an input grammar, and the PVS system checks the PVS syntax and the implications of the type definitions in the data semantics.

The major improvement that the data verifier offers over traditional data specification and verification methods is the potential to mechanically show properties of interest. Traditionally, a human would create the file with the data and then someone would verify the numbers, either by inspecting the numbers themselves or by visualizing the numbers in some way. While there is no substitute for validity checking performed by humans, mechanical translation and property proofs are a powerful asset in catching subtle flaws.

Such a process also offers significant benefits to regulators. A regulatory authority could look at the PVS output and check that all required properties were proved (and run the proofs him or herself, if desired). This is a major improvement over the current need to either inspect the files manually through a time- and labor-intensive process, or to trust the organization that created them to have completed its task correctly.

### An Example of Data Specification

In this section, we summarize a small, partial specification for the configuration data for the FAA’s MSAW system that we have developed to illustrate our data specification technique. Even though the example is small, several parts are not elaborated here because of their length. The current specification used by the FAA uses a systematic but informal structure. The details of the configuration data are defined partly in an FAA document entitled *Standards and Guidelines to Define and Adapt Values for Minimum Safe Altitude Warning and Conflict Alert Site Variables* [ref. 1] and partly as comments in the configuration data file itself. We have consolidated this information in our example.

The MSAW configuration file contains the information required by the MSAW system to provide low-altitude monitoring service. The file is organized as a series of specification tables with each table specifying details such as airport and runway layouts, local terrain elevations, MSAW monitoring zone definitions, and MSAW inhibit areas. For a prototypical regional airport, the file spans 26,530 lines, of which 17,225 are comments and 985 are blank. In the remaining 8,320 lines, the file defines 223 tables of which 105 are empty. The longest table defined in the file is a terrain map occupying 6,806 lines. The portion of the configuration file we chose to study describes how to document aural alarm inhibit areas. We developed our partial specification primarily by direct translation from FAA materials. Our specification has not been otherwise validated.

The example in this section is presented as it might be for inspection. We have chosen to break up the organization primarily according to the BNF syntactic structure, so that the major components of the BNF grammar are shown together with their corresponding semantic and contextual elements. A complete specification using our technique

would consist of a set of specification tables where the form and order of these tables is based on the syntactic structure of the data. In this example, the specification begins with the syntactic definition of the MSAW configuration file as a set of data tables where each data table begins with the keyword “begin” followed by the table name and ends with the keyword “end”. This is shown in Table 1.

<b>Names</b>	MSAWFile, MSAWTable	<b>T A B L E  1</b>
<b>Syntax</b>	MSAWFile ::= (“begin” MSAWTable “end;”)+ MSAWTable ::= “CAM_AURAL_IFR” CAM_AURAL_IFR   “CAM_MCI_INHIB” CAM_MCI_INHIB   “CAM_AURAL_REM” CAM_AURAL_REM	
<b>Semantics</b>	MSAWFile : TYPE = [# CAM_AURAL_IFR_tables: finseq[CAM_AURAL_IFR], CAM_MCI_INHIB_tables: finseq[CAM_MCI_INHIB], CAM_AURAL_REM_tables: finseq[CAM_AURAL_REM] #]	
<b>Context</b>		
<i>Symbols</i>	MSAWFile                   : Set of properties used to configure MSAW. MSAWTable                  : Data associated with a specific configuration property. CAM_AURAL_IFR              : MSAW data table defining an <u>IFR aural alarm inhibit area</u> . CAM_MCI_INHIB              : <i>(not elaborated here)</i> CAM_AURAL_REM              : <i>(not elaborated here)</i>	
<i>Defn’s</i>	IFR                         : Instrument Flight Rules: circumstances under which pilot controls altitude, attitude, and course using flight instruments. aural alarm inhibit area   : Geometric shape within which <u>aural alarms</u> are disabled. aural alarm                 : Sound generated in monitoring center intended to draw controller’s attention to detected altitude violation.	

The syntax specifies that (for the purposes of this example) an MSAW data table is either a CAM\_AURAL\_IFR, a CAM\_MCI\_INHIB, or a CAM\_AURAL\_REM data table. For brevity, we focus on the CAM\_AURAL\_IFR table—it is used to define a special type of MSAW inhibit area in which aural MSAW low-altitude alarms are to be suppressed but visual alarms should still be displayed. The high-level specification for the CAM\_AURAL\_IFR data table is shown in specification Table 2.

Each row of the CAM\_AURAL\_IFR table defines an aural alarm inhibit area and consists of two boolean fields indicating whether the inhibit area is applicable to MSAW or to the conflict alert system, respectively, and a REGION\_GEOMETRY field defining the geographic region to which the inhibit area applies. Geographic regions may be defined in a variety of ways using three different coordinate systems, and so the REGION\_GEOMETRY field warrants its own specification, presented below in Table 3. Again for brevity, we only include the specification for the latitude/longitude coordinate system.

There are several possible shapes available for specifying the geometry of each region. We have chosen one that illustrates the potential complexity of such data files: the 3-range, 3-azimuth shape, specified in latitude/longitude coordinates. The specification for this shape is given below in Table 4.

We end our example with the structure holding the ranges and azimuths for the region, specified in Table 5. The reader can gain a more thorough understanding of what the specification means by reading the context portion accompanying each syntactic element.

In many cases, the reader will notice that the semantic definition in PVS mirrors the syntactic definition in BNF. This is unavoidable when using two different languages because the basic syntax must also be included with the semantics. We note, however, that some of the PVS types are much stronger than an enumeration of the possible lexical

<b>Names</b>	CAM_AURAL_IFR, CAM_AURAL_IFR_Row	<b>T A B L E  2</b>
<b>Syntax</b>	CAM_AURAL_IFR ::= (CAM_AURAL_IFR_Row ;)+ CAM_AURAL_IFR_Row ::= C_AUR_LA_CA_IFR_1 C_AUR_LA_CA_IFR_2 REGION_GEOMETRY	
<b>Semantics</b>	CAM_AURAL_IFR: TYPE = [# rows: finseq[CAM_AURAL_IFR_Row] #] CAM_AURAL_IFR_Row: TYPE = [# C_AUR_LA_CA_IFR_1 : bool, C_AUR_LA_CA_IFR_2 : bool, REGION_GEOMETRY_35: REGION_GEOMETRY #]	
<b>Context</b> <i>Symbols</i>	CAM_AURAL_IFR_Row : Row in <u>IFR aural alarm inhibit area</u> table. C_AUR_LA_CA_IFR_1 TRUE : <u>IFR aural alarm inhibit area</u> is applicable for <u>MSAW</u> . FALSE : <u>IFR aural alarm inhibit area</u> is not applicable for <u>MSAW</u> . C_AUR_LA_CA_IFR_2 TRUE : <u>IFR aural alarm inhibit area</u> is applicable for <u>conflict alert</u> . FALSE : <u>IFR aural alarm inhibit area</u> is not applicable for <u>conflict alert</u> . REGION_GEOMETRY : <u>geometric data</u> .	
<i>Defn's</i>	geometric data : Formal representation of the <u>aural alarm inhibit area</u> .	

<b>Name</b>	REGION_GEOMETRY	<b>T A B L E  3</b>
<b>Syntax</b>	REGION_GEOMETRY ::= POLY_REGION   RECT_REGION   CIRC_REGION   <b>RNG3_REGION</b>   RNG6_REGION   RNG2_REGION	
<b>Semantics</b>	geometry_regions: TYPE = {POLY_REGION, RECT_REGION, CIRC_REGION, RNG3_REGION, RNG6_REGION, RNG2_REGION} geometric_data : TYPE = [# REGION_TYPE: GEOMETRY_REGIONS, POLY: POLY_REGION, RECT: RECT_REGION, CIRC: CIRC_REGION, RNG3: RNG3_REGION, RNG6: RNG6_REGION, RNG2: RNG2_REGION #]	
<b>Context</b> <i>Symbols</i>	POLY_REGION : (not elaborated here) RECT_REGION : (not elaborated here) CIRC_REGION : (not elaborated here) <b>RNG3_REGION</b> : Area defined by a <u>center point</u> and the union of 3 <u>sectors</u> . RNG6_REGION : (not elaborated here) RNG2_REGION : (not elaborated here) REGION_TYPE : Which element of the semantic definition must contain valid values (necessary to maintain type consistency).	
<i>Defn's</i>	center point : Point specified with a latitude/longitude pair. sector : Pie-shaped region defined by a <u>range/azimuth</u> pair. See Figure in Table 5. range : Length in <u>nautical miles</u> from a prescribed <u>center point</u> in a prescribed direction defined by an <u>azimuth</u> . azimuth : Angle in degrees, measured clockwise in the horizontal plane, between true North and a ray originating from a prescribed <u>center point</u> . nautical mile : Linear distance of 6076.115 feet.	

<b>Names</b>	RNG3_REGION, RNG3_LATLON_REGION	<b>T A B L E  4</b>
<b>Syntax</b>	RNG3_REGION ::= RNG3_LATLON_REGION   RNG3_CART_REGION   RNG3_POLAR_REGION RNG3_LATLON_REGION ::= ALTITUDE_RNG LATLON_COORD RNG3_AZIMUTH3	
<b>Semantics</b>	RNG3_GEOMETRY : TYPE = {rng3_latlon, rng3_cartesian, rng3_polar} RNG3_REGION : TYPE = [# region_type : rng3_geometry, latlon : RNG3_LATLON_REGION, cartesian : RNG3_CART_REGION, polar : RNG3_POLAR_REGION #] RNG3_LATLON_REGION : TYPE = [# altitude : ALTITUDE_RNG, center : latlon_coord, shape : rng3_azimuth3 #]	
<b>Context Symbols</b>	RNG3_LATLON_REGION : Type of RNG3_REGION located by a <u>center point</u> , that has a specified range of applicable altitudes, and for which shape is defined by three <u>ranges</u> and three <u>azimuths</u> (see Table 5). RNG3_CART_REGION : <i>(not elaborated here)</i> RNG3_POLAR_REGION : <i>(not elaborated here)</i> ALTITUDE_RNG : Minimum and maximum altitudes between which points in the <u>aural alarm inhibit area</u> will be monitored. LATLON_COORD : <u>center point</u> of RNG3_LATLON_REGION RNG3_AZIMUTH3 : Sequence of three <u>ranges</u> and three <u>azimuths</u> that define the sectors of the RNG3_LATLON_REGION RNG3_GEOMETRY : Which element of the semantic definition must contain valid values <i>(necessary to maintain type consistency)</i> .	

tokens for the BNF production, and also that the PVS theorems offer semantic constraints that cannot be reproduced in BNF.

Observations: The original FAA document that we used in our example has been carefully constructed and reviewed by many experts over several years. It exhibits the overall consistency and completeness that one would expect from such a document, even though these properties have not been checked mechanically before. There are, however, a few things we found that are either inconsistent or, we feel, could provide a ready opportunity for mistakes by people who are not intimately familiar with the document.

The only inconsistency we found was in the specification for encoding a 3-range, 3-azimuth geometric region (such as defined by RNG3\_AZIMUTH3, above). The allowed range of values, based on the encoding sent to the software, is 0 - 4096; but there is a special-case option to use the value 5000, outside of that range, which indicates a different region shape. Use of a value of 5000 in a specification would be flagged by our data verifier unless a change was made to reflect this option in the allowed range of values. This is a simple example of how mechanical analysis can catch inconsistencies and errors that are difficult for humans to spot.

The shape of a 3-range, 3-azimuth region in general was a puzzle for us as novices. The shape is nonstandard, not intuitive, and not documented in the FAA specification (although it is addressed in the data files themselves). The FAA has built a tool that will visualize this region, which is very helpful; but the tool is not part of the specification. The FAA is trying to phase out this type of region as they update the software, but it does currently exist in operation.

We noted two other potential threats to common understanding. The first was inconsistency in the direction of reference for angular measurement. For some types of region, the direction of reference is *true* North; for others, it is *magnetic* North. The second was the length of a nautical mile. There are, in fact, two slightly different definitions but

<b>Name</b>	RNG3_AZIMUTH3
<b>Syntax</b>	RNG3_AZIMUTH3 ::= DECIMAL DECIMAL DECIMAL INT INT INT
<b>Semantics</b>	<pre> rng3_azimuth3 : TYPE =   [# range1   : real,    range2   : real,    range3   : real,    azimuth1 : int,    azimuth2 : int,    azimuth3 : int #] </pre>
<b>Context</b>	
<i>Symbols</i>	<p>DECIMAL : Fixed-point, base-10, real number.</p> <p>INT : Integer.</p> <p>range1, 2, 3 : Three <u>ranges</u> defining sectors 1, 2, 3, respectively, of the RNG3_LATLON_REGION (See Figure)</p> <p>azimuth1, 2, 3 : Three <u>azimuths</u> defining the span of sectors 1, 2, 3, respectively, of the RNG3_LATLON_REGION (See Figure)</p>
	<p style="text-align: right;">Azimuth<sub>i</sub> is the angle between True North and range<sub>i</sub></p>

which was expected was not specified. In each of these cases, someone familiar with the system might use the appropriate measure as second nature, but someone new to it might not.

### Conclusions

The dependence of many safety-critical systems on extensive local data is significant, and errors in the data can be just as serious as those in the software. A comprehensive treatment of data is important, and a data lifecycle is one way that such a treatment might be arranged.

An important aspect of such a lifecycle is data specification. We have presented a technique for the specification of data for data-driven, safety-critical systems. The technique uses formal languages for those elements that are amenable to formal specification and a rigorous approach to natural language for the remainder. The formal languages are supported by a data-analysis toolset, and the natural language is supported by methods that guide high quality representation of domain information. Using an example from the air-traffic-control domain (MSAW), we have illustrated the use of the approach and the capabilities that it provides for analysis.

Our experience with the MSAW data specification suggests that the technique we have developed can be applied to the type of application with which we are concerned. The tools that we have developed provide assurance that the constraints imposed by the application domain are met.

## Acknowledgements

We are extremely grateful to Gary Dyer of the FAA for his extensive help in understanding the MSAW data configuration file. This work was funded in part by NASA Langley Research Center under grants numbered NAG-1-2290 and NAG-1-02103.

## References

1. Federal Aviation Administration, Common ARTS: Standards and Guidelines to Define and Adapt Values For Minimum Safe Altitude Warning (MSAW) And Conflict Alert (CA) Site Variables, A6.05/A2.09, October 2003.
2. Hanks, K. and J. Knight. *Improving Communication of Critical Domain Knowledge in High-Consequence Software Development: An Empirical Study*. 21st International System Safety Conference, Ottawa, Canada, August 2003.
3. Hanks K., J. Knight, and E. Strunk. *Erroneous Requirements: A Linguistic Basis for Their Occurrence and an Approach to Their Reduction*. Software Engineering Workshop, NASA Goddard Space Flight Center, December, 2001.
4. Jackson, M. Software Requirements Specification. Addison Wesley, 1995.
5. Jackson, M. Problem Frames. Addison Wesley, 2001.
6. National Transportation Safety Board. Controlled Flight Into Terrain, Korean Air Flight 801, Boeing 747-300, HL7468, Nimitz Hill, Guam, August 6, 1997. Aircraft Accident Report NTSB/AAR-00/01. Washington, DC, 2000.
7. RTCA. Software Considerations in Airborne Systems and Equipment Certification. Document RTCA/DO-178B. Washington, DC: RTCA, December 1992.
8. RTCA. Standards for Processing Aeronautical Data. Document RTCA/DO-200A. Washington, DC: RTCA, September 1998.
9. RTCA. Guidelines for Communication, Navigation, Surveillance, and Air Traffic Management (CNS/ATM) Systems Software Integrity Assurance. Document RTCA/DO-278. Washington, DC: RTCA, March 2002.
10. SRI International, The PVS Specification and Verification Systems. <http://pvs.csl.sri.com/>
11. Storey, N. and A. Faulkner. *Data—The Forgotten System Component?* Journal of System Safety, Vol. 39, No. 4 2003.
12. Storey, N. and A. Faulkner. *The Characteristics of Data in Data-Intensive Safety-Related Systems*. 22nd Int. Conf. on Computer Safety and Reliability, Safecomp 2003, Edinburgh, September 2003, pp. 396-409.
13. Wasson, K. *Partial Reductive Paraphrase: Toward More Transparent Requirements*. University of Virginia, Department of Computer Science, Technical Report CS-2004-16, May 2004.

## Biography

John C. Knight, Professor of Computer Science, University of Virginia, 151 Engineer's Way, P.O. Box 400740, Charlottesville, VA 22904-4740, USA, voice - 434.982.2216, fax - 434.982.2214, e-mail - knight@cs.virginia.edu

Elisabeth A. Strunk, Ph.D. candidate in Computer Science, University of Virginia, 151 Engineer's Way, P.O. Box 400740, Charlottesville, VA 22904-4740, USA, voice - 434.982.2225, fax - 434.982.2214, e-mail - strunk@cs.virginia.edu.

William S. Greenwell, Ph.D. candidate in Computer Science, University of Virginia, 151 Engineer's Way, P.O. Box 400740, Charlottesville, VA 22904-4740, USA, voice - 434.982.2225, fax - 434.982.2214, e-mail - greenwell@cs.virginia.edu.

Kimberly S. Wasson, Ph.D. candidate in Computer Science, University of Virginia, 151 Engineer's Way, P.O. Box 400740, Charlottesville, VA 22904-4740, USA, voice - 434.982.2225, fax - 434.982.2214, e-mail - ksh4q@cs.virginia.edu.