Strategies and methods for ensuring adequate system safety have developed in recent years to adapt to the ever-increasing complexity of safety-critical systems. One of these new methods is the safety case with a rigorous safety argument documented in the Goal Structuring Notation (GSN). GSN has been shown to improve structure, clarity, and rigor of safety arguments, and to support better safety case maintenance. Despite the advantages of GSN, safety cases using GSN can still be difficult to manage. For real systems that need to integrate heterogeneous systems into a whole, GSN arguments tend to be large, complex and difficult to reason about. This paper presents ACCESS, a toolset designed to support creation, inspection, validation, maintenance, and other activities related to GSN that helps engineers document their belief in the adequacy of the safety of their system.

Introducing ACCESS

In this paper, we introduce the Assurance Case Construction and Evaluation Support System (ACCESS). ACCESS is a software toolset that aids in the development, maintenance, and analysis of safety cases, especially those with large and complex arguments. A safety case is a collection of documents that “should communicate a clear, comprehensive and defensible argument that a system is acceptably safe to operate in a particular context” (ref. 6). A safety case consists of a safety argument that demonstrates how to reasonably conclude that a system is adequately safe, and evidence referenced by the safety argument. Engineers construct a safety argument in order to explain how the evidence presented with the argument supports the conclusion that their system is adequately safe.

As the complexity of safety-critical systems has increased, the usage of safety cases has become increasingly prevalent (ref. 3). Several European government agencies require safety cases for their systems, including the U.K. Civil Aviation Authority (ref. 2) and Ministry of Defence (ref. 4), while others recognize the importance of developing and documenting safety cases (refs. 5, 9).

Without a way of presenting a safety argument to reviewers, the argument would be of little value. There are multiple methods for presenting safety arguments, including free text, tabular structures, and others (ref. 6). Each has its advantages and disadvantages; ACCESS focuses on the Goal Structuring Notation (GSN).

GSN is a graphical notation that documents safety arguments. A GSN diagram includes nodes that represent goals, argument strategies, contexts, assumptions, justifications, and evidence. The structure of a GSN diagram shows explicitly the logical flow of a safety argument, the role of evidence in supporting the argument, and the rationale driving the argument (ref. 6). Safety arguments are frequently documented using GSN (ref. 14).

Using GSN to develop and codify safety arguments for complex modern safety-critical systems results in large goal structures, often with hundreds of nodes. While clearer than methods such as plain text, large GSN arguments are still difficult to create, to reason about, and to maintain manually (ref. 7). Engineers would benefit from being able to quickly and effectively create, manipulate, and maintain large safety arguments, including the use of safety case patterns (refs. 6, 11). The goal of ACCESS is to support the development, maintenance, and analysis of safety cases that include arguments documented in GSN, and the toolset contains a number of features that are specifically designed with large arguments in mind.

Motivation for Using GSN

GSN is used in industry to document safety arguments (ref. 14); according to Heimdahl, “[r]ecent work on Goal Structuring Notation (GSN) is a direction that promises to assist in the construction of rigorous arguments” (ref. 15). GSN’s limited graphical structures and defined semantics impose rigor on the creation process and encourage the engineer creating a safety argument to think systematically. Defining a safety argument using free text can easily
yield poor-structure and imprecise content (ref. 6). Graphical representations, in general, lend themselves well to providing structure, and they tend to simplify and organize the thoughts that go into a safety argument.

Because a GSN argument has a clear logical flow (ref. 6), incremental additions are easier to place. Recent safety standards, such as the U.K. Defence Standards 00-56 (ref. 12), require the incremental development of safety cases. With free text, safety engineers creating a safety case would have to sift through potentially hundreds of documents to try to find and change any documents relevant to whatever incremental change they were recording each time they needed to make a change.

Kelly and Weaver write on the importance of safety case patterns in safety case development (refs. 6, 11). Safety case patterns follow the same general line of thought as classical design patterns (ref. 13) in software engineering: a safety case pattern is a general, reusable way of arguing about a commonly-occurring situation in safety case construction. Safety case patterns document successful safety argument structures for reuse and are partial solutions to the safety case: they take care of one specific aspect of argument development in a large safety case.

However, even with the improved structure, clarity and rigor of GSN, large arguments are still difficult to work with. A safety case for a modern system may have hundreds or even thousands of GSN nodes in its safety argument. Manipulation of GSN arguments of such size can be daunting. The inclusion of patterns in arguments adds to the complexity: safety case developers must know which parts of an argument are patterns (so as not to disturb the individual elements of a pattern) and which are not.

**Design**

ACCESS is a Microsoft Windows application, built using the .NET Framework. It uses Microsoft Visio to display and edit graphics and Microsoft Word to view related documents. ACCESS supports the full lifecycle development of a safety case, from the original creation, maintenance, and inspection of the case. ACCESS has core features for the rapid prototyping and creation of safety arguments, argument maintenance tools, and an inspection framework. The tools discussed below show how ACCESS achieves its goal of providing engineers with a toolkit for managing the life cycle of a safety case.

**Implementation**

Safety cases in ACCESS are a collection of files: Visio diagrams for safety arguments, and Word documents for other relevant documents, e.g. evidence. ACCESS collects and organizes all safety-case files and documents in one location. Once a safety case is created in ACCESS, engineers can re-open a project and resume their creation, management, or inspection of a current safety case.

ACCESS provides tools that help organize the documents associated with the safety case and help build the associated argument in GSN. Creating or opening a safety case presents users with the main project window (Figure 1), where the user can add and edit safety case documents and the safety argument.

**Creation:** For safety case creation, ACCESS has rapid prototyping, node naming, pattern manipulation, and node coloring capabilities.

Rapid Prototyping: Developing compelling safety arguments is often a process of trial and error. A graphical safety argument might be created in GSN and found to be in need of change either during argument development or as a result of a review. Desired changes often only become clear once part of the argument has been written. Being able to build an argument quickly so as to facilitate insight is an important capability.

To enable trial and error, ACCESS provides a GSN prototyping mechanism. When prototyping a GSN argument, nodes are added using keyboard shortcuts and positioned using a simple positioning algorithm based on the node currently selected and the new node. For example, a goal node is added to an argument by entering the character “G” and this node remains selected. The new goal node will be placed beneath a strategy if a strategy node was selected, or adjacent to a different goal node if that was selected. The new node is connected with the proper arrow type to the proper node accordingly. Where multiple goals are derived from a single strategy, they are spaced evenly below the strategy automatically. The user can change the selected node by clicking on the desired node.
Figure 1 — The core of the ACCESS application is the safety case editor. This is where safety engineers can start constructing a safety case, or continue one that they have already started. The menu at the bottom, outside the window, is the result of clicking the ChangeSelectionColor menu.

Node naming (Figure 2) is completed automatically using a pattern that the user specifies as a set of three parts. For example, if the user wants all goal nodes to be labeled with the letter “G”, a string identifying the associated system or subsystem, and a sequence number, ACCESS generates the labels from the core system.

Figure 2 — The node naming feature allows users to quickly name or rename a collection of nodes in an argument (left window) with a pre-defined naming scheme using the right window. Users are also able to individually name or rename nodes and edit node summaries.
Using the node naming tool, engineers can also enter the brief text that is usually included in a GSN node. For example, an engineer could change the text for node Subsystem.2.Transducers from “<Summary>” to “The pilot movement transducers are acceptably safe” with the node naming tool.

Pattern Manipulation: When building safety cases, engineers often encounter situations that present an opportunity to reuse a previously-successful part of an argument, i.e. a safety case pattern (refs. 6, 11). These patterns may be of the engineers’ own design, or already established patterns. The capability to retrieve a pre-defined safety case pattern and use it in the correct context in an argument provides: (a) rapid argument creation; and (b) the ability to benefit from the engineering insight and experience of others codified in the pattern.

The ACCESS pattern manipulation tool allows users to search for, view, and import safety case patterns into arguments. The pattern manipulation tool displays both the graphical notation for the pattern and its documentation (Figure 3). Users can easily reuse a pattern they know to be valid, as long as the pattern and its corresponding documentation are imported into the pattern library.

![Figure 3 — The pattern manipulation tool. Users can select from a number of different patterns; the pattern library then displays the corresponding documentation in the top left-hand corner, and the mini-Visio window allows users to copy and paste the pattern into the current safety argument. Here, the user has selected and imported the “Absence of Omission Hazardous Failure Mode” pattern into the open safety argument.](image)

Node Coloring: Modern safety-critical systems incorporate many heterogeneous subsystems. For example, an aircraft’s flight control systems might include pilot movement transducers, computer hardware, computer software, digital data buses, hydraulic pumps, and hydraulically operated actuators. To argue the safety of the entire system, engineers must argue the safety of the individual subsystems. In a large argument, the distinctions between subsystems can be difficult to visualize. To provide a way to add visual distinction to different parts of an argument, ACCESS provides the node coloring mechanism. Users can define meanings for various color groups and apply color to any set of nodes.

**Development and Maintenance:** To aid engineers in safety case development and maintenance, ACCESS provides annotation capabilities and an argument summary tool.

**Annotation:** During the lifetime of a safety case, engineers may want to record information about the argument, e.g. thoughts about a change to the structure of the argument or questions about a specific goal. The ACCESS annotation
tool (Figure 4) is a mechanism for recording an author or reviewer’s comment. Annotations are associated with a specific node, are typed, and report generators can be added to ACCESS to create summary reports of annotations.

Figure 4 — The argument annotation tool. Users can add and edit various types of comments for the safety argument.

Argument Summary: The size of an argument for a safety-critical system can become quite large as the argument is developed. While creating and reviewing an argument, an engineer can become overwhelmed by the number of visible nodes. ACCESS includes a tool that highlights important aspects of a large argument to aid navigation and comprehension. Shown in Figure 5, the argument summary tool gives a detailed overview of the argument that is currently open and presents a selectable summary of the argument beside other information pertaining to the nodes of the argument.

Figure 5 — The argument summary tool. On the left of the window is the scaled view of a sample argument, and on the right are options for reviewing the structure and content of the argument. This example shows the highlighting of the goals associated with the selected context.
Summaries are subsets of nodes with particular properties. A node context summary, for example, is a display of a subset of the argument in which all of the contexts that are in-scope for a selected node are shown. The rest of the argument is omitted. A node context summary helps both development and review of arguments by providing an engineer with the complete context for a node quickly and easily.

The opposite of the node context summary is the context node summary. This summary displays all the nodes that are with the scope of a selected context. Other summaries include traces from a selected node to the top-level claim in which all the intermediate nodes are displayed but all other nodes are omitted.

Inspection: ACCESS provides a framework that supports the design and conduct of inspections of large safety arguments. The inspection tool (Figure 6) presents the safety argument in a format that can easily be inspected. Inspections are driven by checklists that can be set by the user so that they can be tailored to the needs of an organization or a

Figure 6 — On the top is an example argument. On the bottom is the inspection tool, which is used to examine the contents of each node and the argument as a whole.
specific safety project. Inspectors can navigate through a hierarchical, tree-like textual representation of the graphical argument (a tree-list). This representation of the argument has the advantage that branches can be expanded or collapsed according to the part of the argument upon which the inspector wants to focus. When selecting nodes in the tree-list, the information for the node is automatically populated on the right side of the tool, enabling the inspector to easily examine the contents of each node.

Nodes in the tree-list can be marked Incorrect, Needs to be Fixed, Review Later, or Complete, and each marking has its own color. These colors enable easy location of marked nodes when the argument is corrected. Finally, inspectors can add comments to nodes to guide subsequent repair.

**Evaluation**

To get a preliminary evaluation of ACCESS, we applied it to a large synthetic argument. The synthetic argument was created by merging several copies of a single pattern and ended up with 325 nodes (Figure 7). This argument was not intended to be complete or valid, merely large enough to test ACCESS’ argument manipulation facilities. This first construction step provided an evaluation mechanism for the pattern tool and yielded a large argument for subsequent tests.

**Pattern Manipulation Tool:** To evaluate pattern manipulation, we used the pattern tool to locate the target pattern, select the pattern, copy the pattern, locate the insertion point in the evolving synthetic argument, and paste the pattern into the that argument. This process was repeated 13 times with a pattern containing 25 nodes to yield a synthetic argument with 325 nodes. All of these various manipulations were completed in approximately five minutes.

![Figure 7 — The 325-node argument used during the evaluation.](image)

To provide a comparison, we estimated the time needed to recreate this same argument by dragging and dropping shapes, including arrows, from the stencil (manual re-creation time). The pattern has 25 nodes and 22 arrows, for a total of 611 placements from the stencil. Assuming a placement time of 10 seconds per placement, the manual re-creation time is 6,110 seconds, or about 1 hour and 48 minutes: approximately twenty times slower than with the pattern manipulation tool. While the total placement time of 5 minutes is artificially fast (given that we were only using one pattern), this rate demonstrates the feasibility of the pattern manipulation tool.

**Rapid Prototyping:** To evaluate the rapid prototyping mechanism, we considered the case where an engineer would need to quickly lay down a substantial goal structure to try out a certain idea (the trial and error situation). Using the prototyping tool to add a shape and its corresponding arrow to the large argument took at most 5 seconds. We estimate the time needed to add a node by dragging and dropping shapes, including arrows, from the stencil (manual re-creation time) to be at least 10 seconds per shape per placement in Visio.
A reasonable size for a substantial trial goal structure is 50 nodes. Assuming the number of arrows is approximately equal to the number of nodes, the estimate for the rapid prototyping tool is 250 seconds, or just over 4 minutes. For manual re-creation, the number of shapes dropped is 100; thus, the manual re-creation time estimate is 1000 seconds, or approximately 17 minutes; four times slower than ACCESS.

Node Naming: To test the node naming, we divided the synthetic argument into three distinct groups: top, middle, and bottom. With the node naming tool, it took approximately 3 minutes to name the entire argument. If we assume naming one node without the tool, i.e., using Visio text manipulation, takes 30 seconds, naming the entire 325-node argument would take about 2 hours and 42 minutes, i.e., fifty-four times slower than ACCESS.

Node Coloring: To evaluate node coloring, we separated the synthetic argument into the same three groups: top, middle, and bottom. Node coloring using ACCESS took about 3 seconds per group. Node coloring using Visio’s Fill Color took about 6 seconds per group.

Argument Annotation Tool: The goal of the annotation tool was to be able to record comments by viewers of the safety argument. The annotation tool logs the user, date, time, comment, and comment type for each annotation, preserving both the comment and its context for future viewers. By contrast with manual argument annotation, the annotation tool provides a homogenized annotation format and links (in ACCESS) to the context of the annotation. This functionality streamlines annotations substantially.

Inspection Tool: The inspection tool aids inspection by organizing the data in the safety argument into a format that is simple and quick to navigate. The advantages of using the inspection tool in comparison to conducting a manual inspection are summarized in Table 1.

<table>
<thead>
<tr>
<th>Manual Inspection Problems</th>
<th>Inspection Tool Solutions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Difficult to navigate structure, determine what to inspect</td>
<td>Collapsible/expandable tree-list representation of argument</td>
</tr>
<tr>
<td>Ad hoc methods of keeping track of whether a given node is valid, whether an argument is valid (holistically)</td>
<td>Node checklist, global checklist</td>
</tr>
<tr>
<td>Difficult to keep track of which nodes have already been inspected, which nodes to fix, etc.</td>
<td>Color-coded Mark states: Incorrect, Needs to be Fixed, Review Later, or Complete</td>
</tr>
<tr>
<td>Various pieces of information for inspection must be manually organized</td>
<td>All information organized and presented in one place</td>
</tr>
</tbody>
</table>

The inspection tool deals with four major manual inspection issues. Most importantly, the organization and presentation of data that the inspection tool provides makes the ACCESS inspection tool a complete environment for inspection of safety arguments.

Argument Summary Tool: The argument summary tool provides a set of novel features that allow engineers to see summary information about the argument; the advantages of these features are compared against manual argument summarizing in Table 2.

<table>
<thead>
<tr>
<th>Manual Summary Problems</th>
<th>Argument Summary Tool Solutions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uncertainty regarding which contexts a certain node depends on</td>
<td>Displays information on which contexts a certain node depends on, colors depended-on context nodes</td>
</tr>
<tr>
<td>Uncertainty regarding which nodes depend on a certain context</td>
<td>Displays information on which nodes depend on a certain context, colors depending nodes</td>
</tr>
<tr>
<td>Cannot characterize usage of a certain type of node</td>
<td>Shows all nodes of a certain type</td>
</tr>
<tr>
<td>Impact of changing a node far away from the top-level goal is difficult to determine</td>
<td>Shows path from current node to top-level goal</td>
</tr>
</tbody>
</table>

The argument summary tool summarizes the synthetic argument within the summary window. The coloring
mechanism of the tool outlines the goals within a sub-argument, associated with a particular context, producing a relevant and useful visual overview. Viewing the summaries of all of the associated goals gives a textual summary of the argument. The tool was able to show all nodes of a certain type, in order for engineers to get a sense of their use in the argument. Finally, the summary tool was able to show the impact of changing a node in the argument by highlighting the path from the node to the top-level goal. With the manual development of a safety case, none of these summarization techniques are possible; without them, engineers are less able to understand the structure and logical flow of a large argument.

**Related Work**

The most capable commercial tool for manipulating safety cases is the Adelard Safety Case Editor, or ASCE (ref. 1). ASCE provides a number of features integrated into a sophisticated toolset. ASCE supports development of arguments in GSN and a second notation, Claims-Argument-Evidence (CAE). By contrast, ACCESS is experimental. It is designed to support prototyping and evaluation of new safety-case techniques. New features in ACCESS that are not available in other tools include a rapid prototyping mechanism for arguments in GSN, programmable node labeling, node annotation coupled to report generation, support for rigorous inspections, and comprehensive support for patterns.

**Future Work**

Possible future work includes the development of a method for ACCESS to keep track of who has responsibility for which parts of a large, complicated safety case. This information would allow domain experts to be associated with the parts of the safety case that they know most about. The information would also allow identification of domain experts by others in parts of the argument with which they are unfamiliar.

A second possibility for future work is to develop support for the addition of new capabilities to ACCESS using a plug-in capability. ACCESS is freely available, but it certainly will not meet the needs of all who might wish to use it. A plug-in system would be a conceptually simple and straightforward way to allow users to enhance ACCESS to meet their needs.

A more thorough evaluation of ACCESS would include detailed user studies and comprehensive performance tests.

**Conclusion**

In this paper, we have introduced ACCESS, a novel toolset for the creation, maintenance, and validation of safety cases. ACCESS is a framework for experimentation with novel approaches to safety case development and maintenance. ACCESS uses Microsoft Visio to provide graphic editing and display, and Microsoft Word for other forms of documentation that a safety case might require. ACCESS is presently being evaluated and new features will be added as the requirements are determined.

**Acknowledgments**

The authors thank the students in CS6240 in 2010 at the University of Virginia who contributed some of the ideas in ACCESS. This work was supported in part by NSF grant number 0905375.

**References**


**Biography**

P. Steele, Graduate Student, Department of Computer Science, School of Engineering and Applied Science, University of Virginia, 151 Engineer’s Way, P.O. Box 400740, Charlottesville, VA 22904-4740, USA, telephone – (703) 964-7712, facsimile – (434) 982-2214, e-mail – pvs5x@virginia.edu.

Panayiotis Steele is a graduate student in the Computer Science Department at the University of Virginia. His current research interests include software safety and the use of formal methods in software engineering.

K.R. Collins, Undergraduate Student, Department of Computer Science, School of Engineering and Applied Science, University of Virginia, 151 Engineer’s Way, P.O. Box 400740, Charlottesville, VA 22904-4740, USA, telephone – (804) 731-2151, facsimile – (434) 982-2214, e-mail – krc2p@virginia.edu.

Kristine Collins is a fourth year undergraduate student in the Computer Science Department at the University of Virginia. She will be working at the Johns Hopkins University Applied Physics Laboratory beginning in August. Her research interests include software safety in safety-critical systems, specifically space systems.

J.C. Knight, Ph.D., Department of Computer Science, School of Engineering and Applied Science, University of Virginia, 151 Engineer’s Way, P.O. Box 400740, Charlottesville, VA 22904-4740, USA, telephone – (434) 982-2216, facsimile – (434) 982-2214, e-mail – knight@cs.virginia.edu

John Knight is a professor of computer science at the University of Virginia. He holds a B.Sc. (Hons) in Mathematics from the Imperial College of Science and Technology (London) and a Ph.D. in Computer Science from the University of Newcastle upon Tyne. Prior to joining the University of Virginia in 1981, he was with NASA’s Langley Research Center.