SANDIA REPORT
SAND2006-7728
Unlimited Release
Printed December 2006

Risk Assessment Meta Tool LDRD Final Report

Ann M. Bouchard and Gordon C. Osbourn

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation,
a Lockheed Martin Company, for the United States Department of Energy's
National Nuclear Security Administration under Contract DE-AC04-94AL85000.

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

Printed in the United States of America. This report has been reproduced directly from the best available copy.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone:	(865) 576-8401
Facsimile:	(865) 576-5728
E-Mail:	reports@adonis.osti.gov
Online ordering:	http://www.osti.gov/bridge

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd.
Springfield, VA 22161

Telephone:	(800) 553-6847
Facsimile:	(703) 605-6900
E-Mail:	orders@ntis.fedworld.gov
Online order:	http://www.ntis.gov/help/ordermethods.asp?loc=7-4-0#online

SAND2006-7728
Unlimited Release
Printed December 2006

Risk Assessment Meta Tool
LDRD Final Report

Ann M. Bouchard[1] and Gordon C. Osbourn[2]
[1]Lasers, Optics & Remote Sensing and [2]Complex Systems Science
Sandia National Laboratories
P.O. Box 5800 MS 1423
Albuquerque, New Mexico 87185-1423

Abstract

The goal of this project was to develop a risk analysis meta tool--a tool that enables security analysts both to combine and analyze data from multiple other risk assessment tools on demand. Our approach was based on the innovative self-assembling software technology under development by the project team. This technology provides a mechanism for the user to specify his intentions at a very high level (e.g., equations or English-like text), and then the code self-assembles itself, taking care of the implementation details. The first version of the meta tool focused specifically in importing and analyzing data from Joint Conflict and Tactical Simulation (JCATS) force-on-force simulation. We discuss the problem, our approach, technical risk, and accomplishments on this project, and outline next steps to be addressed with follow-on funding.

ACKNOWLEDGMENTS

We thank m anagement and staff of the Secu rity System s and Technology Center 6400 for ongoing interest in and support for this project.

CONTENTS

FIGURES

NOMENCLATURE

DOE	Department of Energy
DOD	Department of Defense
SNL	Sandia National Laboratories
LDRD	Laboratory Directed Research and Development
SAS	self-assembling software
UI	user interface
JCATS	Joint Conflict and Tactical Simulation
FAQs	frequently asked questions
SQL	Structured Query Language (programming language for querying databases)
NextGen	Next Generation Security Simulation
ATLAS	Adversary Time-Line Analysis System

1. PROBLEM STATEMENT

There are many risk and vulnerability assessment tools already in use, in the DOE community and in other communities. Analysts at DOE sites today struggle to reconcile blast-effects models, most-vulnerable-path analyses, and force-on-force simulations and exercises to arrive at a complete picture of the risk posture of their sites. All of these different tools, although requiring redundant data in many cases, have different data formats, and therefore no way for these tools to effectively "talk to each other." As a result, many analysts perform off-line calculations, either by hand or in a spreadsheet, to compute their risk. Such calculations are error-prone, difficult to document and defend in audits, and difficult to compare from one analyst to the next (i.e., one DOE site to the next.) At the enterprise level, DOE has limited ability to consolidate and compare the risks and vulnerabilities at multiple DOE sites, to arrive at a complete picture of the security posture of the nation's nuclear assets. Analyses of non-nuclear assets, such as water supplies, power plants, dams, etc. can suffer similar problems—disparate analyses and file formats, with no way to scale up and pull together the bigger picture.

The technical issues can be summarized as follows: (1) When a site analyst wants to combine the data from two different tools, he wants to do it *today*, not wait several months for a programmer to write a conversion routine. And if next week, he wants to combine another tool, he'll have yet another wait for the programmer. How can the analyst have the power in his own hands to generate a converter *on demand* for *any* new tool he chooses? (2) Once the tools' data are consolidated, the site analyst (or a tool developer) may want to try many different ways of combining these data. The capability is needed, particularly post-9/11, to rapidly generate new analyses that were never considered when the original tool was built. How can the site analyst or tool developer have the power in his own hands to generate new analyses, from equations or an algorithm, on demand? (3) At the enterprise level, issue (2) can be abstracted one level higher. Rather than defining analyses to combine the data from different tools, the enterprise risk analyst or manager wants to combine risk, consequences, etc. of different sites. Again, she may want to try many different ways to combine, compare, or compute results from the data, without waiting for a programmer. How can the enterprise analyst have the power in her own hands to generate new enterprise-level analyses on demand? (4) How can one tool scale, so that it provides for the needs of both the site analyst and the enterprise?

2. PROPOSED SOLUTION

2.1. Risk Assessment Meta Tool

To address these problems, we proposed to develop, not a new analysis tool, but a *meta* tool—a tool that will enable analysts both at the site and enterprise levels to combine and analyze data from multiple other tools on demand.

The key features of the fully developed meta tool are: (1) It will generate conduits to the many different analysis tools and their data files, to bring these disparate sources of data into a single environment. To the extent possible, the meta tool will discover the file format and generate the conduit automatically. Where necessary, the user will have to provide the file format, either via the system's user interface, or via a format specification file. (2) Through the established conduits, it will provide a means of exporting data imported from one tool to the appropriate format for another tool. This should help reduce the amount of redundant data to be entered. (3) A user interface will allow an analyst to "wire up" a calculation from the disparate data sources imported, for example, combining probability of attack from one tool, target consequence from a second, probability of interruption from a third, etc., into an overall risk (or other quantity) calculation. Alternatively, the analyst could create new analyses using the data from a single tool. For example, he might create a cost-benefit analysis on top of his favorite existing tool. Different calculations can be created and saved, and run again with different input files. So, in a sense, this meta tool could provide the capabilities of many individual new analysis tools. (4) Calculations and their results can be shared, so that an enterprise-level user can combine and compare data or calculations from facilities across the enterprise.

2.2. Technical Approach

The technical approach is based on Sandia's recently patented [1] self-organizing software (also called self-assembling software) technology. Self-assembling software (SAS) was modeled after the dynamic self-assembly processes of proteins within living cells, which enable proteins to carry out much of the construction, information processing, reconfiguration, and repair of living systems.

We have identified a few crucial properties of proteins and their interactions that are sufficient to enable dynamic self-assembly. (1) Proteins have tremendously selective binding sites, operating much like a lock and key. (2) Binding or unbinding a complementary protein at one of these sites can result in a conformation change (change in shape) of another part of the protein. This conformation change can perform some sort of actuation, such as moving (e.g., in motor proteins) or catalyzing an assembly or disassembly reaction (e.g., in enzymes). (3) A conformational change can also expose (or hide) additional binding sites, which in turn can bind and cause a conformational change resulting in actuation, or exposing or hiding yet another binding site.

We abstract these important self-assembly properties of proteins into an "agent," the fundamental building block of our self-assembling software. An agent can store data, perform

some simple or complex computation, or both. Each agent has one or more binding sites (each labeled with a numeric key) that can bind only to complementary sites (property (1)). Sites are said to be complementary when they have keys with the same absolute value but opposite sign. Once bound, property (2) enables the agent to actuate (perform its computation). Property (3) enables it to then bind to another agent, to trigger it to execute next.

A specific execution sequence or biological signaling pathway can be "wired" together by including a set of agents with binding sites that drive them to execute sequentially. For example, suppose each agent has a "trigger" site that activates it (causes it to execute some code) and a "done" site that is exposed when its task is complete. Suppose agent A's done site is complementary to agent B's trigger site, agent B's done site is complementary to agent C's trigger site, and agent C's done site is complementary with agent D's trigger site. Once A is triggered, then B will execute, followed by C, followed by D.

It is important to note that such an execution sequence or pathway is not hard-coded, but *self-assembled*. That is, the agents are just "dumped" into the simulation environment, and the execution order occurs as a natural consequence of the order in which binding and unbinding events occur. As a result, the self-assembling executable code is inherently *dynamic* in nature. The structure of the executing code is assembling and disassembling all the while it is executing, with execution pathways that are driven dynamically by matching keys between agents. All that is required to change the execution pathway—"re-wire" what the code does, or turn code on or off—is a change of keys. Additional details on the self-assembling software approach have been published elsewhere. [2,3]

SAS technology provides a mechanism for the user to specify his intentions at a very high level (e.g., equations or English-like text). Then the agents required to execute these commands are automatically generated, self-assemble, and execute in proper sequence, taking care of all the implementation details. SAS enables the meta tool to self-assemble the code to import new file formats or perform new calculations as needed.

2.3. First Version Requirements

Since the meta tool was intended to address the needs of security analysts, we interviewed a number of security analysts at Sandia and other DOE sites regarding their needs and priorities. General consensus held that the greatest need was to create *ad hoc* analyses from the output of a widely used force-on-force simulation tool, Joint Conflict and Tactical Simulation (JCATS) System. JCATS, designed by Lawrence Livermore National Laboratories, generates several different types of output files. One type contains raw data—every event that occurs in the simulation—but is not formatted in a way to facilitate human interpretation. Others are compiled results, organized in a clear human-readable report, but pre-programmed to include only some of the events of interest to the analysts. Thus, analysts currently tend to scour the unfriendly all-event data manually, looking for individual events, then adding them to the compiled reports and re-computing statistics manually. This is a very tedious, time-consuming, and potentially error-prone process. At a minimum, having some way to automate this process, making it robust and reliable, would be an advantage over current capabilities.

In addition, the JCATS lab at Sandia National Laboratories in Albuquerque runs JCATS simulations for many outside customers, including DOE, DOD, and other government agencies. Each customer may have different objectives, so it is common for customers to ask the JCATS lab staff to perform some type of analysis on the JCATS output that they have never done before. This frequently involves additional manual sifting and compiling of data. Thus, the ability to create new analyses on-the-fly would be a tremendous benefit, especially to the Sandia JCATS lab staff, but also to other JCATS users as well.

We therefore determined that the first version of the meta tool would focus on JCATS analyses specifically. We decided to first support importing the raw data files. Since these files contain all events that occurred in the simulation, this provides maximum flexibility on what the user can analyze. While designing and implementing the SAS generator (capability to automatically generate agents that self-assemble and execute a specified task), it was useful to have a set of realistic tests. A small set of analyses that JCATS users commonly perform manually served as this test set. Once created as tests, we decided to provide these analyses already pre-built to users for their convenience. Finally, we would develop a user interface to enable users to create new analyses of JCATS (or other tabular) data, in English-like sentences and equations.

Although the first version was specifically geared toward JCATS analyses, the infrastructure and features support a much greater variety of tasks than that. It would be ready to handle any other type of tabular data, once provided with a format specification. The expected follow-on data sets, after JCATS, are Next Generation Security Simulation and force-on-force exercise data. Both of those types of data sets are text files in a tabular form. Version 1 of the meta tool should be fully capable of analyzing all three of the data sets together as needed.

2.4. Technical Risk

In keeping with the high-risk/high-payoff spirit of the Laboratory Directed Research and Development (LDRD) Program, this project was very high risk. In fact, we acknowledge that we did not appreciate at the outset of the project just *how* high-risk developing the new SAS paradigm would be, particularly in its impact on the project schedule.

Programming for established platforms—take Windows for example—has a huge advantage in the decades of infrastructure, pre-build widgets (buttons, checkboxes, etc.), patterns, templates, libraries, newsgroups, and FAQs that are widely available. Such infrastructure and networks of other knowledgeable programmers can greatly enhance a programmer's productivity, in that so much code can be re-used, plugged in, or adapted, and one is likely to find someone else who has solved a problem quite similar to one's own.

In contrast, in this project, we have been developing a new paradigm from the ground up. We did not have pre-built buttons and check-boxes, because SAS versions of them did not yet exist. There were no templates or libraries until we built them. The amount of design and implementation involved in this project has been enormous and unusually complex and innovative. These designs have been *nothing* like any code we've ever written or seen before (and we are seasoned programmers with decades of experience). Problems arising with the SAS approach are inherently different from other approaches such as structured or object-oriented

programming. And there is no network of other programmers with experience in SAS we can turn to for lessons learned. So we are discovering and learning as we go, much as had to be done in the early days of establishing those other paradigms. The bottom line impact on the project was that we almost, but not quite, completed the first version of the meta tool within the three-year life cycle.

In spite of the delay, however, our accomplishments to date very strongly support the validity of our approach. As will be discussed in greater detail in the following section, we are automatically importing JCATS files, automatically generating appropriate agents that self-assemble and execute analyses on JCATS data, and so forth. In other words, the SAS approach is working. It is also worth noting that this project's investment in the SAS infrastructure (tens of thousands of lines of source code) will have far-reaching impact, well beyond this project.

3. ACCOMPLISHMENTS

3.1. Intellectual Property

Throughout the life of this project, we have generated a significant amount of intellectual property. We have submitted fifteen technical advances (referred to in some businesses as invention disclosures) on various aspects of the SAS infrastructure, generator, and interface, developed as part of this project. An LDRD final report should have unlimited distribution, and we wished to avoid setting a bar date for filing patent applications. As a result, some of our accomplishments are not discussed, or not discussed in detail, in order to avoid disclosing here any intellectual property that has not been protected.

3.2. First Version Capabilities

As mentioned earlier, the first version of the meta tool was focused on JCATS analyses. The minimum capabilities would therefore be to allow the user to: (1) import JCATS data, (2) view it, (3) organize multiple JCATS files, (4) run pre-built analyses on the data, (5) create new analyses on-the-fly, and (6) view the results. Additional capabilities, to make it a complete solution, would be to export and/or print the results, but our initial efforts were focused on the core six capabilities enumerated above. All of these core capabilities require a user interface component. Items (4) and (5) also involve generating self-assembling agents to execute the specified analyses under-the-hood.

A number of design principles drove the design of the user interface. We sought a unifying metaphor that would make viewing data, organizing data, viewing results, running pre-built analyses, and creating new analyses similar enough to each other that users could intuitively grasp how to do one given an understanding of the other. It had to be easily extensible to other types of data than tables (e.g., databases). It had to be modular so that self-assembling agents could be generated automatically to implement new objects that users create on the interface. Since we were developing a new paradigm, we felt no hesitation in breaking out of the standard Windows interface mold. Indeed, exploring totally new, easier ways of interacting with software was a side benefit of using the new SAS paradigm.

3.2.1. The Book/Bookcase Metaphor

The metaphor we established was one in which data is stored in books on bookcases. When a new JCATS file is imported, it goes in a book on a bookcase. Multiple tables (for example, multiple simulation runs of the same scenario) can be grouped in a single book. When an analysis is launched, the inputs are entered into a book. When results are generated, they go in a book. In this way, inputs and outputs can be organized together into a single book. The color and location of the book on the bookshelf are the unique identifiers for the book.

Figure 1 is a screen capture of the meta tool user interface (UI), i.e., working meta tool software, showing ten bookcases. The left-most bookcase has two books on the top shelf. Clicking on the narrow edge of one of the stacked bookcases exposes its full width (see Figure 2).

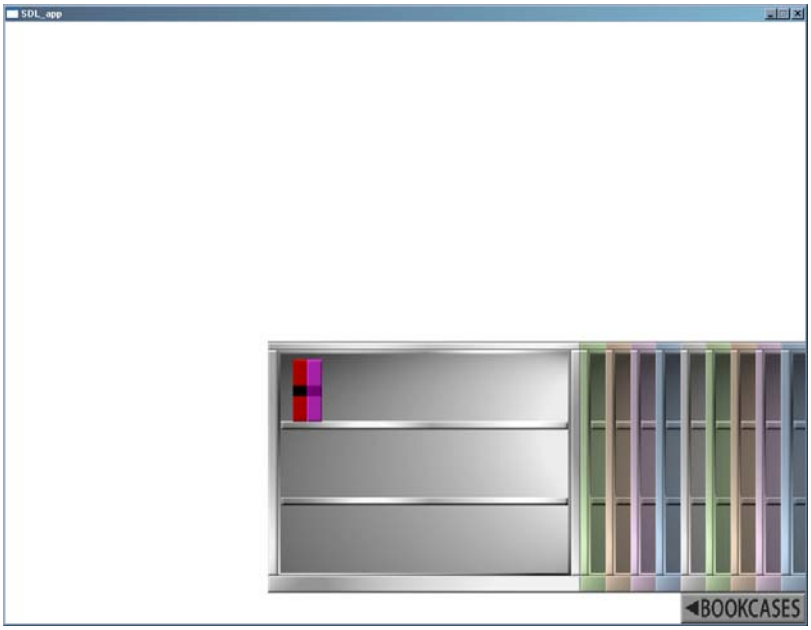

Figure 1. Meta tool screen capture showing ten bookcases, with two books on the left-most bookcase.

Figure 2. Clicking on the blue bookcase exposes its full width.

Clicking the spine of a book opens the book (see Figure 3). There are two ways to turn pages, which are analogous to the real-world flipping a big chunk of pages and turning a single page. Clicking somewhere within the page edges jumps to that page. E.g., starting from page 1, clicking 80% of the way into the right pages edges of a 100-page book turns to page 80 (see Figure 4). Alternatively, clicking the page margin turns a single page. The open book can be moved by dragging it from the top of the binding. Clicking the left or right binding re-shelves it.

Clicking the “BOOKCASES” button in the lower right corner of the UI alternately hides and exposes the bookcases. Note that the bookcases are hidden in Figure 4.

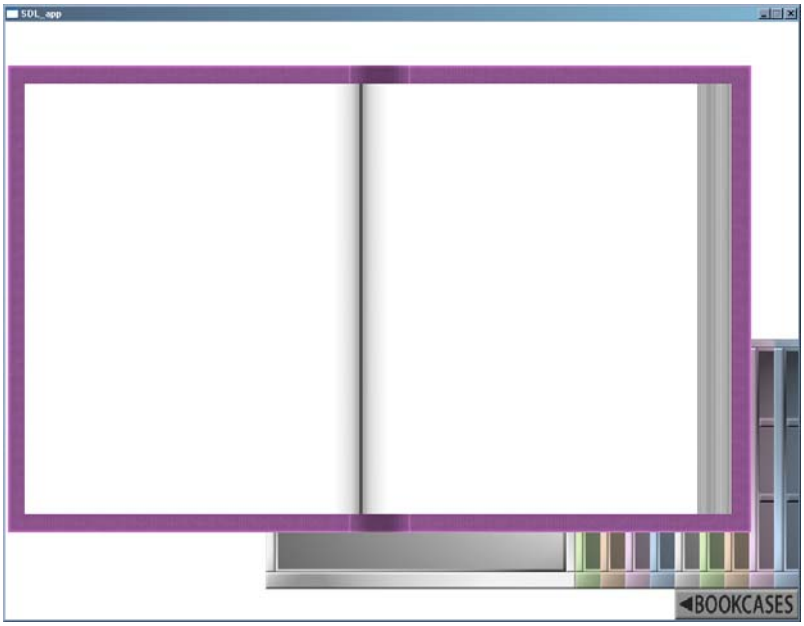

Figure 3. Clicking on the magenta book spine of Figure 1 opens the book. In this screen capture, the book is opened to the first two pages, which are blank.

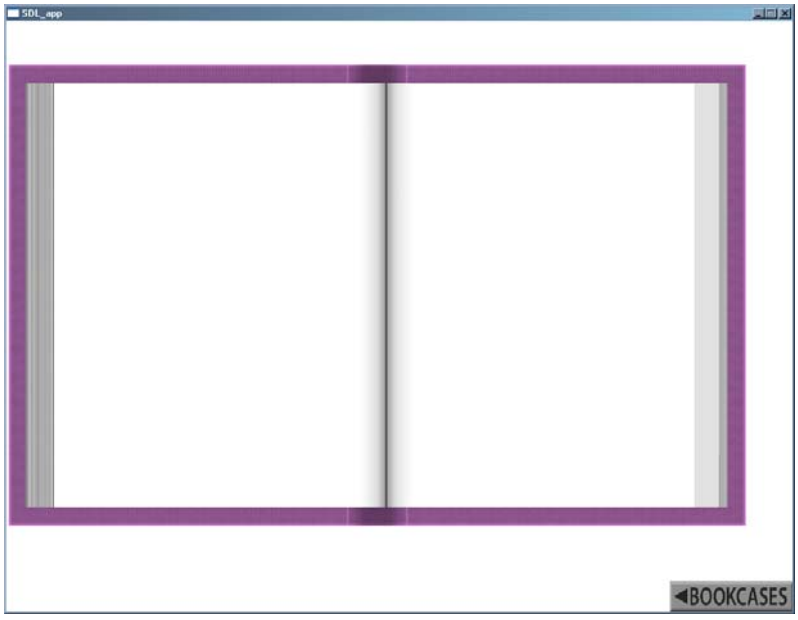

Figure 4. Clicking somewhere in the the page edges turns to the selected page. Here we have turned to page 80 of the 100-page book. Clicking on the page margin (highlighted in light gray on the right page) turns a single page.

3.2.2. Making New Books

Each object that appears in the meta tool (e.g., bookcases, books, etc.) has a UI component and code underlying it. The underlying code to make new books (allocate space in memory, set up the appropriate data structures, etc.) and the UI component (making it visible on the screen) are completed, as is evident from the figures.

A related feature is a mechanism for the user to invoke the make-book command. We have designed a context-sensitive menu mechanism for this function (as well as other functions). A context-sensitive menu is a menu that appears when the user clicks in a particular location, where the menu choices (things the user can do) are specific to the location. For example, all books go in bookcases, so if the user clicks in empty space within a bookcase, a menu will be displayed in which one of the choices is "Make Book". If the user clicks the "Make Book" command, a new book will be made and inserted in the bookcase where the user clicked. If the user clicks elsewhere, where books cannot be made, "Make Book" is not a choice on the menu. Although the make-book function is implemented, the context-sensitive menu is not complete as of the time of this report.

3.2.3. Importing Tabular Data

The raw JCATS data files (those that contain all events that occurred during the simulation run) have a published format. In order to be able to import these JCATS raw files, we wrote by hand a format specification file. We have implemented the meta tool code to use the format specification file to import a file of a specified name, and insert it into a book. Figure 5 shows a book containing an imported JCATS file. Note that this JCATS file was created from a "toy" example, and is not related to any real secure facility or real security forces.

This code is general, in that any other types of text files containing tabular data can be imported using the same code. We only need to write a format specification for the new file format. After the first version of the meta tool is completed, we plan to write additional format specification files for Next Generation Security Simulation files, and files containing data captured from live force-on-force exercises.

As with the "Make Book" command, the "Import" command is invoked via a context-sensitive menu (not yet implemented). Since tables can only be imported into a book, when the user clicks on a blank page of a book, a menu will be displayed that contains the "Import" command.

3.2.4. Viewing Tables

The full contents of large tables (such as those generated from JCATS runs) cannot be displayed all at one time. We designed tables to be broken up into "views" eight columns wide (four columns each on the left and right pages). To see additional *columns*, the user must click on tabs at the sides of the table. The tabs work analogously to tabs in a binder. They act almost as a mini-book within the current visible pages. The same rows are always displayed, but the tabs

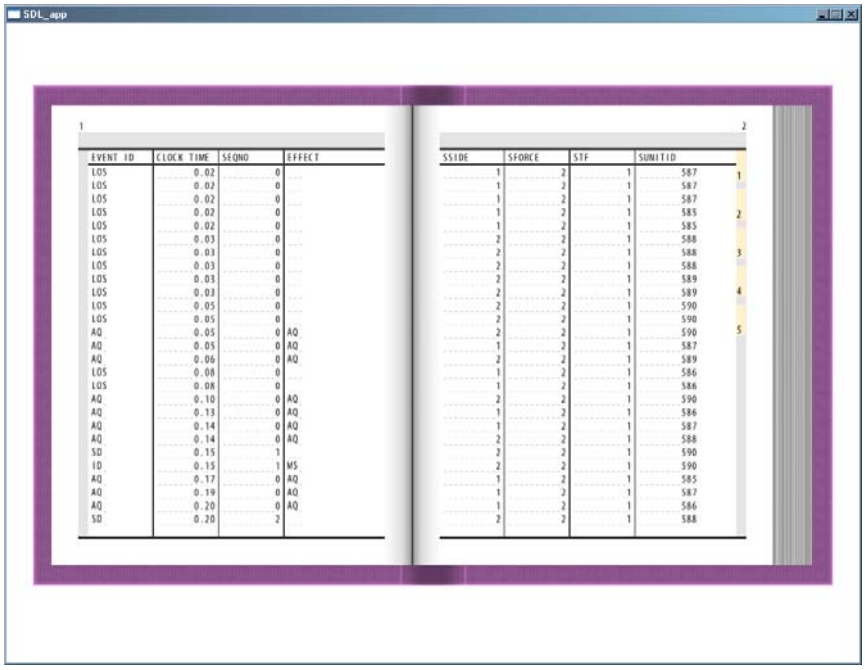

Figure 5. A JCATS file imported into a book.

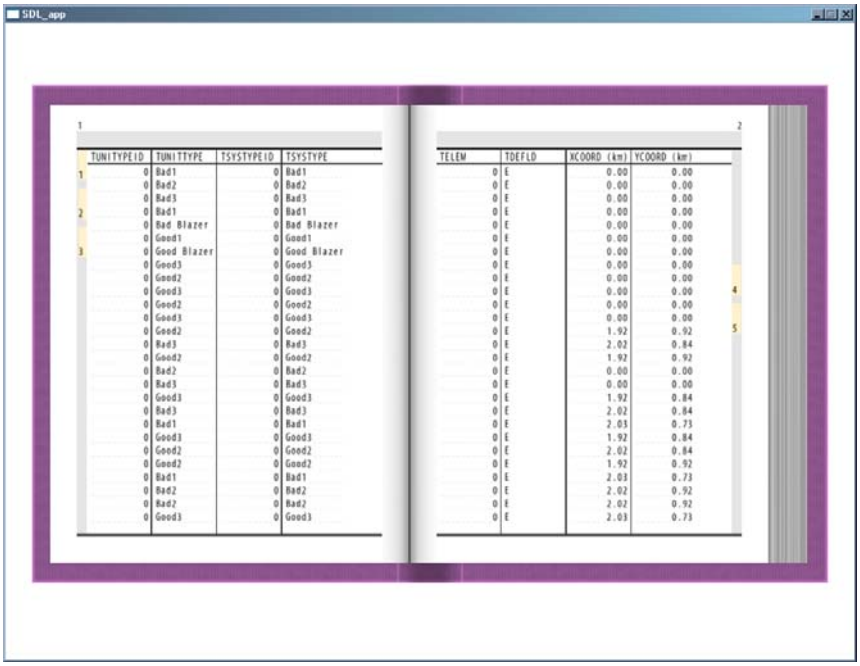

Figure 6. Clicking on tab 3 flips tabs 1-3 to the left, analogous to flipping tabs in a binder. The rows shown in this figure are the same as those shown in Figure 5, just different columns.

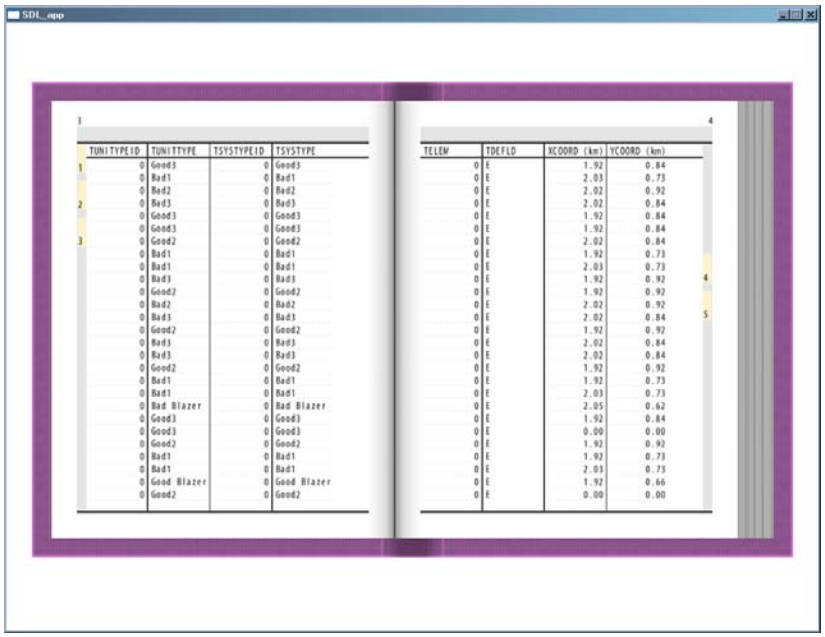

Figure 7. Turning the page of the book shown in Figure 6 shows the next set of rows, while maintaining the same view of the table (same columns).

allow the user to flip back and forth between different sets of columns. Figure 6 shows a different view of the JCATS table that is shown in Figure 5. To see additional *rows*, the user turns the pages of the book. As each page is turned, the same view of the table (same columns) is displayed (see Figure 7).

3.2.5. Analyzing Tabular Data

As with other objects in the meta tool environment, analyses have a UI component and underlying code.

3.2.5.1. UI

The analysis UI was designed to specifically provide the types of analyses that the JCATS specialists described to the project team. JCATS specialists perform these analyses frequently, for most JCATS scenarios they run. Due to the sensitive nature of these analyses, they cannot be described here. However, in general, the goal is to select, sort, and summarize the events of a force-on-force simulation (scenario) in a meaningful way, such as who was killed and when, who shot them, what weapon was used, etc. It is important to note that, due to the limitations of JCATS' own standard reports, the JCATS specialists currently transfer the JCATS analysis results to a spreadsheet program, such as Excel, hand-enter additional data, and perform the desired counts of events and other summaries manually. This process is tedious, time-consuming, and potentially error-prone.

The Meta Tool imports JCATS raw data, in addition to the standard reports, thereby providing the user with a more complete dataset to begin with, and eliminating the need to hand-enter additional data. The key analysis capabilities provided by the Meta Tool are mathematical

functions, a “find” command, and a “watch” command. All commands are constructed in English-like sentences. For example, “In [this JCATS file], find all the events between time 0 and 5 minutes, where the Shooter is ‘GoodGuy1’.” Then, “In [those results], find all the events where the effect was a kill.” Then, “Count [those results].” These commands can be combined into a task, with the JCATS file as an input. In addition, a “watch” can be set up so that, whenever a new JCATS file is imported, this task is automatically launched. All the user needs to do is import the file, then look at the results.

The code to implement this UI has not been completed. The key accomplishment here was in designing how to provide the key searching and summarizing capabilities needed by the JCATS analysts, without requiring them to know a complex programming language (such as SQL) to specify their search. The only language required is English. Programming without a programming language (e.g., C, C++, Java, etc.) is a long-standing problem in computer science, and our progress on this problem can have consequences reaching far beyond this project.

3.2.5.2. Underlying Code

In addition to devising a user interface with which non-programmers can program new analyses, we also had to design a means of translating the UI commands into self-assembling agents to carry out the specified task. We began by designing a set of universal building blocks, agent types that could be used over and over, that could be rearranged or reconfigured to carry out different tasks, and yet successfully and reliable self-assemble and execute in the proper sequence no matter what task the user designs. These building blocks served the functions of data storage, looping, logic branching, etc.

The next challenge was to design a way to generate from the user’s UI commands the appropriate agent types with appropriate binding-site keys. Since the UI has not been implemented, we instead designed a data structure that would capture the UI commands when it is implemented, and then hand-wrote some of those data structures to use as test cases. Specifically, we used real JCATS analyses performed by JCATS analysts.

Both the building blocks and the auto-generator have been implemented. Given the data structures that represent the test JCATS analyses, the auto-generator does generate the appropriate building blocks, and they do self-assemble, execute the analysis, and provide correct results. We have performed further tests where the analysis contained multiple hierarchies of sequences of commands (i.e., a program that calls two sub-programs, each of which calls sub-sub-programs), and the auto-generator generated the building blocks to self-assemble and execute the complex commands correctly.

Figure 8 shows the results of one analysis, run on the table shown in Figures 5-7. The English equivalent is, “In [this JCATS table], find all events where the effect is a kill, and the shooter side is 1 (good guys).” These results were created by the auto-generator generating all the appropriate agents to carryout this analysis. They then self-assembled and executed, giving a result table, which was then inserted into a book. KK in the effect column means kill, and 1 in the SSIDE column is the good guys. There were only 4 events that met these criteria, so there are four rows in the results table of Figure 8.

Whenever results are created, they are automatically saved as well. From the user's perspective, they are in the book. He does not have to assign or remember a filename or folder name on the hard drive. When he opens the book again, the results are there. The same is true for the original JCATS table that was imported. Everything is in the book in which it was originally placed.

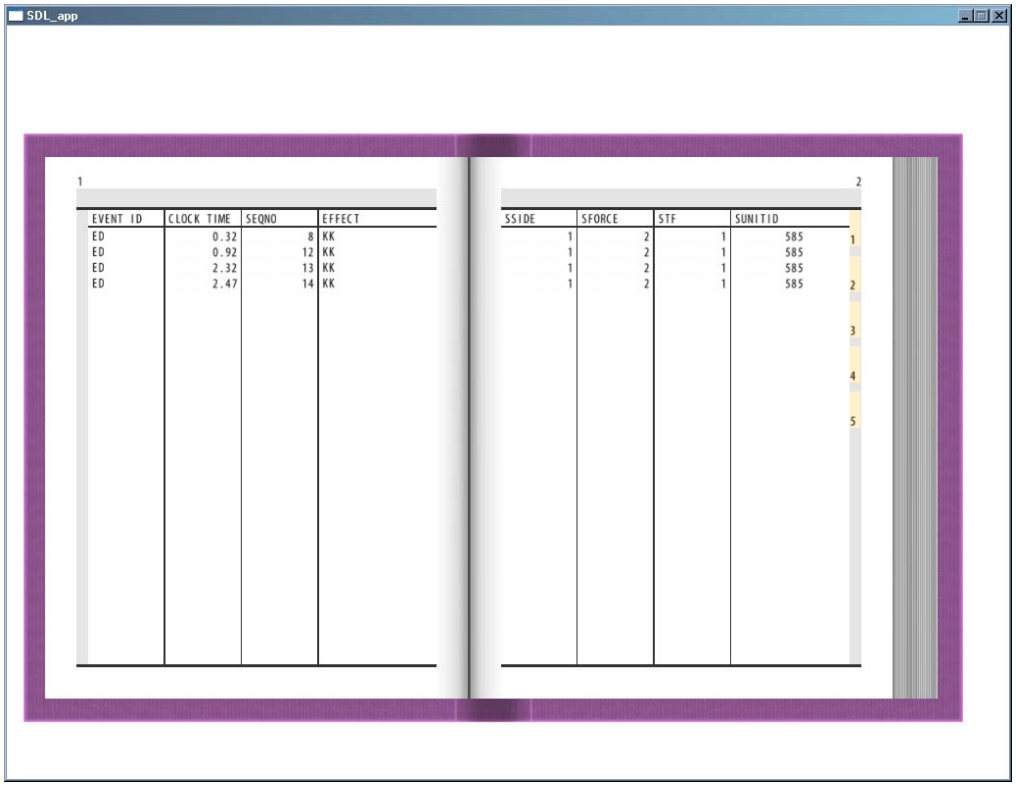

Figure 8. Results of a JCATS analysis.

4. NEXT STEPS

The project team is continuing to develop the first version of the meta tool using follow-on funding. The next steps are to complete the implementation of the context-sensitive menus, so that the user can invoke the Make Book and Import commands directly from the bookcase and book, respectively. We then must implement the UI for specifying analyses, the English-link programming interface. Both of these tasks, the context-sensitive menus and the analysis interface, involve rendering objects to the computer screen and writing event handlers such that, when the objects are clicked, it executes the make book, import, or auto-generate/execute code that is already written.

We have received Human Studies Board approval to conduct a pilot usability study after completion of version 1. Essentially, we will invite users to try out the software, and attempt to do different tasks, such as open books, turn pages, run analyses, etc. The purpose of this testing is to determine how much of the system is intuitive and easy to use, and where users run into problems. This will suggest areas for improvement.

We will also create format specifications for the Next Generation Security Simulation (NextGen) files and force-on-force exercise files. This will enable the development of analyses that compare the three different types of force-on-force evaluation systems—JCATS, NextGen, and exercises.

If time and follow-on funding allow, we will then extend the tool to incorporate additional security analysis tools. DOE analysts have indicated that the next tool it would be most valuable to incorporate would be the Adversary Time-Line Analysis System (ATLAS), developed by Sandia National Laboratories.

5. CONCLUSIONS

There are many risk and vulnerability assessment tools already in use, and each has its purpose and strength. However, since these tools do not talk to each other, analysts are faced with tedious and potentially error-prone manual processes to combine, merge, or compare input and output from these various tools. Additionally, analysts are handicapped in situations in which they must quickly respond to a newly recognized threat, because their software tools were not designed to handle such previously unrecognized threats, and it may take months or years for the software developers to add the necessary enhancements to the tools.

Our proposed solution to this problem was not another risk assessment tool, but a *meta* tool—a tool that enables analysts to combine and analyze data from multiple other tools on demand. Our approach was based on our innovative self-assembling software technology. Conduits to import files from these other tools could be self-assembled, so that their data exists within a single environment. Then, the user can create analyses himself, using equations or English-like text, and the code to execute these tasks would be self-assembled under-the-hood, taking care of all the minutiae. In this way the analysts has the power in his own hands to program new analyses on-the-fly, whether it be in response to changing threats, or simply to look at the data in new ways.

Because our approach was based on a new software paradigm—self-assembling software—we invested substantial resources into developing the infrastructure on which the meta tool itself was based. The impact of embracing a high-risk/high-payoff approach (suitable for an LDRD project) was a delay in the completion of version 1 of the meta tool. However, accomplishments to date indicate that the approach is technically sound. We have implemented a novel user interface based on a books/bookcases metaphor, in which all data is stored in books on bookcases, rather than a hierarchical tree of folders (with names that can be difficult to remember). We have implemented code to import a JCATS file and insert it into a book. We have completed the code to translate a data structure reflecting the English-like programming commands of the user into self-assembling software agents that self-assemble and execute the program. Thus, much of the capabilities slated for version 1 are complete. What remains, and will continue to be developed with follow-on funding, are context-sensitive menus to invoke the commands to make new books and import new JCATS files as well as the user interface for the English like programming.

It should also be emphasized that the investment in developing the self-assembling software infrastructure (tens of thousands of lines of source code) as well as the book/bookcase interface and auto-generation capability (tens of thousands more) will have impact well beyond the meta tool itself. We believe this project enabled us to move from a simple proof-of-concept of the self-assembling software to a highly intuitive user experience in which the non-programmer can program his/her own software as needed. This can truly be the basis for a software revolution.

6. REFERENCES

1. A. M. Bouchard and G. C. Osbourn, Method for Self-Organizing Software, US Patent 6,957,415 B1, Oct. 18, 2005.
2. G. C. Osbourn and A. M. Bouchard, Dynamic Self-Assembly of Hierarchical Software Structures/Systems. AAAI Spring Symposium "Computational Synthesis: From Basic Building Blocks to High Level Functionality," held at Stanford University, March 24-27, 2003. AAAI Press, 2003. [Published in Proceedings]
3. A. M. Bouchard and G. C. Osbourn, Dynamic Self-Assembly and Computation: From Biological to Information Systems, First International Workshop on Biologically Inspired Approaches to Advanced Information Technology, held in Lausanne, Switzerland, January 29-30, 2004. Springer, 2004. [Published in Proceedings]

DISTRIBUTION

1	MS0780	Stephen Ortiz	6424
2	MS9018	Central Technical Files	8944
2	MS0899	Technical Library	4536
1	MS0123	D. Chavez, LDRD Office	1011
1	MS0161	Patent and Licensing Office	11500


Sandia National Laboratories