

# Ontology Development and Evolution in the Accident Investigation Domain

Robert Carvalho  
NASA Ames Research Center  
MS 269-1  
Moffett Field, CA 94035  
650-604-3593  
Robert.E.Carvalho@nasa.gov

Dan Berrios  
University of California, Santa Cruz  
MS 269-2  
Moffett Field, CA 94035  
650-604-0470  
berrios@email.arc.nasa.gov

Shawn Wolfe  
NASA Ames Research Center  
MS 269-1  
Moffett Field, CA 94035  
650-604-4760  
Shawn.R.Wolfe@nasa.gov

James Williams  
NASA Ames Research Center  
MS 213-4  
Moffett Field, CA 94035  
650-604-6377  
James.F.Williams@nasa.gov

*Abstract*—InvestigationOrganizer (IO) is a collaborative semantic web application designed to support mishap investigations, and has been used for accidents ranging from those involving only minor property damage to the loss of the Space Shuttle Columbia. The development and use of IO in support of these investigations has provided significant lessons about the use of semantic web technologies in real-world systems. IO is a data and knowledge repository for a wide range of mishap related information in which investigators meaningfully structure information and link together evidence, causal models, and investigation results. The types of knowledge that investigators can include in the repository are defined by its **investigation ontology**, a component of the system that expresses investigation concepts using a logical formalism. IO developers can dynamically alter this ontology without having to recompile the application. This paper describes the development of the investigation ontology for IO, focusing on its growth in response to user needs during the investigations, as well as efforts to control that growth.<sup>12</sup>

## TABLE OF CONTENTS

1. INTRODUCTION.....	1
2. ONTOLOGY DEVELOPMENT PROCESS.....	2
3. INITIAL FINDINGS.....	3
4. THE IO ONTOLOGY MODELING SYSTEM.....	5
5. LIMITATIONS OF THE IO ONTOLOGY SYSTEM .....	5
6. LESSONS LEARNED.....	6
7. RECOMMENDATIONS FOR FUTURE WORK.....	7
8. REFERENCES .....	7

9. ACKNOWLEDGEMENTS.....	7
10. BIOGRAPHIES.....	8

## 1. INTRODUCTION

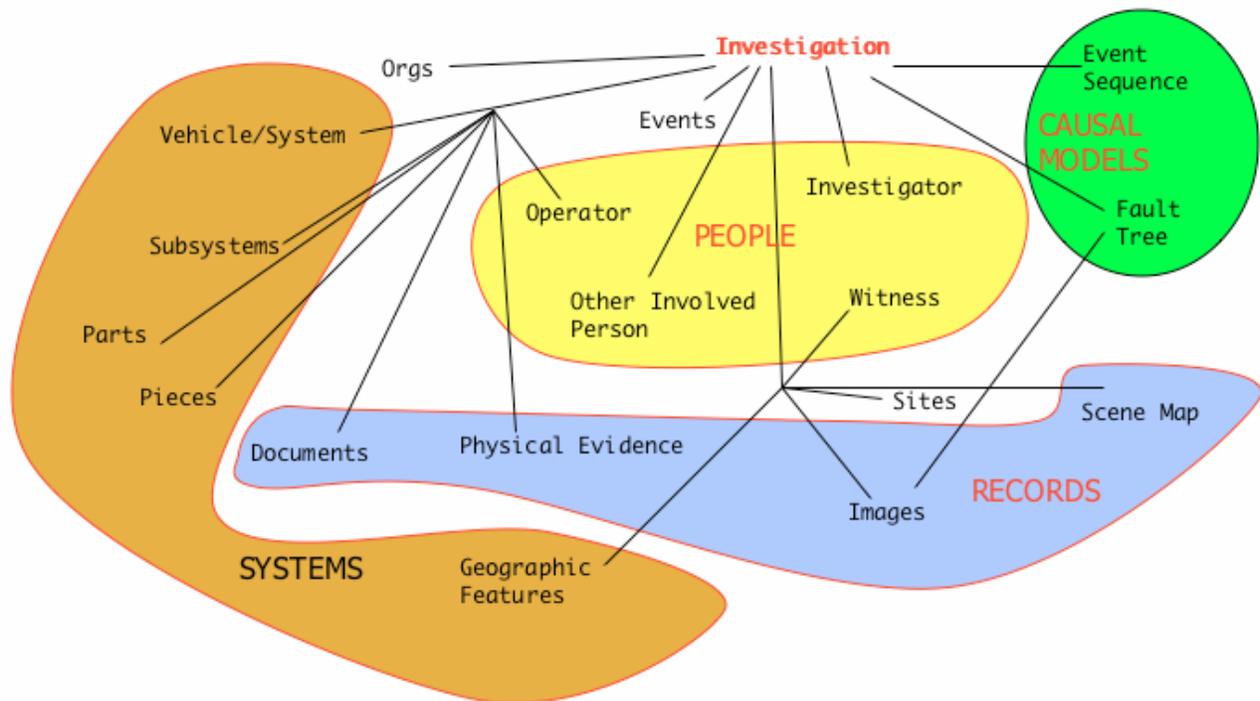
### *InvestigationOrganizer*

InvestigationOrganizer (IO) was developed starting in 2002 as a new application of the Semantic Organizer system [1]. IO is a data and knowledge repository for a wide range of mishap related information in which investigators meaningfully link together evidence, causal models, and investigation results. It has been subsequently used in support of investigations into a wide range of aerospace accidents, helping to manage and distribute evidence, track the progress of elements of the investigation, and support the development of investigation results. The investigations using IO range from an air show incident resulting in minor property damage to the loss of the Space Shuttle Columbia.

The types of knowledge that investigators can include in the repository are defined by its **investigation ontology**, a logical formalism of the concepts in a particular domain [2]. The ontology represents the heart of how the IO system functions, defining forms for knowledge entry, browsing, searching, and display. While there were other tools that could have been used to develop this ontology (e.g., Protégé [3]), none of these tools at the time supported the collaborative and dynamic alteration of ontologies over the web. Instead, the IO modeling mechanisms were used to build this ontology. This paper relates the experience developing and refining this ontology based on knowledge and experience from actual investigations that used IO.

<sup>1</sup> U.S. Government work not protected by U.S. copyright.

<sup>2</sup> IEEEAC paper #1106, Version 5, Updated December 23, 2004



**Figure 1 - The initial IO ontology conceptualization. Each of the lines connecting classes represents an important semantic relationship (labels of these relationships were omitted for clarity). Not included in the diagram, the concept includes many more relationships between the classes, and many of the classes have specialized subclasses**

## 2. ONTOLOGY DEVELOPMENT PROCESS

### *Initial IO Ontology Development*

An initial version of the ontology for IO was created based on *NASA Procedural Requirements for Mishap Reporting, Investigating, and Recordkeeping* (NASA Procedure Guideline (NPG) 8621 [4]). This conceptualization began as a series of sketches in PowerPoint (e.g. Figure 1) then was translated into a more structured ontology description before being formally represented in the IO system. The original specification consisted of 29 classes and 228 properties (142 relation types and 86 literal property types).

Once this was done, two engineers with extensive mishap investigation experience reviewed the ontology, making mostly minor modifications. The ontology was then tested by entering archived data from a minor mishap that had occurred at NASA Ames Research Center several years before, involving the failure of a Canard Rotor Wing (CRW) model. This test included the creation of over 200 instances of various classes in the IO system.

### *Rapid Development*

Before development and testing was completed, IO was brought in to support an actual investigation when the Comet Nucleus Tour (CONTOUR) spacecraft was lost. Moving the tool into a real-time critical application caused a substantial shift in the way development of the system and

the ontology were handled.

Six months after the CONTOUR mishap, the loss of the Space Shuttle Columbia once again thrust the tool into an investigation support role. The demands on the tool were examined and subsequent changes to the process were made to support these mishap investigations.

Testing continued throughout these investigations, which had thrust the IO development team into an environment that was both high-pressure and high-visibility. During each of these investigations, system developers did their best to accommodate ontology change requests as quickly as possible. The rapid changes frequently bypassed any kind of review of the nature or impact of the changes to the overall system. While each request made sense to the investigator at the moment, it was difficult to address the long-term impacts such changes would create. For a more detailed review of how IO impacted these investigations, see the companion paper [5].

By the time the Shuttle Columbia investigation concluded, the ontology had grown to 64 classes, with more than 560 properties (including 470 relation types and 93 literal property types).

### 3. INITIAL FINDINGS

#### *User Studies*

Due to the close working relationships during these investigations, users were able to provide near real-time feedback on usage, features, and issues. The team worked to incorporate these changes as soon as possible, and to use them to support subsequent investigations.

During the initial start-up phase for the Columbia investigation, two of the primary users from the CONTOUR investigation provided important feedback based on their experience with IO. After using the ontology to encode their information, they indicated that many of the relation types in the ontology were never used.

During the Columbia investigation, several users complained that too many choices existed to classify items and relationships. Subsequently, a NASA Human-Computer Interaction (HCI) team performed a study of IO usage by the Columbia Accident Investigation Board (CAIB). Their study found that "subtle distinctions between each of the different types ... complicated the process of choosing the correct type" or class and led to different users using different classes and relations to model similar information. For example, the study found that most users were frequently choosing the more general class "Document" and none of its eight subtypes. Users similarly avoided highly specific relation types, because they had difficulty quickly choosing the correct one.

Despite its early shortcomings, there was a high level of

interest in the tool, and NASA began to develop a "commercial-grade" version of IO with Xerox Corporation. An ontology review team was formed to make recommendations that would reduce the ontology to a more manageable size. First, a report was produced as to how often each class, literal property, and relation was used during five investigations (the CRW test, a minor air show mishap investigation, CONTOUR, Columbia, and HELIOS) for which the system had been used. This report was reviewed to determine which ontology components were actually used, and which might not be needed.

In order, the most frequently used classes were "Document," "Photo," "Background Fact," and "Finding" (Table 1). "Visual Record" was also more heavily used than was expected given the extent of its subclasses. Because of the unexpectedly high usage of "Document" and "Visual Record" classes, the ontology review team looked at several random samples of instances of these classes to determine whether users were overlooking existing subclasses or whether key subclasses were missing from the ontology.

In the "Document" class the ontology review team found that some instances consisted of aggregated information (several types of documents in one), some were investigation logistics documents, and others were misclassified or could have fit within one of the existing subclasses of Document. They also found that many of the "Visual Record" instances were, in fact, weather data such as satellite and radar images. Based on this investigation, the sub-team created a few important new subclasses out of the existing, highly-used classes.

**Table 1. Usage (number of instances) of IO ontology classes during five air and space accident investigations. Only the 40 most-used classes are shown.**

Class	Usage	Class	Usage
Document	2990	Participant	86
Photo	1196	Workgroup	71
Background Fact	720	Chart	66
Finding	632	Physical Evidence	61
Analysis Report Record	404	Organization	52
Fault Tree Node	399	Project Control Record	49
Person	336	Video Recording	49
Review Document	323	Drawing	42
Visual Record	311	Interview	37
Recommendation	273	Scene Map	37
Event	220	Observation	36
Investigator	205	Item Folder	33
Action Item	181	Project Control Procedure	33
Investigation Report Section	177	Training Record	25
Request For Information	164	Photo Folder	22
Record Folder	142	Item	21
Orbiter System	140	Condition	20
Potential Factor	128	Record	20
System	124	Notes	18
Document Folder	108	URL	18



**Figure 2 - The pruning of the class tree. At left is the class tree at the end of the Columbia investigation. At right is the revised class tree showing greater depth.**

*Pruning Efforts*

The ontology review team set a threshold of use then removed or merged 18 classes and more than 200 relations. The labels of several relations were modified so that their meaning was more specific and clear. For example, some hastily added relations had been labeled as "related to", "pertinent to", etc. These labels were deemed so general that users often were at a loss as to how to use them or interpret their use. The ontology review team also merged several classes, and regrouped others in more logical ways. This led to the creation of a deeper class hierarchy with many fewer upper-level classes.

*Further Testing*

The HCI team that conducted the CAIB IO use study recommended that before the final commercial version was released the ontology would be tested with experienced investigators to make sure that documents in the ontology

would be classified in a manner consistent with its current design. After the ontology was provided to Xerox for inclusion in a commercial version of IO, the HCI team conducted a second study of an early Xerox prototype, observing it being tested by experienced accident investigators. A result showed that the extent of the ontology was still daunting to users when fully laid out. The study also found that users were confused about which link to use given particular task contexts. In some cases, users did not understand the terminology used in the ontology.

Additionally, Xerox conducted an internal study of use by engineers rather than investigators, and found that the engineers had difficulty discriminating between relations based on their labels. This made linking items challenging. Creating items was similarly difficult due to the large number of candidate classes.

NASA is continuing to revise the IO ontology based on the results of these analyses. One significant change under way includes definitions with each of the ontology's classes and relation types. Instead of relying solely on labels for classes and relations, users will be able to see a description with either a click or a mouse over in the IO interface. These descriptions will be linked together whenever possible through linguistic taxonomies such as WordNet [6].

#### 4. THE IO ONTOLOGY MODELING SYSTEM

All elements of the ontology were represented using the IO ontology modeling system. This system reuses the IO instance creation interface to create and manage the IO ontology, i.e. the investigation ontology is modeled and structured in IO just as the investigation information is. This eliminated the need to develop a specialized editor, and provided a framework with many desirable qualities, such as supporting rapid, concurrent, and distributed development. Additionally, elements of the ontology are stored in the same database as instances in the IO knowledge base for all investigations based on the ontology.

The expressive power of the IO ontology modeling system is roughly equivalent to that of Resource Description Framework Schema (RDFS) [7]. Plans for development include the adoption of the World Wide Web Consortium's Web Ontology Language (OWL) representation [8]. The IO ontology modeling system is similar to other knowledge representation systems in that there are three types of items: classes, literal properties, and relations. Classes represent the *types* of concepts modeled in the system; some examples are "Person", "System", and "Causal Model". Relations represent the types of links between instances of classes; an example is "Authored-By", a relation between "Person" and "Document". Literal properties represent the attributes of the instances; an example is "Phone Number" is a literal property for "Person". The classes are organized in a strict class hierarchy; for example, "Review Document" is a subclass of "Document" which is a subclass of "Evidence". Each subclass inherits all literal properties and relationships from the superclass and adds more of its own. Multiple superclasses for a single class (multiple inheritance) are also supported.

One of the unusual aspects of IO's ontology system is that all classes in IO have a corresponding "folder" class. The semantics of these folder classes turned out to be surprisingly complex. They were originally intended to represent sets of items of a given type (e.g., a set of hypotheses) to simplify certain interface interactions. The folders were defined to contain only items of one type (or any of its subtypes). Folder classes have only the basic metadata defined for all types of items but share the same relations as their corresponding class. This allows users to make links over sets of items, (e.g., to create a link between a mishap event and a related folder of hypotheses). However, links between folders were forbidden because the

implied relationship between their enclosed elements was unclear (i.e., was the link to be applied to all combinations of elements in the linked folders or just some combinations?). Early attempts to implement folder semantics in code proved difficult to maintain as the complexity of the interface grew. Instead, the IO team developed an inference agent that used rule sets to "compile" a simpler specification into a complex representation of the ontology that encapsulated the correct folder semantics. The rule set was complex but was relatively insulated from changes in the interface, eliminating the ontology-code maintenance issues.

Different groups of users frequently wanted distinct labels for the same class in the IO ontology (e.g., the same class might be variously termed "fault tree node", "causal model node", or "fault tree element"). IO customized the presentation of ontology classes for different groups of users through an aliasing mechanism called *profiles*. Each profile represented display preferences for sets of classes, so that teams could use unique, familiar terms instead of the system-defined class names--for instance, allowing the class "Event Sequence Node" to be referred to by one group as "Event Sequence" and by another as "Potential Event Tree." The profile mechanism only customized the presentation of classes. Relations and literal properties are consistently displayed to all users who can view the associated classes. Profiles allowed IO to hide unneeded classes from user groups, an essential feature because Semantic Organizer (the underlying core system within IO) was developed to operate over a single large ontology covering many disparate disciplines, including scientific investigations, robotic exploration, and data mining. A more powerful approach would have been to define appropriate ontologies for each user group and to use ontology alignments [9], or cross-ontology maps, to map these ontologies onto the common ontology. The IO team is considering experiments with this alternative design.

#### 5. LIMITATIONS OF THE IO ONTOLOGY SYSTEM

Limitations in the IO interface and the underlying system increased the difficulty of ontology development and maintenance:

##### *Efficiency*

Built primarily with Java servlets, the interface suffers momentary delays due to network transmission time, browser rendering, etc. Though these delays are rarely more than a few seconds, they become more significant for complex tasks, such as ontology development, that involve more than just a few user interface interactions. In addition, the inference agent used to be several orders of magnitude slower, and it was not uncommon to wait a day for ontology changes to be fully compiled. Refinements to the inference agent have eliminated this problem, but much of the ontology development was done using the slower inference agent.

### *Closed environment*

Despite the similarities in representational power, IO did not support export or import to any standard format. Thus, it was not possible to examine or edit the ontology using other editing tools, such as Protégé. Nor was it possible to develop the ontology in another installation of IO (like on a development server) and import the changes, since ontology was stored together with all data from the investigation itself. This made it difficult for ontology developers to experiment with changes to the ontology without actually making them in the production system.

### *Lack of expressivity*

The expressive power of the IO knowledge representation format was mostly adequate, but the particular ability to restrict the ranges of relations for subclasses was lacking. For example, if "Fault Tree Node" and "Event Sequence Node" are subclasses of "Causal Model Node", and the relation "Causal Model Node" *has\_child* "Causal Model Node" is defined, it would be possible to make a *has\_child* link between an instance of "Fault Tree Node" and an "Event Sequence Node". To avoid this problem, ontology developers would make specialized versions of such relations for each subclass of "Causal Model Node" while avoiding the general case completely.

### *Lack of flexibility*

Moving classes using the IO ontology editing tools was initially not possible and remains a difficult task. Particularly difficult was adding a new class as a super-class of an existing class. It is also non-trivial to change the class of existing instances in the system if a user finds a better class or creates one. Instances could not be mapped from an old class to a new class, nor could links be mapped to new relations or metadata to new literal properties. In addition, the profiling mechanism was restricted to classes and did not include relations and literal properties. This meant that all users who could see a class would see all the same literal properties and relations (if at least one of the range types was also viewable). In many cases in which a user wanted to expand a profile, these system drawbacks prevented him from simply including an existing class or creating a super-class of an existing class to include in the profile. As a result, ontology developers would often create multiple, parallel branches of the ontology for different user groups, increasing ontological sprawl. These limitations also meant that once users realized they did not need a class or relation, it was difficult to remove it from the system without losing data.

### *Incomplete retraction of inferences*

As described above, the ontology modeling system used an inference agent to compile a specification of the ontology into a more complex representation that included the notion of folders. Though the inference agent was robust, the coupled truth maintenance mechanism was faulty and erratic. As a result, removing elements from the ontology

often had unpredictable effects. This also led to ontological sprawl, because ontology developers understandably chose to make new classes rather than make significant modifications to existing classes. In addition, multiple inheritance, though theoretically supported by the system, was deliberately avoided by ontology modelers because it was impossible to predict what would happen when changing classes with multiple parents.

### *No specific support*

Furthermore, the lack of specific tools made ontology development and maintenance more difficult. Until late in the investigations, there was no sufficient way to visualize the ontology, and no way to verify that it was properly set up, except by testing as a user of the targeted group.

## **6. LESSONS LEARNED**

Domain experts are essential collaborators in the process of developing complete and correct ontologies for semantic web applications, supplying knowledge critical for defining specific and unambiguous classes, properties, and axioms. However, the experience developing this ontology shows that domain experts and users have difficulty recognizing the shortcomings of an ontology before they use them in an application, requiring an iteration of steps in the development of the ontology. First, the domain expert, in collaboration with application developers, asserts a new set of knowledge representation statements for the ontology. These statements are then incorporated into the application (through the modified ontology). End-users then formulate new requirements for the application, and relay them to the developers or the domain expert. Finally, the domain expert translates the new requirements for the application into a new set of statements for the ontology, and the cycle repeats. In other words, the ontology cannot be fully developed without use of the system, and thus, the system often must be deployed with an incomplete ontology.

The iterative nature of the ontology development process means that features that provide ontology modeling flexibility are of high value and should be a development priority in ontology modeling system. The ontology development environment should support quick and easy specification changes, including moving or deleting classes, properties and axioms. For example, the ability to move classes easily around in class hierarchies is highly desirable. Domain experts often decide to change properties of new classes once they start using them. Those classes might then be more appropriately placed elsewhere in a class hierarchy. If relocating classes (or instances of classes) is difficult, the amount of work required to meet the knowledge representation requirements for the application will increase dramatically. Similarly, changing the name (i.e., the display label) of a class should be simple and should not require other changes to the ontology. Not only does easy relabeling of classes speed ontology development, it also promotes knowledge reuse; different groups of

domain users frequently want different labels for the same sets of classes.

## 7. RECOMMENDATIONS FOR FUTURE WORK

### *Develop Consistency Checking Tools*

The magnitude and dynamic nature of the IO ontology prevented the ontology engineers from maintaining a familiarity with the entire ontology. Combined with the lack of consistency checking tools, the engineers repeatedly had difficulty finding and understanding the meaning of existing properties of classes in the investigation ontology, which contributed to the following problems:

**Element Duplication--Ontology engineers often unnecessarily created new literal properties, classes, or relations instead of reusing existing elements of the ontology.**

**Element Misuse--Conversely, existing literal properties, relations, and classes were occasionally misused because the semantics were not clear from their labels, and the ontology elements were inadequately defined.**

**Misplaced Classes--New classes were frequently misplaced in the class hierarchy because it was difficult to examine how two or more classes are similar.**

**Flat Ontology--Ontology engineers frequently omitted modeling super-classes of required classes, making it difficult to expand on that work later or to reuse certain elements.**

Some of these observations were due in part to accelerated development schedules and the lack of a hierarchy of relations. However, IO has no ready tools to help ontology modelers by providing a broad awareness of existing properties in the ontology. Such tools could automatically select and propose reuse of existing properties for new classes. They could also point out common properties of existing classes.

### *Balance specialization and generality*

The experience of developing the IO ontology has highlighted the need to find a balance in the specialization of concepts in the ontology. On the one hand, investigators need an investigation ontology that is sufficiently customized for their particular tasks. On the other hand, overspecialization of concepts and properties in the ontology can make an application quite difficult to use. Additionally, overspecialized classes can be more difficult to map to those in another ontology. In some organizations, such as the National Transportation Safety Board, investigation processes are sufficiently standardized that the use of a single standardized ontology for all investigations is practical. In other organizations, such as NASA, the nature of the investigations varies substantially from one type to

the next, as well as over time. Ideally, a method to allow the easy specification of the IO ontology would exist to meet the user's needs during the investigation while automatically maintaining mappings to other standard investigatory ontologies.

## 8. REFERENCES

- [1] R. M. Keller et al., "Semantic Organizer: A Customizable Semantic Repository for Distributed NASA Project Teams," International Semantic Web Conference (ISWC-2004), Hiroshima, Japan, 2004, pp. 767-781
- [2] N. Guarino, "Formal Ontology, Conceptual Analysis and Knowledge Representation," *International Journal of Human-Computer Studies*, vol. 43, pp. 625-40, 1995.
- [3] Noy, N.F., et al., *Creating Semantic Web Contents with Protege-2000*. IEEE Intelligent Systems, 2001. 16(2): p. 60-71
- [4] The most current version of this document is now NASA Procedure Requirement (NPR) 8621.1A, available at [http://nodis.hq.nasa.gov/displayDir.cfm?Internal\\_ID=NPR\\_8621\\_001A\\_&page\\_name=main](http://nodis.hq.nasa.gov/displayDir.cfm?Internal_ID=NPR_8621_001A_&page_name=main)
- [5] R. Carvalho et al, "InvestigationOrganizer: The Development and Testing of a Web-based Tool to Support Mishap Investigations," IEEE Aerospace Conference 2005, Big Sky, MT, 2005.
- [6] Miller, G.A., "WordNet: A Lexical database for English," *Communications of the ACM*, vol. 38, pp. 39-41.
- [7] *RDF Vocabulary Description Language 1.0: RDF Schema*, in <http://www.w3.org/TR/rdf-schema/>, D. Brickley and R.V. Guha, Editors. 2004, World Wide Web Consortium.
- [8] *Web Ontology Language (OWL)*, in <http://www.w3c.org/2004/OWL/>, World Wide Web Consortium.
- [9] Chalupsky, H., E. Hovy, and T. Russ, *Progress on an automatic ontology alignment methodology*, in *ANSI Ad Hoc Group on Ontology Standards*. 1997: Stanford University.

## 9. ACKNOWLEDGEMENTS

The authors are grateful to the following investigation boards: the Columbia Accident Investigation Board, the CONTOUR Mishap Investigation Board, and the HELIOS Investigation Board. Portions of this work were funded through the Engineering for Complex Systems Program.

## 10. BIOGRAPHIES

**Robert Carvalho** is a Computer Engineer with NASA Ames Research Center's Intelligent Systems Division. For more than 10 years, He has supported a variety of NASA projects in various Enterprises. He has received numerous awards for his work on Investigation Organizer, including the NASA Aerospace Turning Goals Into Reality: Associate Administrator's Choice Award.



His research interests include knowledge management for safety and design. He received his B.S. in Computer Engineering from Santa Clara University.

**Dan Berrios** has over 10 years experience in the fields of scientific computing and informatics. He has led multi-center research projects in epidemiology while at the University of California, and managed scientific and technical information retrieval research while at Stanford. He currently develops collaborative information systems for scientists, technologists, and engineers at the



University of California, Santa Cruz. He has a bachelor's degree in mathematics from Brown University, a doctorate of medicine from the University of California, San Francisco, a master's certificate in public health from the University of California at Berkeley, and a Ph.D. in biomedical information sciences from Stanford University.

**Shawn Wolfe** is a computer scientist at NASA Ames Research Center's Intelligent Systems Division. His research interests include the Semantic Web, data mining, information retrieval, information integration and knowledge management. He received his M.S. in Computer Science from the University of Oregon.



**James Williams** is a project manager at NASA Ames Research Center in the Engineering Division. He is also a trained accident investigator and was involved in two type A (highest degree of severity) mishaps for NASA (including the Columbia Space Shuttle accident). He is the Project Manager for Investigation Methodologies and Tools (IMT)

at NASA with specific oversight on the investigation of the

External Tank Foam microstructure. Formerly a nuclear materials research engineer at Westinghouse, James has a bachelor's degree in engineering with advanced work in nuclear materials from Penn State University. He is currently finishing his master's certificate in Materials Engineering at San Jose State University.