Virtual documents that explain How Things Work: 
Dynamically generated question-answering documents

Thomas R. Gruber¹
Sunil Vemuri²
James Rice³
Knowledge Systems Laboratory
Stanford University
gruber@ksl.stanford.edu

Abstract
Virtual documents are hypermedia documents that are generated on demand in response to reader input. This paper describes a virtual document application that generates natural language explanations about the structure and behavior of electromechanical systems. The application structures the interaction with the reader as a question-answer dialog. Each "page" of the hyperdocument is the answer to a question, and each "link" is another question that leads to another answer. Unlike conventional hypertext documentation, the system dynamically constructs answers to questions from formal engineering models.

The work illustrates several of the advantages of delivering product information in virtual documents. Since the documentation is generated on demand from engineering models, the information presented always reflects the current design model of the artifact. Because the documentation is delivered using standard WWW protocols, it can be truly integrated into other WWW-based documentation such as email-based design discussions, version-managed design documents, interactive tutorials, and information retrieval systems. Moreover, delivering product information in the form of virtual documents changes the way that documentation is "authored". Engineers can work in the medium of their practice --- annotated engineering models --- while a virtual document generator handles the rhetorical task of composing information to meet the needs of individual readers.

In this paper we demonstrate the application (with examples that run), describe some techniques used in deploying it on the Web, and discuss general properties of virtual documents exemplified by the system.

Keywords:
Virtual Documents, Modeling, Question Answering, Explanation, Active Documentation, Design Documentation, Interactive Documentation

¹ http://www-ksl.stanford.edu/people/gruber/
² http://www-ksl.stanford.edu/people/vemuri/
Contents

1. Virtual Documents
   1.1 What are they?
   1.2 Virtual documents on the Web
   1.3 The use of virtual documents for design documentation
2. The DME Virtual Document Generator
   2.1 What does DME do?
   2.2 How does a DME virtual document look and feel?
   2.3 How does DME work as a virtual document?
3. Three Virtual Document Techniques
   3.1 A One-button Query Interface
   3.2 Preserving persistent query identity
   3.3 Compilation versus on-demand generation
4. Discussion: Advantages and Limitations of Virtual Documents
   4.1 Integrating documentation with professional practice
   4.2 Presenting information in a collaborative context of use
   4.3 The modeling bottleneck
5. References
1. Virtual Documents

1.1 What are they?

Virtual documents are hypermedia documents that are generated on demand from underlying information sources, in response to user (reader) input. Virtual documents look like conventional Web documents: they can be displayed using standard document viewers and they can be embedded into other documentation webs. Once generated, they may be indistinguishable from hand crafted webs. However, virtual documents go beyond static documents in several respects. Because they are generated from underlying models, they can provide answers to a huge space of potential information needs not enumerated in advance (e.g., a universe of questions bounded by a query language and domain vocabulary). Virtual documents can present up-to-date information in domains where paper documents quickly become obsolete. Because they are dynamically created in response to input, they can adapt the presentation to the reader’s particular information and communication needs. Neil Stephenson’s A Young Lady’s Illustrated Primer [12] is an example of a virtual document.

1.2 Virtual documents on the Web

The World Wide Web (WWW) is a natural habitat for virtual documents. It provides a standard presentation language that allows for user input (HTML⁴), a protocol for delivering machine-generated documents and accepting user input (HTTP⁵ and CGI⁶), and a basis for integrating virtual documents with any other document on a wide area network (URIs⁷).

Perhaps the simplest case of a virtual document is a WWW document that is constructed in response to a search query. For example, Example 1 is a virtual document about the use of the term "virtual document" in sources indexed by the CUI W3 catalog⁸. The page that is returned looks like a document (e.g., a newsletter column that lists announcements). The subject of the document is stable and the document has a persistent name (http://cuiwww.unige.ch/w3catalog?virtual+document), yet the contents are generated on demand and may change over time.

---

⁴ http://www.w3.org/hypertext/WWW/MarkUp/MarkUp.html
⁵ http://www.w3.org/hypertext/WWW/Protocols/Overview.html
⁶ http://www.w3.org/hypertext/WWW/CGI/Overview.html
⁷ http://www.w3.org/hypertext/WWW/Addressing/Addressing.html
⁸ http://cuiwww.unige.ch/w3catalog

* We would like to draw a distinction between virtual documents and the technologies used to achieve the effect of virtual documents. One can use CGI, HTML, downloadable applets, and mediating proxies to generate presentations on the fly. However, these technologies do not inherently produce results that look and feel like documents, are generative (e.g., constructed or composed from parts), and change content in response to reader input. These are the salient properties of virtual documents, however they are implemented.

⁸ http://cuiwww.unige.ch/w3catalog
Another example is the Xerox map viewer\(^9\), which dynamically generates maps from an underlying database. It looks and feels like an atlas, except the pages are custom made in response to queries. In this case, the queries may be specified using form fields and interactive graphic images.

### 1.3 The use of virtual documents for design documentation

Virtual documents are potentially most useful in settings where the readers vary in information needs (e.g., they ask a variety of questions), the information space is large (e.g., polynomial in the terms of a query language), the answers to queries change over time, and the information can be structured in a succinct representation from which answers can be generated.

Consider the problem of design documentation. Documentation about a designed artifact covers a broad range of topics: functional specification, conceptual design, detailed design, components and materials, manufacturing processes, maintenance and operations procedures, diagnosis and repair procedures, and product revision. Typically, though, the static documentation on a design captures only a small fraction of the potentially relevant information. Engineers ask a wide range of questions about a design, from simple queries about artifact structure to complex questions about the intended or possible dynamic behavior [5]. The information about the artifact changes over time as the design progresses from concept to manufacturing, new product versions are designed, and new components become available. Maintaining accurate design documentation is difficult and expensive, and much of the information that an engineer might need is only available by finding and asking people who might know the answer.

Fortunately, in many engineering domains there is a succinct representation of knowledge that can be the basis for generating answers: engineering models. Engineering models include specifications of physical and logical structure, expected behavior, desired functionality, constraints that must be satisfied, and objective criteria that should be optimized.

Engineers use Computer Aided Design (CAD) and other modeling tools to create models that represent their designs. They use simulation and analysis tools to ask questions about possible structures and behaviors. However, engineers typically use a different set of tools --- document preparation tools --- to write specifications, capture decision rationale, and present the results of their formal analyses to colleagues in design discussions. The task of creating designs is therefore disconnected from the task of the creating documentation about them.

Tools that "understand" engineering models can help to integrate documentation with design practice by generating answers from the models, rather than recording and replaying static documentation [5]. For example, Cristina Garcia has created an "Active Design Documentation" system to support the process of parametric design [2]. In parametric design problems, design decisions can be characterized in terms of

\(^9\) [http://www.parc.xerox.com/map](http://www.parc.xerox.com/map)
the values of design parameters. Designers work with the system to specify parameter values, and the system assimilates the decisions into its internal model of the design. The program can then create justifications of the decisions by analyzing the relevant constraints, objective criteria, and preferences. "Readers" can then query the design support system to get information about the design that is up to date and accurately reflects the current model.

The technology to generate answers from an underlying model can be the basis for a virtual document system for design. To build such a virtual document one also needs techniques for (1) presenting answers in a readable, document-like format, (2) eliciting questions from readers, and (3) delivering the virtual document in the same context as other design documentation.

This paper reports on a system with these capabilities, called DME (Device Modeling Environment)\textsuperscript{10} [1, 4, 8, 9, 10]. DME generates explanations about how things work from annotated component and constraint models. It can answer questions about the structure and dynamic behavior of an engineered artifact. It presents the answers as natural language explanations that are filtered and organized for human readability. It delivers the explanations as pages of a hypermedia document, so that each page is the answer to a question, and each link is another question that leads to another answer. It elicits questions from readers by offering a set of relevant follow-up questions in the context of each explanation. Instead of typing in free-form strings, users ask questions by traversing hyperlinks emanating from the current page. It delivers the virtual document in the context of other design documentation, by assigning a persistent, context-independent identifier (a unique URL) to each explanation. This allows one to embed references to pages of the virtual document in other WWW documents. Figure 1 illustrates how DME's question-answering interface is delivered as a virtual document.

\textsuperscript{10} http://www-ksl.stanford.edu/htw/dme.html
Before going into the details of DME, it is worth considering the contexts in which virtual documents of this sort may be used. **Figure 2** shows the role of virtual documents in three kinds of design documentation. One application is a **requirements** document that specifies the intended behavior of a system. Instead of describing behavior in words and drawings, the designer could point to a **demonstration** of the intended behavior that is delivered as a virtual document of a simulation. Similarly, specifications of **operating procedures** make reference to the behavior of a system under certain configurations and operating conditions. References to accurate and operational specifications of behavior are important both for the design of operating procedures and for their delivery in user manuals and training modules. A third use is in **collaborative design discussions**. An engineer might argue for a position in a note or email message by referring to the results of an analysis that are documented by a virtual document. The reader of the message can not only see a nice presentation of results, but can explore the model upon which the analysis depends. **Example 2** is a hypothetical email message that contains embedded references to a DME-generated virtual document.
2. The DME Virtual Document Generator

In this section we demonstrate the DME application and show how it generates virtual documents. In Section 3 we describe some techniques used in deploying it on the Web, and in Section 4 we discuss general issues that are relevant to any question-answering virtual document.

The DME system was originally implemented as a proprietary design support system with a sophisticated graphical user interface. The advent of the WWW offered an opportunity to deliver it to a larger user population on a variety of platforms with minimal computational resources. Delivering the explanation capability in the form of a virtual document raised some interesting challenges in interface design and the use of client server protocols. We believe it was the first question-answering system deployed on the WWW that generates explanations in natural language (the first DME virtual document appeared in late 1993 and has been running continuously since then). This section will explain what it does and show how it works. If this paper is being read on a web browser, the reader will be able to try out the examples of dynamically generated documentation.

2.1 What does DME do?

At its core, DME is a tool for modeling and simulating dynamic engineered systems. Users create models of systems by composing elements from model libraries, connecting them, annotating them, and setting up exogenous conditions. DME can then compute numerical and discrete simulations to predict the behavior of the modeled systems, and perform other model-based analyses [7]. The results of simulation and analysis are presented in the form of explanations. Each explanation is a coherent answer to some query about the modeled system. Explanation types include definition (what is this object or behavior?), qualitative state abstraction (what important events occurred in this state of the simulation), causal influence (what modeled quantities influenced this quantity?), logical precondition analysis (what conditions led to this event?), and scenario summarization (what happened in this simulation?).

DME presents explanations to the user by generating text and graphical images in response to queries. A query is a request for a given kind of explanation (e.g., explain the causal influences on a quantity) instantiated on specific arguments (e.g., the pressure quantity at a given valve at a given point in time). The answer to a query is constructed to present the relevant information at an appropriate level of detail. Causality, salience, level of abstraction, and level of detail are determined using domain-independent techniques. For example, the answer to a causal influence query is a description of exactly those variables that had a causal effect on the quantity in question, as determined by algebraic analysis of the constraint equations. More information on DME and the explanation generation algorithms are available in other reports [3, 4].
The DME user interface organizes the dialog with the user as a question-answering dialog, as shown in Figure 1. Each explanation is the answer to a question. Included with each explanation is a set of follow-up questions. The reader can ask for more information by choosing one of these questions. As a result, the path of questions and explanations is generated on demand on the basis of the reader's choices. If the level of detail is too coarse in an explanation, the reader can click on one of the objects question to get further information. If the explanation answered the wrong question, the reader can select alternate, related questions listed on the same page. If the reader is lost in hyperspace, general navigation options are always available.

2.2 How does a DME virtual document look and feel?

The best way to learn about how virtual documents compare with static documents is to experience them first-hand. Consider the following examples of DME generating explanations on the WWW. The examples are from a simulation of the NASA Space Shuttle's Reaction Control System (RCS), which is a system of thrusters that steer the spacecraft. The intended readers for this virtual document are aerospace engineers designing procedures for operating the system under failure conditions. Each example is a "page" of the virtual document about this system in a given scenario.

Note to the reader: when exploring these pages, notice that each page is rich with hyperlinks to follow-up questions. Remember to come back to the paper! It may be helpful to view the examples in a separate window.

Example 3: A "Title Page" for the virtual document

In this presentation, DME introduces the RCS scenario and provides an ISMAP image of the component topology. A scenario is defined by a particular engineering model under specified initial conditions and simulated over some period of time. In this scenario, exogenous actions occur (operators doing things) as well as events predicted by simulation.

Example 4: Summary of a complete scenario

In this explanation, DME reports on salient events that occurred during the simulation. Although there are over 160 quantities and 150 components simulated over 14 states, DME summarizes what happened in a few pages of natural language. This serves as a kind of "table of contents" for a simulation; each reference to an object, event, quantity, or state is linked to relevant follow-up questions.

Example 5: Summarizing Salient Changes in Simulation State

http://wintermute.ncsa.uiuc.edu:8080/map-tutorial/image-maps.html
At State 10, DME identifies an important milestone in the simulation. In the explanation of what happened, the system reports only the most important events of the state and filters extraneous detail (i.e., although many quantities changed values, only this pressure variable had interesting consequences). The techniques for identifying important events and filtering extraneous detail are independent of the domain and model.

**Example 6: Explaining Logical Preconditions**

In this example, DME explains why a qualitative event occurred (i.e., why the pressure regulator is in its pass-through mode). This style of explanation is obtained by analyzing the logical preconditions of the model fragment that represents the "pass through mode" of the component and then filtering any variables that can be proven irrelevant.

**Example 7: Explaining Causal Influences**

In this example, DME explains that the change in pressure that led to the qualitative change in the pressure regulator is determined by the pressure at the helium tank, because a the isolation valve that lies between them is open (see the RCS system schematic\(^\text{12}\)). Considering that there are over 160 variables involved, most of which are linked by constraint equations to the pressure at the regulator, this is a remarkable reduction of complexity for the user who is trying to understand what is happening.

### 2.3 How does DME work as a virtual document?

The technical problems in generating a virtual document system such as DME fall into two broad categories: answer generation and presentation. Answer generation for DME is the subject of other papers on explanation [3, 4]. Presentation is achieved by delivering DME services on the WWW.

DME virtual documents are delivered on the WWW as servers using the HTTP protocol. Figure 3 illustrates the document generation process. Questions are elicited as ordinary hypertext links (i.e., HTTP GET requests for URLs), and answers are produced in HTML. Each URL encodes a query with its arguments. Given a query, a domain model, and a trace of a simulation on that model, DME composes an explanation to answer the query. It then renders the explanation in HTML and sends it to the WWW client. Rendering in HTML requires linearization of a recursive explanation plan, using a method akin to the code generation phase of a compiler. Section 3 will describe some of the virtual document techniques in employed in this work.

The author of a simulation scenario configures a model of a system and performs some simulation runs. To document a particular scenario, the author issues a command that causes DME to generate an executable program encapsulating that scenario. This program then acts as an HTTP server for the virtual document, taking requests for URLs and returning HTML.

As stated above, each URL encodes a query. The space of queries is parameterized by component, variable, simulation state, and explanation type. The routines that generate natural language take these parameters as inputs. For example, the query answered in Example 7 is "explain causal influences" and its arguments are the component called "primary regulator A" and the simulation state called "State 10".

Using a technique called compositional text generation [4], phrases corresponding to model entities are composed and the resulting sentence structure reflects their model structure. Since each phrase corresponds to a different entity, each phrase is given its own URL. For instance, in Example 7, the phrase "The pressure at the input-terminal of primary-regulator A" contains two anchors, one referring to the pressure (a quantity) and the other to the terminal (a component structure).

For each entity mentioned in an explanation, there is actually a set of relevant queries about that entity in that context. For example, when a component is mentioned, possible queries include what is this component?, what are its subcomponents?, what are its quantities?, how is it modeled?, and what are the constraint equations associated with it? Instead of offering a pop-up menu of questions under each object, DME makes a default assumption about which query is most appropriate in the given context and offers an answer to it. On the answer page, the alternative queries are offered. This technique, called the One Button Query Interface, is discussed in more detail in Section 3.1.
Delivering DME capability as a virtual document using WWW standards freed us from platform, resource, and access constraints. Although DME is a research prototype using proprietary technology, anyone on the Internet can use the service. DME virtual documents have been used in collaborative design along with WWW-based teleconference and group discussion tools (e.g., the MADEFAST\(^{13}\) project). Based on this experience, the Stanford Knowledge Systems Lab\(^{14}\) has expanded the range of services offered to include ontology development and remote access to modeling tools\(^{15}\).

3. Three Virtual Document Techniques

Delivering the DME explanation capability as a virtual document led to the development of some techniques that may be of use to other virtual document systems of the question-answering variety. Here are three.

3.1 A One-button Query Interface

DME offers the user nothing but hyperlinks --- there are no places for typing in queries --- yet every possible query can be reached. This is possible because the set of relevant queries is highly constrained. First, the set of query types (e.g., definitional, structural, causal, logical precondition) are determined by the modeling and simulation architecture, and can be built into the explanation generation algorithms. Second, the space of possible queries is constrained by the context in which an explanation is generated (the specific simulation state). Each phrase that is presented as an anchor in an explanation corresponds to a particular aspect of a specific object in a given state; for each such phrase, there is a small number of possibly relevant questions to ask. For example, it makes sense to ask about the parts of a component but it does not make sense to ask about the parts of a state variable. These constraints reduce the number of possible questions per object to about a handful, which means it is practical to offer an explicit set of alternatives from which to choose. If there were too many possible queries associated with each object, then some other elicitation interface, such as a type-in box, would be required.

In its original Graphical User Interface (GUI), DME presented a menu of possible questions per object in its presentation context. In the HTML interface, these menus are "flattened". For each object type and context, there is a default question. This is the target of the anchor assigned to that object, so when a reader clicks on the object the default question about that object in that context is answered. (See Figure 1.) The question is rephrased as the title of the HTML page, so as to be clear about which question is being answered. On that page, the remaining, less common questions are listed explicitly under the heading "Other Questions". The phrase corresponding to the currently answered question is rendered in italics without its

---

\(^{13}\) http://www.madefast.org/
\(^{14}\) http://www-ksl.stanford.edu/
\(^{15}\) http://www-ksl-svc.stanford.edu:5915/
anchor, which is akin to graying out menu items in contemporary desktop GUIs. There are some questions that transcend any particular object or state. These serve as navigation aids and are listed at the bottom of each page as "Other Options".

3.2 Preserving persistent query identity
Since one of the goals of making the virtual document version of DME was to enable information sharing, it was important that readers be able to share references to information generated by the documents (e.g., Figure 2). The technique we chose for realizing this objective was to provide a unique, context-independent name for each explanation, and encode it in a URL. Users can then browse through the virtual document to find explanations that make the desired points, and save them on hotlists or embed them in other documents. For example, in the email message of Example 2, the designer refers to "the drop in pressure at the oxygen tank". This particular page in the virtual document is not listed in any table of contents or found by string search; it is the endpoint of a path of inquiry through the virtual document taken by the designer in analyzing this case.

To achieve persistent query identity requires that the namespaces for the parameters of the query (e.g., object and simulation state names) must be persistent over invocations of the server and independent of the user. This precludes two techniques commonly used in CGI programs: generating random object URLs on the fly and encoding dialog state in a form using the POST method of HTTP. These techniques do not preserve the mapping from URLs to information across sessions or reboots of the server.

Note that persistent query identity does not require that the results of the query --- the contents of that page of a virtual document --- be identical on every access. One of the purposes of a virtual document is generate new content to reflect changes in the underlying sources of information. The persistency requirement is semantic rather than syntactic: that all responses retrieved under that query identifier are always meaningful answers to the "same" query.

3.3 Compilation versus on-demand generation
Although virtual documents are, by definition, generated on demand, it is possible to precompile some of the information that is computationally expensive to generate. This is a classic software engineering tradeoff: compiling some aspects of the virtual document can improve performance but reduce generality. For any virtual document application that maps URLs to a parameterized space, the compilation/generation mix can be characterized by the dimensions of the space that are fixed for a given virtual document and the parameters that range freely in the URLs. For DME, we chose to create separate virtual documents in which the scenario (the component topology and initial conditions) is fixed, so that we could precompute the numerical integration that predicts changes in continuous variables. This choice is justified because the space of explanations can be generated

---

from a given scenario is large and valuable (e.g., designers do a lot of analysis within a given topology and set of initial conditions). To support explanations of alternative (“what if”) scenarios in the same virtual document for a given modeled system, one would include the parameters that characterize alternative scenarios in the URL encoding of a query. This would introduce other performance tradeoffs that are being addressed in our most recent virtual document systems (e.g., scenarios take time to simulate; how many can be saved for how long?).

The extreme of compilation is to precompute all possible explanations and save the resulting HTML files (i.e., cache the transitive closure over the question-answer relation). As long as the URLs are relative (i.e., don't mention the server or an absolute directory path), then this technique can be used to produce a standalone web for a CD ROM or high performance server (we also found it useful for regression testing). For the RCS example demonstrated in this paper, expanding all of the questions and answers produces about 80,000 virtual document pages weighing about 150Mb (much larger than the program that generates the virtual document). However, simply walking the hypertext web does not cover the interactive graphics (i.e., using ISMAP), in which the server maps pixel locations to pages of the virtual document. To precompile a virtual document with interactive graphics would require a standard for declaratively assigning geometric regions to URLs, so that HTML viewers could determine the URL associated with a region on a graphic without depending a particular HTTP server.

4. Discussion: Advantages and Limitations of Virtual Documents

Finally, it is worth discussing in more general terms some advantages and limitations of virtual documents when compared to static documents on the World Wide Web.

4.1 Integrating documentation with professional practice

A classic problem with documentation is that the process of creating documentation is usually decoupled from the process of creating the subject of the document. For example, design has the design rationale problem: it is difficult for designers to record the rationale for their designs in a reusable way without interfering with the design process itself. Not only is documentation viewed as a different task than design, it is done with a different set of tools and skills. This observation might help explain why product documentation is so often out of date and inaccurate.

Virtual documentation may change the practice of design. If virtual documentation were to be generated as a side effect of normal practice, then it could be of use to practitioners while they are doing their work [5]. If design documentation were generated by design support tools, for instance, then more of the designer’s attention could be devoted to design and less to writing perishable documentation. Instead of working in the medium of text, designers would work in the medium of design models (see Figure 4). Documentation derived from design models would also be more authentic, reflecting the current hypotheses and commitments of designers rather than an idealized post hoc rationalization. The standards and installed base
of the World Wide Web now makes it practical to deliver virtual documentation, and integrate it with access to remote engineering services.

4.2 Presenting information in a collaborative context of use

As the ability to generated complex artifacts improves rapidly, our ability to understand or learn to use them does not. User manuals are not a solution, because they present information out of the context of use. Virtual documents offer an opportunity to deliver documentation where and when it is needed, for operator assistance, training, or performance support. For example, some desktop application programs have replaced the paper manual with context-sensitive, customizable help. These are virtual documents for the single user and application. They provide "just in time learning" [6] or software applications.

Virtual documents on the WWW allow the possibility of documentation that is delivered in the shared context of collaborative work. The context of a distributed team collaborating over a computer network includes shared databases and threaded discussions. Virtual documentation can be used to integrate shared information with what people say about it. For example, annotation servers (e.g., ComMentor\textsuperscript{17} [11]) and joint authoring systems (e.g., Hypernews\textsuperscript{18} the KSL Ontology Editor\textsuperscript{19}) generate virtual documents by combining the input of several contributors. Similarly, MUD/WWW gateways (e.g., ChibaMoo\textsuperscript{20}) use virtual documents to describe objects in shared virtual environments. Interactive improvisation (e.g., interactive fiction\textsuperscript{21}, poetry generator\textsuperscript{22}) are other examples where virtual document

\textsuperscript{17} \url{http://www-diglib.stanford.edu/COMMENTOR/}
\textsuperscript{18} \url{http://union.ncsa.uiuc.edu/HyperNews/get/hypernews.html}
\textsuperscript{19} \url{http://www-ksl-svc.stanford.edu:5915/}
\textsuperscript{20} \url{http://sensemedia.net/sprawl/}
\textsuperscript{21} \url{http://www.awa.com/stories/stories2.html}
\textsuperscript{22} \url{http://www.nando.net/toys/cyrano.html}
technology blurs the distinction between documentation as a repository of knowledge and documentation as a medium for collaboration.

**4.3 The modeling bottleneck**

An inherent limitation of virtual document systems is also their source of power: the underlying models. While systems such as DME can mechanically generate text and graphics, it is not easy for humans to create the models that are the source of information for virtual documents. What was an *authoring bottleneck* for static documents is displaced by a *modeling bottleneck* for virtual documents. However, in domains where the model is stable and amenable to formal representation, there is a tradeoff point where it is more cost effective to build a model library and virtual document generation program than to maintain a complex web by hand. Understanding where this tradeoff lies, and devising ways to lessen the burden of model building, is an area for future research.

**Acknowledgments**

The work described in this document was conducted at the Stanford Knowledge Systems Laboratory, under funding from ARPA and NASA under NASA grants NAG 2-581 (ARPA Order 8607) and NCC 2-537. The DME system described is a product of the How Things Work project, under the direction of Richard Fikes. The code and algorithms for explanation generation were developed by Tom Gruber and Patrice Gautier. The code and algorithms for putting it on the web was developed by Sunil Vemuri, Tom Gruber, James Rice, Patrice Gautier and Yves Peligry. The simulation and model formulation approaches in DME are largely due to Yumi Iwasaki. Ken Forbus, Dan Russell, and Philippe Piernot have also been very influential in this work.

**References**


Examples

Example 1:

Please enter a search word/pattern or provide a Perl regular expression:

Submit

NB: Searches are case-insensitive.

Result of search for "virtual document":

February 17, 1994:

- **Virtual documents** about how things work. Using model-based reasoning and machine-generated explanation technology, these documents explain the behavior and structure of engineered systems (e.g., thrusters on the NASA space shuttle, power generation plants). They dynamically generate answers, in natural language, to questions as the reader browses the document. These are true virtual documents in that they are generated from underlying models and are not simply remote user interfaces to conventional software. (nwn)

This file was generated by hgrep v1.6a.
Example 2:

Embedding References to Virtual Documents

Here is a hypothetical e-mail message between engineers who are collaborating on a design. Within the message are hypermedia links to a virtual document generator.

The message is about the Reaction Control System (RCS), the network of tanks, valves, and thrusters that are used to steer the NASA Space Shuttle. Using hypermedia links (shown as highlighted text), the author refers to a "scenario" and to specific events in it (the drop in pressure, the operator response, the result on the system, an alternative scenario).

In this example, the engineer has tested an operator procedure on the RCS, and is sharing the results with colleagues. When the reader traverses one of the highlighted hypermedia links in the email message, DME generates the corresponding explanation as a virtual document. Each virtual document offers a set of follow-up questions, leading to further explanation.

Ralph Johnson <johnson@rcs.ss.nasa.gov>

- Mail folder: RCS Design Team Email
- Previous Message: Fred Smith: "RCS testing schedule"
- Next Message: Mary Jones: "Cost evaluation of recent design change"
- Reply: Fred Smith: "Re: Operator response to RCS system failure"

Date: Thu, 21 Oct 93 17:29:19 PDT
From: Ralph Johnson <johnson@rcs.ss.nasa.gov>
To: RCS design team <design-team@rcs.ss.nasa.gov>
Subject: Operator response to RCS system failure

Fred,

Here is a scenario in which the RCS has a leak in one of the legs near a thruster.

In the training session, the operator sees the drop in pressure at the oxygen tank and then closes the valves in the order prescribed by procedure RCS-OP-24.5. As shown in the scenario at State 4, the operator closed the valves on the thrusters first. Then the operator closed the isolation valves at the thruster manifolds, working toward the tank (see State 6).

After closing the other valve using this breadth-first, upstream strategy, you can see that the pressure is stabilized.

If the operator had closed the valves in a different order, allowing the pressure to drop too quickly in the thruster connected to the leaking leg, then that thruster would have started to cavitate. This scenario is shown in RCS-scenario-38.

Ralph
RCS leak scenario

Introduction

Welcome to the RCS leak scenario. This scenario describes how a leak is tracked down in the Reaction Control System (RCS) of the NASA space shuttle.

This scenario is produced by a numerical simulation using the Device Modeling Environment (DME) developed at the Knowledge Systems Laboratory (KSL) at Stanford University.

Starting Points:

- See a summary of the simulation
- Describe all components in the Initial State
- See the initial conditions
- How to use these explanations

RCS System Schematic

You may click on a component at any time to get more information about that component.
Summary of the complete scenario

Initially:
- primary-regulator A was regulating normally,
- thrusters 1, 2, 3, 4 and 5 were working normally,
- secondary-regulator A was in pass through mode and
- the quad-check-valve was operating normally.

At $T=5.00\text{ s}$ (State 2), the operator modified the leak of leg to thruster 1. As a result, leg to thruster 1 was leaking seriously and no longer airtight. As a consequence the quantity of gas in the helium-tank ($N,He$-tank) was decreasing significantly faster and the quantity of gas in the oxygen-tank ($N,O_2$-tank) was decreasing significantly faster.

At $T=9.00\text{ s}$ (State 4), the operator modified the statuses of thrusters 1, 2, 3, 4 and 5. As a result, thrusters 1, 2, 3, 4 and 5 were closed and no longer working normally.

At $T=22.00\text{ s}$ (State 6), the operator modified the statuses of isolation-valves for manifolds 1, 2, 3, 4 and 5. As a result, isolation-valves for manifolds 1, 2, 3, 4 and 5 were closed and no longer open. As a consequence $N,O_2$-tank was sharply increasing.

At $T=35.00\text{ s}$ (State 8), the operator modified the status of isolation-valve for manifold 1. As a result, isolation-valve for manifold 1 was open and no longer closed. As a consequence $N,He$-tank was decreasing significantly faster and $N,O_2$-tank was sharply decreasing.

The system was simulated from $T=35.00\text{ s}$ to $T=37.91\text{ s}$, when the pressure at the input-terminal of primary-regulator A ($P_{in}/P_{reg-A}$) reached $1.75E+6$, which is one of its landmarks. As a result, primary-regulator A was in pass through mode and no longer regulating normally.

At $T=40.41\text{ s}$ (State 11), the operator modified the statuses of isolation-valves for manifolds 1 and 2. the status of thruster 2. As a result,
- isolation-valve for manifold 1 was closed and no longer open,
- isolation-valve for manifold 2 was open and no longer closed and
- thruster 2 was working normally and no longer closed.

As a consequence $N,O_2$-tank was increasing.

At $T=73.41\text{ s}$ (State 13), the operator modified the statuses of isolation-valves for manifolds 3, 4 and 5. the statuses of thrusters 3, 4 and 5. As a result,
- isolation-valves for manifolds 3, 4 and 5 were open and no longer closed and
- thrusters 3, 4 and 5 were working normally and no longer closed.
Example 5:

What happened in State 10?

New values were predicted as the result of simulation:

the pressure at input-terminal of primary-regulator A crossed 1.75E+6 Pa, which is one of its landmarks.

The behavior of a component changed:

- primary-regulator A is now in pass through mode and is no longer regulating normally.

Other questions:

- What Happened Next

Other Options:

- Describe all components in State 10
- Show Schematic for State 10
- Jump to State: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14
- Go back to scenario introduction
- How to use this document

This page was generated by DME on Jul 14, 1995 15:10 -0800
Example 6:

**What caused primary-regulator A to be in pass through mode?**

In State 10 (T=37.91sec),

- primary-regulator A was in pass through mode

because

- the pressure at the input-terminal of primary-regulator A was 1.7534753943398106E+6 Pa.

**Other questions:**

- Why is this happening
- Display this model's preconditions
- Display equations
- Show Source Definition

**Other Options:**

- Describe all components in State 10
- Show Schematic for State 10
- Jump to State: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14
- Go back to scenario introduction
- How to use this document

*This page was generated by DME on Jul 14, 1995 15:11 -0800*
Example 7:

**Why primary-regulator A is in pass through mode**

In State 10 ($T=37.91\text{sec}$), for primary-regulator A to be in pass through mode, the following condition must be met:

The pressure at the input-terminal of primary-regulator A ($\text{Pin[Pr-reg-A]}$) has to be $\text{Q}< 1.75\times 10^6 \text{ Pa}$;

- $\text{Pin[Pr-reg-A]}$ is equal to
  - the pressure at the output-terminal of the helium-tank ($\text{POut[He-tank]}$)

because
  - isolation-valve A was open

leading to the equation:
  - $\text{Pin[Pr-reg-A]} = \text{POut[He-tank]}$

**Other questions:**

- Why is this happening
- Display this model's preconditions
- Display equations
- Show Source Definition

**Other Options:**

- Describe all components in State 10
- Show Schematic for State 10
- Jump to State: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14
- Go back to scenario introduction
- How to use this document

This page was generated by DME on Jul 14, 1995 15:12 -0800