

SECTION 3

PART II: AMI CONCEPTUAL FRAMEWORK

3.1 FRAMEWORK OVERVIEW

The TO authoring process, as described in the Appendix, is a labor-intensive process. The author must compile information from existing technical and LSA manuals, engineering drawings, and various other sources to produce a specific maintenance procedure. Once the procedure is initially authored, it must be validated by comparing it with design documents and safety guidelines, or by actual demonstration on a physical system. If a problem is detected, the author modifies the TO and re-validates the procedure. The cycle is iterated until the maintenance procedure is successfully validated. Once validated, TOs are published by the airframe manufacturer (vendor) and delivered to the Air Force for follow-on verification of the adequacy and accuracy of TO procedures.

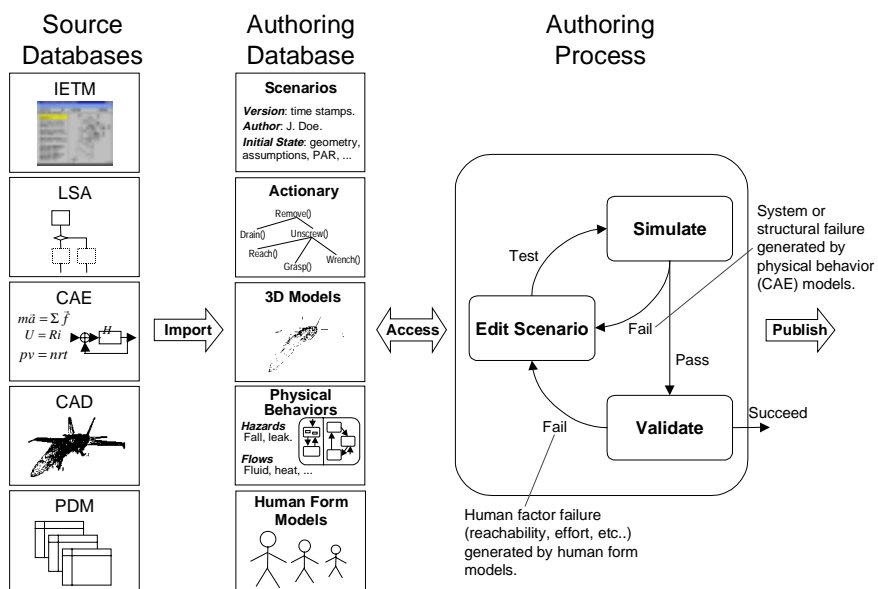


Figure 2. AMI Framework.

Our AMI framework (see Figure 2) will define the concepts and processes necessary to implement a desktop TO validation and verification tool that could support the TO author

in the validation process, and ultimately reduce the time and cost associated with the validation of TO procedures. Such a tool would test a maintenance procedure by simulating it in a 3D virtual environment. The assessment would cover both human factors and system domains. A variety of human models would be used in the simulation to ensure that the ergonomic requirements (visibility, accessibility, effort, and exertion) are within occupational standards. Likewise, the simulation would test whether the procedure is feasible with respect to the maintained systems. Using physics-based models, the simulation would detect unfeasible or hazardous actions.

The main benefit of our framework is to complement current TO authoring systems with a desktop validation tool that would enable rapid prototyping and development of maintenance procedures. The first iterations of the traditional “generate and test” approach would take place entirely at the author’s workstation: the author would edit and simulate a procedure until it passes a validation test. Only a few final validations would need to be done using manual methods such as engineering review or demonstration.

Although the framework is meant to complement existing TO authoring tools, it would make sense to specify the validation tool as a stand-alone application with authoring capabilities. Furthermore, entities could also use the tool in their verification process. The validation may also be stored and replayed. We eventually expect manufacturers and regulating authorities to endorse or certify maintenance procedure validation tools for adhering to industry and government standards.

It is difficult to envision the precise boundaries of the validation tool’s specifications on a conceptual level. However, it is clear that the validation tool would have to interface with other engineering or authoring applications. We anticipate that such integration would be achieved with a PDM system, which could potentially position the maintenance authoring process as a co-design activity early in a product development cycle.

3.2 FRAMEWORK COMPONENTS

Although existing technology does not allow TO authoring to be automated, we propose to simplify the task by automating the validation step. Our framework relies on simulation to apply a validation process by demonstration in a desktop VR environment. We

can simulate a maintenance procedure from a system or human factors perspective to recreate the working conditions of an actual technician (see Figure 3). As in a live validation, the simulation will be successful if the procedure can be completed with the desired result and without hazardous consequences.



Figure 3. Human model at work.

The framework is meant to bring the benefits of verification to the author's desktop. Although it would not serve as a total substitute for physical testing, we expect that most hazards and unfeasible subtasks would be detected. With such a validation tool, an author would be able to repetitively edit and test procedures until they pass desktop validation.

In addition, the 3D representation of the maintained system would enable the author to gain considerable insight into the geometric complexity of the task. The author could survey the scene through the eyes of the virtual technician or from any other point of view. Furthermore, transparency effects and swept volumes could help locate hidden components and evaluate motions in confined spaces.

The simulation will also encompass physical system behaviors. By modeling the system's behavior as it is being operated or maintained, hazards or system-related failures can be detected. Furthermore, model-based reasoning techniques would allow automatic annotation of the sequence of events that led to a hazard or failure.

These geometric and system debugging functionalities will greatly simplify analysis and repair of a procedure.

3.2.1 Validation by Simulation

The framework possesses the ability to simulate a maintenance procedure that would be a viable replacement for live validation. As in validation by actual demonstration, the validation tool will be deemed sound but not complete. In other words, although all the behaviors it produces are realistic (not spurious), it cannot prove that any strict interpretation of the procedure by any technician under any compliant input conditions will be successful.

The soundness of a simulation relies on the fidelity of the model with respect to the system it represents. High fidelity models are complex and require large computational power, which might not be available to produce interactive simulation. It is the responsibility of the modeler to find an acceptable compromise between fidelity and speed.

3.2.2 Proof of Soundness

The only thing an actual or virtual validation by demonstration can prove is that a maintenance procedure is not sound through the detection of action failures or hazards. In other words, a procedure is believed sound until proved unsound. An action fails when its expected effect cannot be observed or when its input conditions cannot be achieved. A hazard is an unwanted physical process conducive to property damage or injury.

System-related failures and hazards independent of the human model will be detected during the first run of a simulation. However, a simulation must be run with different human models that represent the technician population. If all simulations are successful, the procedure can be deemed sound. This need for multiple runs is synthesized in the Simulate and Validate states shown in Figure 2.

3.2.3 Framework Functions

The framework's core process is a simulation generated by a simulator. The simulation corresponds to the execution of a maintenance procedure. However, the validation tool must also provide the following functions:

- **Editing:** Direct editing of maintenance procedures is necessary regardless of whether the tool is running stand-alone or if the author imports procedures from other authoring applications. In the first case, the author must be able to input a whole procedure by using

the system's model library to compose it. In the second case, imported procedures must be cleaned up and dressed up to cover the simulation domains that were ignored by the source application, such as geometry. Editing must be validated by syntactic and semantic checks that allow the author to verify the correctness of the procedure model's form and content.

- **Debugging:** In the case of an action failure or hazard, the validation tool should stop the simulation and help the author isolate the problem. This is the first step toward correcting the procedure. Although geometric reasoning is still too complex for online debugging, causal reasoning of system behaviors is not. A computer can easily reason the sequence of events that triggered a procedural failure and assist a author in isolating the cause. Nevertheless, except for trivial errors, we should not expect the validation tool to automatically repair a procedure.

Although not the focus of this research, the validation tool could also be used to produce media for electronic technical or training manuals, including animations, still images, or interactive simulations. We did not list *publishing* as a function, because it is not strictly necessary for validation. Nevertheless, we should expect publishing to be available under some form in the framework. For example, it could be used to communicate system failures to a design team.

3.2.4 Procedure Diagnosis

Procedure diagnosis takes place during debugging when an author encounters an error and tries to determine the cause of a faulty procedure. Unlike traditional system diagnosis (such as aircraft fault isolation), procedural faults can be attributed to either inaccurate or ambiguous TO procedures, or an error on the maintenance technician's part in not following a validated, published TO procedure. The latter problem is not going to be resolved by a TO validation tool, and is outside the scope of this research. However, the former case can be addressed to some extent by a TO validation tool. For example, assume the maintenance technician is following the step-by-step TO procedures for a task in a suystem. If a component burns out while performing the maintenance task, the relevant failure is not an actual component failure, but rather an induced failure that may have been by an omitted step in the TO procedures (e.g. not turning power off to the aircraft or system that caused the

component to short out) The flaw in the TO procedure could very well be attributed to the author's lack of insight into the different types of hazardous conditions associated with performing the task. In this case, a validation tool might help improve this situation by allowing the TO author to simulate and visualize different scenarios and conditions for accomplishing the task to determine the safest set of conditions, as well as the proper sequence of steps for performing the task

The validation tool is similar to a software development environment. It records a simulation trace, which is analyzed for debugging purposes. The models used in the semi-qualitative simulator can be automatically analyzed along with the simulation trace to help the author locate a problem [9]. The system assists the author by answering standard queries about the function of a device or the factors influencing its behavior. Because of their explicit representation, PAR actions can be included in this reasoning. These questions can also be used as online technical documentation.

Although these self-documented models might help the author understand how a device works, a minimum engineering background will be required to use the tool efficiently.

Current limitations in geometric reasoning do not allow similar debugging facilities to isolate geometric faults. Therefore, only system-related faults can be semi-qualitatively isolated.

3.2.5 Action Failures and Hazards

Some hazards and action failures could be turned off or ignored by the author to focus on specific aspects of the validation. However, if too many of these events are ignored, the simulation might deviate from its expected realistic behavior and compromise the fidelity of the validation.

We recommend a tight *edit-validate-debug* cycle where the author aborts validation at the first error. This argument provides an extra incentive for including substantial editing and debugging functions in the validation tool.

3.2.6 User Interface

The nature of the framework's user interface remains to be chosen. We expect the tool to use a graphical interface through which a maintenance procedure could be represented in three possible modes:

Graph: A procedure is represented as a flowchart. The author builds a procedure by dragging, dropping, and connecting icons representing actions and sequencing constructs (used in LSA and IETM).

Script: A procedure is a script written in a specific high-level programming language. The author must be familiar with the language to key in the procedure.

Free Form Text: The author types or dictates a procedure in plain English (natural language). The interface translates the text in an adequate internal representation (script or graph). Reliable natural language processing (NLP) should be available in near- or long-term. Aside from providing a user-friendly interface, NLP would allow legacy TOs to be imported in a textual or semi-structured form.

All three modes could be combined or layered to offer multiple levels of representation. The graph and natural language modes are the most user-friendly. However, current technology can only deliver a combination of script and graph modes. We propose using the PAR [10] language for scripting human actions.

The user interface should also provide the ability to navigate and modify the virtual world in which the procedure takes place. Ideally, the objects in the scene should be "smart." Their relative geometric positions should be reflected into a corresponding physics-based model. For example, if a plug is inserted in a socket, the simulator should establish an electrical contact between them.

3.2.7 Translation to PAR and Execution of Textual Orders

One essential requirement of the validation tool is to provide the author with a high-level scripting language to control a virtual technician. This language is defined by a set of complex procedural actions and composition operators to sequence them. Each action can be defined by a sequence of lower-level actions, or a direct call to one of the agent's basic skills.

Since our intelligent agent must behave like an average technician, any action vocabulary must be equivalent in both the semantics and level of abstraction to the actions conveyed in textual orders. In a PAR of this scripting language, the high-level action vocabulary, as well as all of the underlying actions are stored in an *Actionary*, which represents the procedural knowledge of the software agent controlling the virtual technician.

Regardless of whether translation from text to script is manual or automated, the closer the Actionary is to the usual TO vocabulary, the better the translation will be. Ianni's specifications [11] exemplify the basic action vocabulary of a virtual technician.

If the translation is manual, the author is responsible for performing a realistic semantic mapping between the text and the actions composed in the corresponding script. If the translation is automated, the validation tool could assess the clarity of the order by exposing potential ambiguities or inconsistencies.

Figure 4 depicts the whole translation and execution process. The author creates a new subtask and fills it with a textual order. The order is translated into a compact PAR script, which is expanded into a set of sub-actions during execution. The software agent performs the primitive actions of the expanded plan. If the effect of the action is hazardous, the author revises the order and starts over.

Translating a procedure consists of scripting each of its subtasks individually. Technically, a whole TO could be modeled as one script made of the sequence of subtasks' scripts. Figure 5 summarizes the different translation steps from a TO to its simulation.

Natural Language to PAR Conversion and Execution

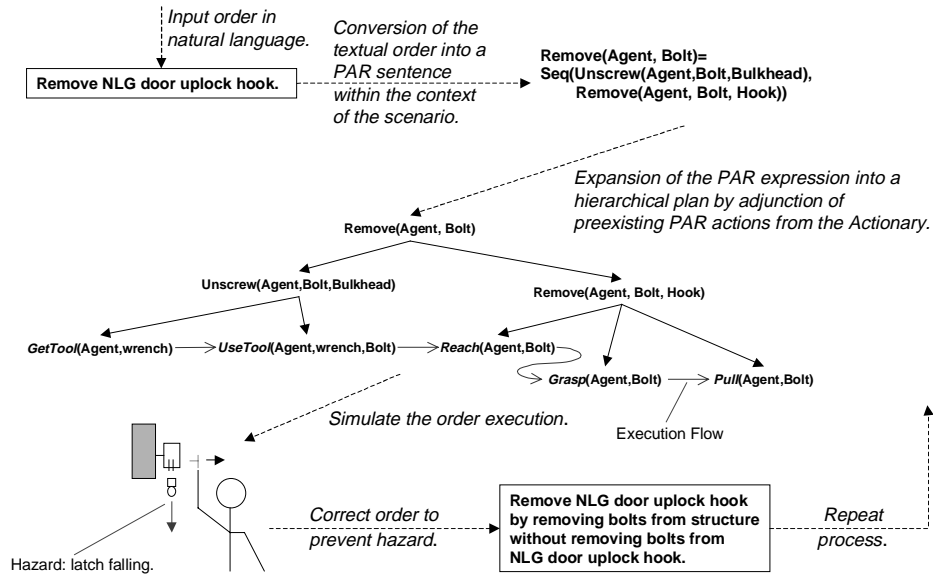


Figure 4. Translation and Execution of Natural Language Orders.

3.2.8 Model Storage and Importation

Each procedure is modeled as a *task scenario*. This top-level data structure is the equivalent of a TO. It defines the input conditions of the procedure by specifying initial conditions and the action sequence to perform. The initial conditions refer to the systems, tools, modeling assumptions, human forms, and metadata necessary to set up a simulation.

The simulator uses various modules to generate human and system behaviors, as well as hazards and failures. Each of the modules covers a specific domain of the simulation: geometric, physics-based, and intelligent agent.

The corresponding models are fetched from the *authoring / simulation library (ASL)* to build the corresponding simulation model. The ASL is a “backlot” of models reused across scenarios. The references between models are closed. This means that all the necessary information is in the database. In other words, the validation tool can run as a stand-alone application.

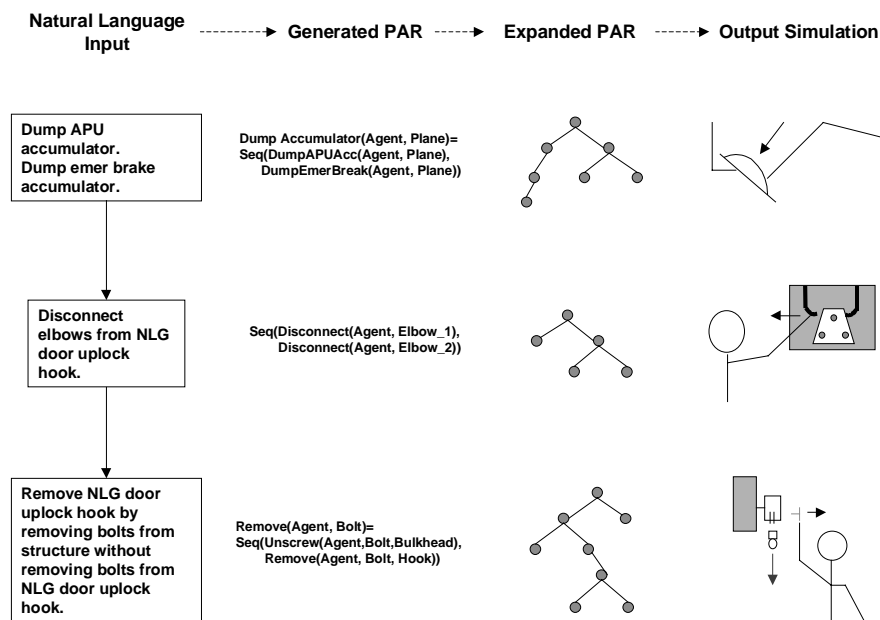


Figure 5. Translation and Simulation of a TO.

Similar to the maintenance procedure author, the framework feeds off source databases to populate the ASL. These databases are maintained by domain specific applications: TO authoring (IETM, LSA); CAE; CAD; and PDM.

The records from input data sources might require specific conversion operations before being stored in the ASL. For example, geometric models must be simplified to allow real-time rendering. The complexity of the importation process is highly dependent on the difference between the format of each particular data source and ASL internal models. Some of these import steps might have to be accomplished manually if the data representations between the source data and ASL differ beyond what can be automated.

We expect the validation tool to interface with each applicable data source (e.g. LSA tables containing task narrative descriptions, etc.) through an enterprise-wide PDM system. This would keep the source data and ASL in synch and trigger the necessary re-validations when an update to a TO procedure occurs.

3.3 SOURCE DATABASES

Figure 2 identifies four kinds of source data necessary to support the framework:

- **TO Data:** The TO data contains a description of the maintenance procedure in a format similar to LSA or IETM. Each description contains the input conditions (maintained system, spare parts, number of technicians, etc.) and a step-by-step narration of the procedure. It can be seen as a semi-structured document containing tabular information (various references, etc.), plain text (subtask narration), and pictures (schematics).
- **Computer Aided Design:** CAD data describe the shape and structure of the maintained systems. Once imported, it will be used to supply geometry to the VR system and model system assemblies. Modern CAD models are parametric; they contain information such as position constraints or dimensioning that can be reused, and component geometry and behavior models.
- **Computer Aided Engineering:** In general, a system CAE model is a block diagram that interconnects quantitative component models. The same topology can be reused in the framework. However, models may need to be simplified to perform interactively while remaining realistic. They also must be upgraded with a qualitative layer. Finite-element CAE models do not correspond to our lumped-parameter framework, and are outside the scope of our framework.
- **Other Product Data:** Other product data is all the data used to complete a simulation model for the validation process. It may include serial or part numbers, references to technical documentation, etc.

Under normal exploitation conditions, the validation tool would routinely exchange maintenance procedures with third-party authoring systems. This should take place under the auspices of the PDM system. CAE, CAD, product data, and any other kind of data necessary to maintain the virtual “backlot” would be imported as needed.

3.4 FRAMEWORK PROCESSES

Simulation can validate both the systems and human factors aspects of a maintenance procedure. The systems aspect checks the soundness of the procedure regarding the maintained system. The human factors aspect checks that the procedure is feasible and safe to perform for a representative set of human models. The human factors check requires the execution and analysis of several simulation runs using technicians of representative

anthropometry for accommodation analysis. This decoupling allows the author to first concentrate on the systems part of a procedure before dealing with specific human factors. We decompose the validation process in two steps:

Step 1: System Dependent Validation (SDV). SDV is a simulation to detect systems failures and hazards. It also detects human factor hazards that are independent of a specific human model. For example, it can detect if the technician is exposed to toxic substances.

Step 2: Human Factor Dependent Validation (HFDV). HFDV is a system validation with a specific human model. It detects the human factor hazards or failures specific to the technician model, such as failure to reach or insufficient strength.

The difference between SDV and HFDV lies in the anthropometric and biomechanic characteristics of the technician, which are relevant in HFDV and not in SDV. For a given procedure, the author will have to run at least one SDV and enough HFDVs for an accommodation analysis.

3.4.1 Editing Process

LSA or IETM data does not contain all of the information necessary to produce a simulation. For instance, the system geometry necessary to render computer graphics and collision detection in a human model is not included in LSA task narrative data. The validation-specific part of the editing process must allow an imported procedure to be completed with the adequate data.

If the validation fails, the author can edit the procedure in the source system. However, in order to use the validation tool as stand-alone, or to quickly re-test a modified procedure, the application should support part of the procedure editing process. In particular, the author should be able to edit a subtask sequence as well as its caution and warning messages.

3.4.2 System Dependent Validation

SDV is related to how simulations are performed. The simulation itself emerges from the interaction of dedicated simulators.

A *scene management system* uses the geometric description of the world to render interactive 3D graphics and detect collisions. A physics-based simulator generates systems behaviors. Finally, an *intelligent agent* drives each technician in the scenario by interpreting the scripted actions or orders.

Simulated procedures interact with each other across domain boundaries. For example, an agent performing an action, such as opening a valve, will generate an animated motion in the geometric domain. The result of its actions also affects the simulated systems (physical or systems domain). This in turn may change the appearance of an object (position of a gauge). Finally, the change, needle motion, can capture the attention of the agent. Having perceived the system's new state, the technician might decide to close the valve. The simulation halts if an action fails or a hazard occurs. The program should then switch to a debugging mode.

The simulation can be broken down into three domains: geometric, physics-based, and agent. These different simulation domains run in parallel and share the state variables common to their models.

3.4.2.1. Geometric Domain. Geometric simulation generates the 3D graphics fed in the user interface. It is produced by a *scene management system* which stores the scene's geometric description in a *scene graph*. This system also is used to detect collisions between simulated solids.

3.4.2.2. Physics-based Domain. A physics-based behavior simulation allows the production of a realistic response from the maintained system to the actions of the technician. This includes simulating physical processes that are reported as hazards (leaks, corrosion, combustion, electrical hydraulic or mechanical failure).

There are two types of physics-based simulations. The first deals with the behavior of physical objects due to their geometry. It prevents objects from interpenetrating with realistic collision reactions and contact forces. The second type is non-geometric and models systems as interconnected functional modules that exchange signals.

Most systems are modeled with non-geometric or lumped parameter models. They are assembled by interconnecting functional modules, as in block diagrams. The modules exchange signals representing flows of matter or energy. This paradigm applies to most

engineering domains (electric, hydraulics, control, mechanics, etc.). Traditionally, simulators use quantitative models, but we recommend semi-qualitative modeling. Semi-qualitative modeling allows physical processes to be simulated independently from the components in which they take place. This essential feature allows specific kinds of processes to be identified as hazards and the simulation to be halted when one of them occurs. For example, one can use a generic model of a fluid flow process and categorize it as a leak if it goes from the maintained system into the environment. In addition, the qualitative part of the simulation can be used as sensory input to the agents in the environment. Finally, a semi-qualitative simulation engine can “explain” the behaviors it generates. This self-explanatory feature is a core element of the system’s debugging functionality.

3.4.2.3. Agent Domain. The main product of TO authoring is a sequence of orders whose strict interpretation by technicians guarantees safe and successful maintenance procedures. The level of detail conveyed in the maintenance procedures must correspond to the skill level of the person performing the task. This can impact the level of detail the author must convey when writing the specific steps for a maintenance procedure. A desktop validation application would require an agent model with similar skills and expertise to interpret and perform a maintenance task in a realistic manner. In particular, interpreting means “understanding” an instruction and inferring the corresponding elementary action sequence. For example, when a technician is instructed to unscrew a bolt, the elementary task of grasping the right tool is not explicitly described. Furthermore, some of these tasks may be optional. In our example, this is the case if the technician already holds the right tool.

We propose to model the agent’s experience and skills with the Parameterized Action Representation (PAR) [10]. PAR would also be used as an internal representation or scripting language for action sequences that are explicitly described in a TO. In other words, in each task scenario, a PAR script would represent the orders stated in the corresponding TO.

A PAR action contains input and output conditions. The action is executed if the input conditions are satisfied. The output conditions are asserted upon completion. These conditions apply to facts about the state of the world known by the agent at the time of execution. For example, an agent operating a valve will be interested in its state (open or closed). These facts are acquired via sensory input simulated as sensory actions, which can

be limited by the situation or capability of the agent. For example, an agent cannot read a gauge if it is out of sight. Alternatively, if the author does not care for sensor modeling, the agent can be omniscient and extract facts from the whole simulation environment at any time.

Some of the input conditions can specify preparatory actions. These condition/action pairs are sub-goaling constructs where the action can be executed to achieve its associated condition if necessary. In our example, grasping a tool before using it is a preparatory action.

An action can either trigger PAR sub-actions or call out a primitive action. Primitive actions are skills such as locomotion, grasp and attention that are built into the agent. Until now, the PAR framework has been implemented with the EAI Jack Toolkit that provides the human model and the aforementioned skills.

PAR interpretation represents the cognitive process of the simulated technician. It drives the actions of the software agent controlling the geometric representation of the technician. A PAR-based agent is *reactive*. This means that its choice of preparatory actions will be based on the state of the world at the time of execution and not on planned or past actions. In other words, PAR actions are pre-set (static) hierarchical plans with optional parts. The agent is responsible for completing an action or reporting a failure, as well as its immediate cause.

While statically-defined actions might be sufficient for most tasks, dynamic action plans will be necessary for complex ones. For example, disassembly requires specific planning algorithms to compute a valid extraction sequence and path for each part of an assembly [12]. One possible solution is to use dedicated *skill modules* to generate PAR actions on the fly. In our disassembly example, a disassembly action would call upon a disassembly-planning module to generate a whole PAR hierarchic plan to perform the task. The action would be, in essence, refined or expanded dynamically. The plan returned by the module would be interpreted as a regular static PAR action.

3.4.3 Human Factor Dependent Validation

HFDV is a series of system dependent validations, each using a different human form model. Each simulation takes into account the specific biometrics of the human model. Most

of the hazards (collision with a moving part) or failures (failure to reach or see) detected at this stage will be caused by geometric factors. Other model-dependent factors such as endurance and strength might reveal that a procedure is too demanding for certain segments of the technician population.

The human models are supported in parametric form by dedicated software such as the EAI Jack Toolkit. The software provides physical skills and capabilities to our virtual technicians. A software agent completes the model with cognitive abilities. As with the biomechanical model, the parameters of the cognitive model could be manipulated to capture different expertise levels. The author could use them to assess whether a TO is explicit enough for an average technician.

HFDV can be automated as a batch process. If the system has a predefined database of representative models, they can be tested in sequence until a fault occurs or until the whole group is exhausted.

3.5 REAL-WORLD APPLICATIONS

3.5.1 Publishing Simulation Graphics

Our framework focuses on supporting the TO validation process. It uses desktop virtual reality to depict the execution of maintenance procedures in a simulated environment. The produced 3D computer graphics appeal to the author's natural geometric reasoning to provide valuable insight into a maintenance task problem. Similarly, electronic maintenance manuals could be enhanced with these graphics to further assist technicians. Likewise, material from failed procedures could be used to communicate maintainability issues to engineering teams.

The published material could range from pictures and movies to animated or interactive 3D environments. It is still too early to say under which form data from the validation tool could be published or exported to another system; however, it will most likely be as an interactive multimedia document. Also, specific data combinations (2D, 3D, sound, and hypertext) could be generated. Such combinations will be platform independent as standards for pictures, movies, virtual worlds; human models will be integrated with semi-structured modeling languages such as the eXtensible Markup Language [13].

3.5.2 Model Importation, Building, and Maintenance

As stated earlier, we expect to use a PDM system to interface the authoring/simulation library (ASL) used to support a TO validation application, with the databases used to support TO authoring and engineering applications used by weapon system manufacturers.

The ASL is populated and updated by importing data (graphical and textual) from sources such as CAD, CAM, CAE, IETM, LSA, etc.). The data from each source must be converted to the ASL format using a specific data import process. Although CAD data is straightforward to convert by automated decimation (see Figure 6), CAE or TO conversion will tend to be more labor intensive. The main
??????????

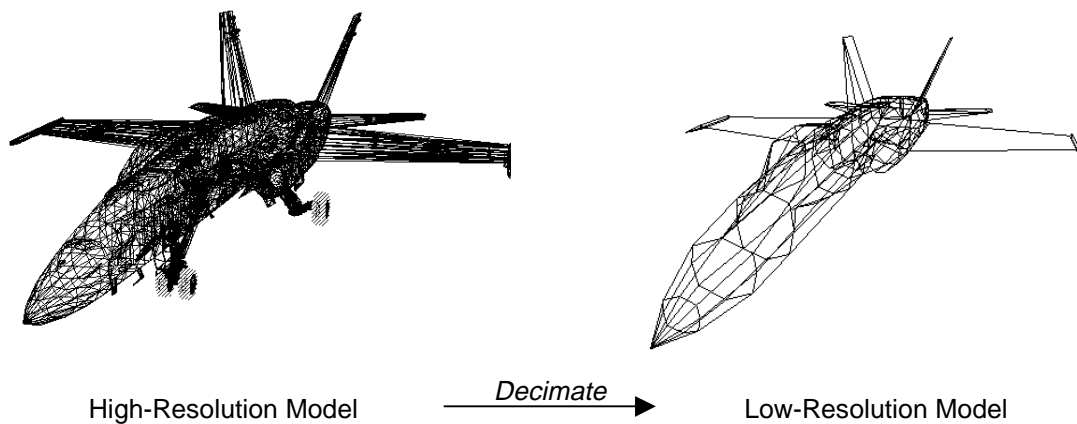


Figure 6. 3D Model Decimation.

reason is that the formats of the source data and ASL are quite different. Natural language processing might assist in converting the narrative parts of LSA and IETM files into PAR.

Data importation will be a major process during the early phase of the validation tool. The virtual “backlot” will be updated with new models each time a procedure refers to a non-catalogued task scenario element. However, this up-front cost will be amortized once the ASL reaches its critical mass. Afterwards, occasional updates will keep the ASL in synch with the rest of the authoring system.

If TOs are edited within the validation tool, they will have to be exported back to the original authoring system (if any).

Manual data conversion may require more time and skills than a single author may have to offer. This implies that support modelers may be required to assist authors in setting up new simulations.

3.6 AUTHORIZING AND SIMULATION LIBRARY (ASL)

Authoring and simulation data support the author-test-validate cycle. The data is obtained through import from other engineering and logistics data sources , or input directly during the authoring activity. In either case, the data is stored as reusable models in the ASL.

We can classify the models by domain and geometric nature. We have three modeling domains: human, system, and task. Each can be divided into geometric and non-geometric components (see Table 1).

Model Domains	Components	
	Geometric	Non-Geometric
Human	Human model	Action representation, human model (biomechanics)
System	Individual system component shape, system assembly	Physics-based component model, hazard models
Task Scenario	Initial environment layout (technician and system position)	Initial environment state, subtask sequence

Table 1. Geometric and Non-Geometric Models.

3.6.1 Task Scenario

A *task scenario* is the internal representation of a TO. It lists the initial state and composition of the environment. In particular, it indicates the number of required technicians, the configuration of the maintained system, and the required spare parts. This information describes the spatial position of each entity having a geometric appearance, as well as its internal state. It also includes the TOs themselves along with caution and warning messages. These subtask sequences are stored in textual and scripted (PAR) form.

Task scenarios are the master data of the validation tool. The rest of the ASL supports them. They are also the subjects of the validation process.

As a master document, a task scenario contains all the references to all the models it explicitly requires. Its metadata tracks their respective versions for revision control purposes. Extra meta-information such as the name of the author and a record of the validation process may be included.

3.6.2 Actionary

The Actionary is the library containing all the PAR actions of the validation tool. It represents the knowledge of the software agent driving the human model. This knowledge must be broad enough to enable a virtual technician and a human technician to interpret an order in a similar fashion. In other words, the Actionary provides a well-founded vocabulary of actions suitable to support direct translation of an order in textual form to a short script.

Each action is a procedure with parameters such as agent, objects, and manner. The agent designates the entity that performs the action. The objects are the entity on which the action is performed. The manner indicates how the action is performed.

An action has input and output conditions. The input conditions are subdivided into *applicability* and *preparatory* specifications. Applicability conditions define the properties that the agent of the object must have by design. For example, the `Open_Container` action will only accept containers that have a lid. Preparatory conditions specify the initial state in which the environment must be to perform the action. In the example, a container must be closed in order to be opened. A preparatory condition can be associated with an action whose execution will satisfy the corresponding condition. This action/condition pair is a way of formulating sub-goaling. In our container example, the `Open_Can` action could have the `Has_Can_Opener/Get_Can_Opener` condition/action pair. An agent would have to get hold of a can opener with the `Get_Can_Opener` action if it started executing the `Open_Can` without one. Output conditions define the effects of an action when its execution is completed.

The Actionary is an action taxonomy in which actions are grouped by categories and subcategories. For example, the `Open` action is refined depending on the type of its

parameters. The most general Open category contains all the Open subcategories. We could have an Open subcategory for opening containers and another subcategory to open doors. The latter category could be refined depending on the way a door opens: rotates or slides (see Figure 7).

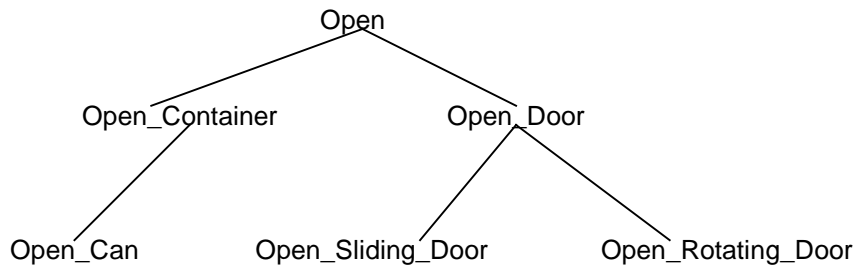


Figure 7. Open Action Taxonomy.

An action inherits the conditions from the action categories to which it belongs. If the Open action requires its subject to be Openable, then all the Open subcategories will perform that check.

When executed, a PAR action can either call a primitive action or execute a sequence of sub-actions (composite action). An action can execute many sequences in parallel. For example, the Remove_Panel action might require the agent to hold the panel with one hand and open its latches with the other. The action's execution sequence would be of the form:

$$\text{Remove_Hatch}(\text{agent}, \text{hatch}) = \text{par_join}(\text{Hold}(\text{agent}, \text{hatch}, \text{left_hand}), \text{Unlatch_Hatch}(\text{agent}, \text{hatch}, \text{right_hand})).$$

The `par_join` construct executes the hold and `Unlatch_Hatch` subactions in parallel. It also ensures that the action completes when both subactions are completed.

Because of its procedural nature, its action composition constructs, and its taxonomic structure, PAR can be used as a scripting language for TOs, as defined by Ianni.

3.6.3 3D Models

3D models capture the shape and structure of the objects represented by the rendering system, including assemblies, tools, and human figures. Although this information is mainly used by the scene management system, it may be accessed by other components of the simulation such as an assembly-planning module.

3.6.4 Human Models

The parametric human model completes a human figure geometric representation with anthropometric data to assess the human factor impact of a given scenario on technicians varying in size, force, and gender. This assessment also includes reachability, effort, and attention. Each individual of the technician population is represented with a unique set of parameters to plug into the parametric model.

There are different implementations of human models, each with its own parameterization. Industry standards are being developed to improve interoperability, for example, by the SAE G-13 subcommittee [14].

3.6.5 Physical Behavior Models

We propose to represent the maintained systems as assemblies of elementary devices. Each device has a model stored in the ASL. Therefore, complex systems can be modeled as a network of interconnected devices. This modeling method is based on block diagrams, and is used in most engineering fields.

We do not need the same accuracy and level of detail as in engineering simulations. We only need physical models that are realistic enough to simulate hazards and action failures that are simple enough to run at interactive rates.

We diverge from traditional engineering practices by advising the use of semi-qualitative models instead of purely qualitative ones. Semi-qualitative modeling allows the numerical behavior necessary for an interactive simulation to be generated and an abstract qualitative representation suitable for the sensory or cognitive tasks of software agents (such as PAR execution, planning, or diagnosis) to be maintained. Aside from its dual representation, semi-qualitative modeling has the following features:

- **Physical Processes Can be Modeled:** Physical processes such as matter or energy flows can be modeled as independent entities. These models are automatically instantiated when the conditions supporting a flow are met somewhere in the simulated system. For example, a fluid flow can be instantiated in any pipe whose pressure gradient is non zero.

- **General Physics-based Models Can be Encoded:** Semi-qualitative models can directly encode the most fundamental behavior of most physical domains. This allows very general model libraries from first principles to be created.
- **Domain Models can be Integrated:** For example, a model library for fluids and a model library for thermodynamics can be combined within the same scenario. Dependencies between domain models can be encoded to provide automated model building.

The use of physical behavior models has two benefits. First, we can model hazards as processes. When such a process is instantiated, it signals itself as a hazard and the simulation stops. For example, any flow of toxic vapors escaping from a pipe can be flagged as hazardous. Second, representing processes separately from the components in which they take place provides a process-centered view of a simulation. This is particularly useful to understand what is happening. For example, it is easy to understand why the fluid level of a tank varies if one knows the active adjacent fluid flows. This type of analysis can be partially automated for debugging purposes.