PRELIMINARY DEVELOPMENT OF AGENT TECHNOLOGIES FOR A DESIGN INTEGRATION FRAMEWORK

Mark A. Hale* and Dr. James I. Craig†

Aerospace Systems Design Laboratory
Georgia Institute of Technology
School of Aerospace Engineering
Atlanta, Georgia 30332

Abstract

Effective design of modern systems requires the systematic application of design resources throughout a product's life-cycle. Information obtained from the use of these resources is used for the decision-making processes of Concurrent Engineering. Integrated computing environments facilitate the acquisition, organization and use of required information. State-of-the-art computing technologies provide the basis for an Intelligent Multi-disciplinary Aircraft Generation Environment (IMAGE) described in this paper. IMAGE builds upon existing agent technologies by adding a new component called a model. With the addition of a model, the agent can provide accountable resource utilization in the presence of increasing design fidelity. Agent fundamentals are illustrated with a zeroth-order agent example. A CATIA™-based agent is described to demonstrates that agent technologies can be scaled to include large and complex proprietary resources. Likewise, multi-proprietary resource systems are demonstrated with an aircraft component modeling system integrating CATIA and ORACLE™ and with a High Speed Civil Transport visualization system linking CATIA, FLOPS, and ASTROS. These examples illustrate the important role of the agent technologies used to implement IMAGE, and together they demonstrate that IMAGE can provide an integrated computing environment for the design of open engineering systems.

Background

Concurrent Engineering (CE) is an engineering approach that formalizes a concurrent decision-making process, as shown in Figure 1.[1] Product and process driven engineering tasks provide information while decision-support methods are used to make decisions. A computing environment facilitates the acquisition, organization and application of information and integrates together the engineering processes.

Many requirements are imposed on a computer environment by the nature of the product and process trades that occur throughout the life-cycle of open engineering systems. The most demanding requirements include the elements of scalability, support for increasing fidelity of design, and support of an increasingly complex product data model.

A number of pilot projects have been implemented that investigate integrated computing issues. These projects include integrated design frameworks (FIDO[2], HiSAIR/Pathfinder[3,4], PACT[5], Designworld[6], Prism[7]), modular conceptual design systems (ACSYNT[8], FLOPS[9]), and quasi-procedural systems (PASS[10,11]). At the Aerospace Systems Design Laboratory (ASDL), two design frameworks are being developed to investigate life-cycle design. LEGEND (Laboratory Environment for the Generation, Evaluation, and Navigation of Design) is a prototype system for quantifying, developing, and instantiating designs.[12] IMAGE[13] (Intelligent Multi-disciplinary Aircraft Generation Environment), which is the subject of this paper, is currently under development by the authors. IMAGE addresses two fundamental issues: formulation of a design model and development of enabling computational technologies to implement a design environment. The latter is the subject of this paper.

IMAGE research has shown that one of the key technologies necessary for implementation of an integrated computing environment is the *agent*. Building on existing agent definitions, IMAGE adds a new component: the *model*. With a model, agents can be used flexibly to generate design information and can be held accountable for their actions. This paper will define key characteristics of these extended agents, show the fundamental importance of agents, and illustrate the benefits of using agents in integrated computing environments.

Agent Definition

Agents are often described by their ability to interact with other agents in a computer environment, as given by the following definition:[5]

> "[An agent is] a computer program that communicates with external programs exclusively via a pre-defined protocol."

Communications is a key requirement, and agents must conform to an ontology. Yet, this definition fails to dictate sufficient requirements for effective support of heterogeneous design resources and decision-making processes.

---

* Graduate Researcher, Student Member AIAA

† Professor, Member AIAA

AIAA-94-4297
5th Symposium on Multi-disciplinary Analysis and Optimization
Panama City, FL, September 7-9, 1994

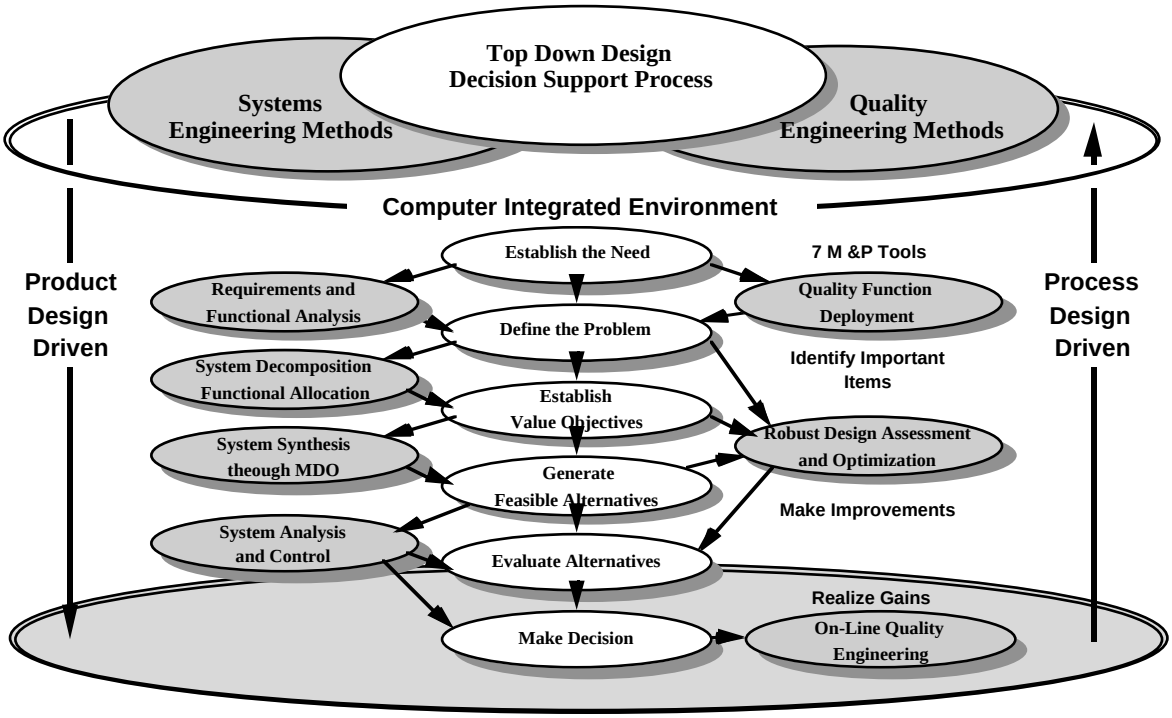


Figure 1. Concurrent Engineering

An expanded agent definition can be found in LEGEND:[12]

> An agent is a resource that has been modeled and wrapped for inclusion in a distributed design environment. The environment dictates requirements for context-oriented documentation and publication and experience mechanisms.

This definition outlines use of resources for the generation of accountable information. However, the definition still lacks specifications to insure that the agent can be used with a variety of resources throughout a product's life-cycle.

The following definition is proposed by the authors:

> *An agent is a resource that has been modeled and wrapped for inclusion in a distributed design environment. Agent design requires a designer-centered, bi-directional wrap that is independent of proprietary boundaries and capable of supporting increasing fidelity models.*

This definition characterizes an agent by its components and behavior. There are three agent components: the resource, the model, and the wrap. Agents must accommodate information obtained from heterogeneous resources and must apply this to design models of increasing fidelity across a product's life-cycle. Agents are one of the key integration tools for a distributed, designer-centered, multi-tasking design environment. It will also be shown later in this paper that agents are able to generate design information and make decisions while maintaining accountability for all actions.

For the first time the role of proprietary resources and information is explicitly stated in the agent definition. Proprietary resources are generally stand-alone in nature, with limited communications capabilities, and preserve software rights through a number of advanced computing techniques. Together, these present a formidable challenge to implementation of integrated design environments. Finally, it should be noted that, in addition, proprietary information must be accommodated and secured in open, integrated environments.

Agent Components

As defined in IMAGE, a generalized agent is either a *tool* or an *agent*. Both incorporate resources and are used to produce design information or make design decisions. A *tool* is the most basic type of generalized agent and is comprised of a resource, typically a computer program, and a wrap, typically program utilities used for communicating with other tools and agents. In IMAGE an *agent,* as shown in Figure 2, is a tool that, along with a resource and a wrap, also includes a model. With the addition of the model, the agent can generate *accountable* design information to be used for making decisions. Accountability was first introduced in LEGEND[12], and accountable information is defined in IMAGE as information with the context in which it was developed. Context, in this situation, includes the "what, why, when and how" information attributes on which accountability can be based.

Agents and tools are the basic elements used in IMAGE to implement an integrated computing environment. Agents operate on the basis of the models that they contain and therefore can provide accountability for the information they

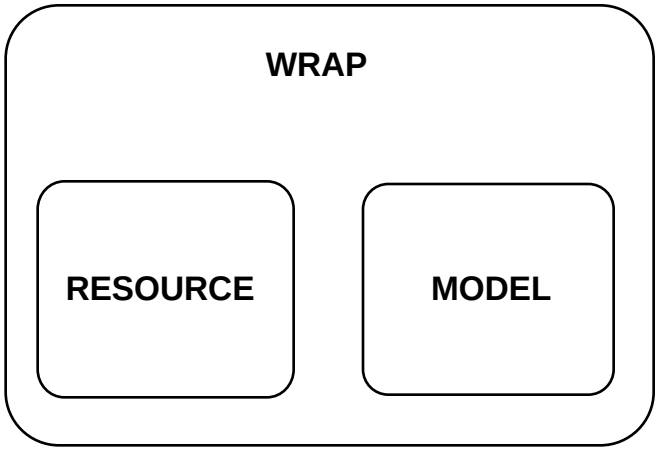


Figure 2. Agent Components

produce. Tools, on the other hand, which do not include a model can produce only information with no context and therefore no inherent accountability. Tools can be as equally useful as agents, but they must be used by either other agents or design experts, either of which must provide the appropriate accountability.

Agent terminology is still evolving at this point in IMAGE development. A few simplified definitions that show the relationship among definitions found in this paper and other definitions are listed below. The following definitions are used in IMAGE:

Agent - A resource that has been modeled and wrapped

Tool - A resource that has been wrapped

The three basic components of an agent are:

Resource - A computer program that does analysis

Wrap - Making resources talk to each other

Model - Engineering assumptions used to describe a process

Another agent definition used by others is the following:

Agent - A resource that has been wrapped (for communication by a pre-defined protocol),

while computer scientists sometimes use the following:

Tool - A resource.

As will be shown below, IMAGE definitions for agents and tools offer advanced capabilities over standard use.

Model

The model adds context to the information produced by an agent and, therefore, provides accountability. Models are typically based on mathematical formulations, engineering principles, or geometrical constructions. In addition, models may also include limitations, units, and details of implementation of the agent.

Resource

Resources are entities that produce additional design information based on existing information. Typical resources include off-the-shelf computer programs such as ASTROS (a structural optimization code), FLOPS (an aircraft convergence code), ACSYNT (an aircraft convergence code), CONMIN (an optimization package), CATIA (a three-dimensional geometric modeling, simulation and analysis package), and ORACLE (a relational database). Often overlooked, the design expert (the designer) and design experience are also design resources. Knowledge-based systems can be used to capture design expert knowledge, while "lessons learned" can be captured in experience-related resources.

Wrap

The wrap manages information generation within an agent and transfer between agents. The wrap implements bi-directional information exchange within the design environment.[4] For computer-based resources, the communication channel needs to be accessible through the multi-user, multi-platform, multi-language, networked workstation systems used in current design systems. A tool that has been successfully used for inter-agent communications in IMAGE is the Tk/tcl utility package developed at U.C.-Berkeley.[14] [Note: Tk/tcl is an interpretive, X11 windowing system.]

The wrap is also responsible for the bi-directional exchange of information within agent resources. Run-time access to agent resources is necessary for the automation of information exchange.

As mentioned earlier, the accessibility of design resources varies significantly between proprietary and non-proprietary codes. Nonproprietary codes are often easier to wrap because source code level access is available. Therefore, wrapping utilities can be directly integrated by restructuring the source code itself. In contrast, proprietary resources are usually provided in an object/executable form. Fortunately, internal resources of mature commercial software products can often be integrated with link-edit procedures, and this can form the basis for agent wrapping.

As will be discussed in more detail later, the use of proprietary resources becomes prevalent in the later stages of design. Therefore, the issues concerned with the wrap of these resources must be addressed if the resources are to be integrated into a design environment. In addition, the ability to wrap proprietary resources presents a considerable step toward design automation.[15]

Agent Importance

There are two fundamental properties that make the agent an effective integration tool in a computing environment. First, agents can provide accountable resource use through the incorporation of the model. Second, new IMAGE wrapping technologies permit agents based on proprietary resources to be utilized at any point in the life-cycle of design.

Resource Interaction

The model provides accountable resource utilization by providing context to information generated by the resource. Context is provided because the agent incorporating the

resource is aware of its scope, assumptions, and limitations through the specifications of its model. Without this structure it is often difficult to avoid eventual degeneration into the "garbage in, garbage out" syndrome.

Life-Cycle Utilization

As shown in Figure 3, traditional systems that employ both agents and tools primarily operate in the conceptual stage of design.[2-10] The design resources used in these systems are mostly non-proprietary codes, as illustrated in Figure 4. However, computing environments must also incorporate proprietary resources that predominate later in a product's design life-cycle, as also seen in Figure 4. As discussed earlier, the integration of these kinds of resources is more difficult than that for nonproprietary resources. However, IMAGE incorporates new technologies to wrap proprietary resources. Therefore, proprietary agents can be used throughout a product's life-cycle, see Figure 3.

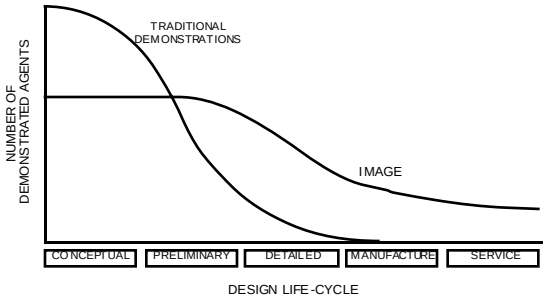


Figure 3. Agent Demonstrations

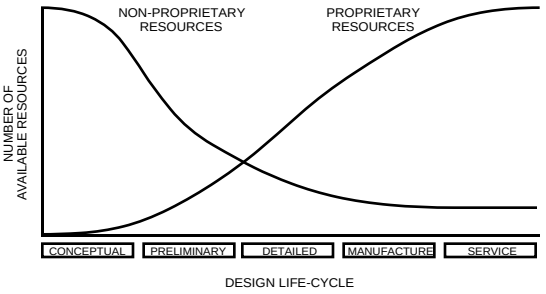


Figure 4. Available Resources

Agent Examples

To demonstrate the construction of agents and the effectiveness of agent systems such as IMAGE, a number of examples are illustrated. Agent fundamentals are first illustrated with a "zeroth-order" agent example. A CATIA-based agent is shown next and demonstrates that agent technologies can be scaled to include large and highly complex proprietary resources. Finally, multi-proprietary resource systems are demonstrated with an aircraft component modeling system integrating CATIA and ORACLE and with a High Speed Civil Transport visualization system linking CATIA, FLOPS, and ASTROS.

Example 1: Zeroth-Order Agent

The zeroth-order agent, or tool, is characterized by two features: the model is not explicitly defined in the agent but rather is implicitly defined in the resource, and the input/output stream of the resource is wrapped. Since the model is implicitly defined, any information generated by the resource lacks accountability. For this reason, the differences between a zeroth-order agent and a tool are indistinguishable.

A points profile agent is used to illustrate the zeroth-order agent. As shown in Figure 5, the points profile agent returns a set of normalized coordinate pairs representing a 2D unit circle (extension to circles of arbitrary diameter is simple).

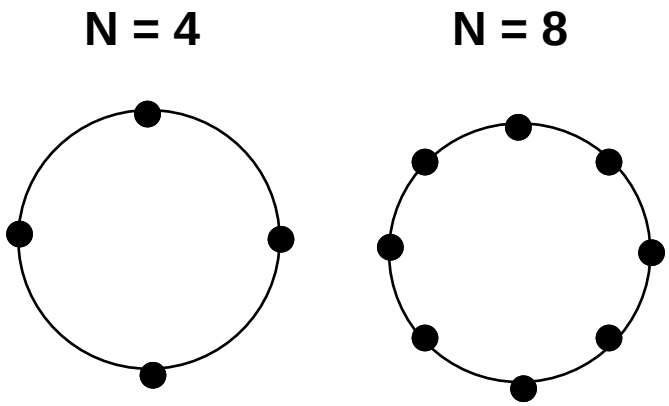


Figure 5. Points Profile

Model

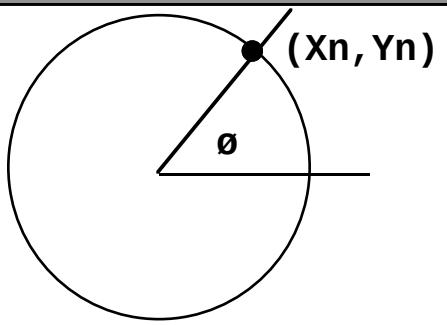
The points profile example uses a simple mathematical model, as shown in Figure 6. When queried for the circle description, the agent simply computes the normalized coordinate pairs based on the number of points needed and a uniform angular distribution.

Resource

In this example, the resource implements the mathematical model in the C language, as seen in Figure 7. The program reads the number of profile points from the command line and writes the normalized coordinate pairs to standard output. Other implementations could be done in C++, FORTRAN, BASIC, PASCAL, UNIX shells, etc. The specific choice of implementation is not important as long as the resource acquires data from the command line and returns values through standard output.

Since the program is based on an implicit model, the model cannot be changed without updating the resource. This model-resource dependency can become a critical issue as design fidelity increases and may cause computing resources to be locked into a single design phase. For example, circles

Mathematical Model



$$\phi = (n-1)\ 2*\pi/N$$

$$Xn = \cos(\phi)$$
$$Yn = \sin(\phi)$$

$$n = 1...N$$

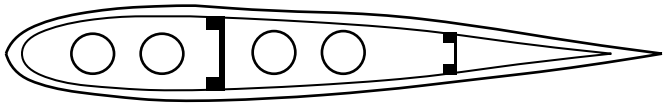
Figure 6. Points Profile - Model

C Program

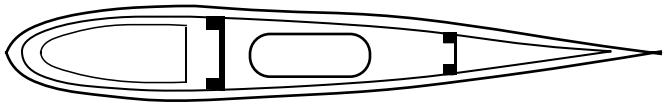
```
/*
 *This program determines the points
 *profile for a unit circle.  N is
 *given on the command line.
 */
#include <stdio.h>
#include <math.h>
#define PI=3.141593
main (int argc,char *argv[])
{
    int N=atoi(argv[1]);
    int n;
    for ( n = 1; n <= N; n++)
       printf(" { %f %f } ",
          cos( (n-1)*2*PI/N),
          sin( (n-1)*2*PI/N));
}
```

Figure 7. Points Profile - Resource

may be used to represent lightening and access holes in an outboard wing rib during preliminary design, as shown in Figure 8a. However, during detailed design, more complicated shapes may be used to represent those features, as shown in Figure 8b. The unit circle mathematical representation used for the zeroth-order agent in this example will not support the higher fidelity representations. A new resource would need to be developed based on an increased fidelity mathematical model.



a) Preliminary Design



b) Detailed Design

Figure 8. Wing Rib Representations

Wrap

The wrap for the points profile example is done in Tk/tcl, as shown in Figure 9. When a request is made, the wrap forwards the number of points through a Unix pipe to the command line of the resource. Then, the coordinate pairs are read from the standard output of the resource. Tk/tcl was used to wrap the resource since Tk/tcl provides agent communication and input/output stream utilities.

tcl Interpretive Script

```
#This procedure determines the points
#profile for a unit circle.  N is
#the number of points.
proc PointsProfile {N} {
    set C-Program [open "|circle $N"]
    set Profile [gets $C-Program]
    return $Profile
}
```

Figure 9. Points Profile - Wrap

As shown in Figure 10, the points profile example is executed in the following manner:

1) The user or another agent queries the points profile agent wrap with the value "`PointsProfile 4`".
2) The wrap gives the resource the value 4 on the resource command line.
3) The resource calculates the normalized coordinate pairs based on the implicit mathematical model.
4) The coordinate pairs are returned to the wrap through standard output.
5) The wrap returns the coordinate pairs to the querying user or agent.

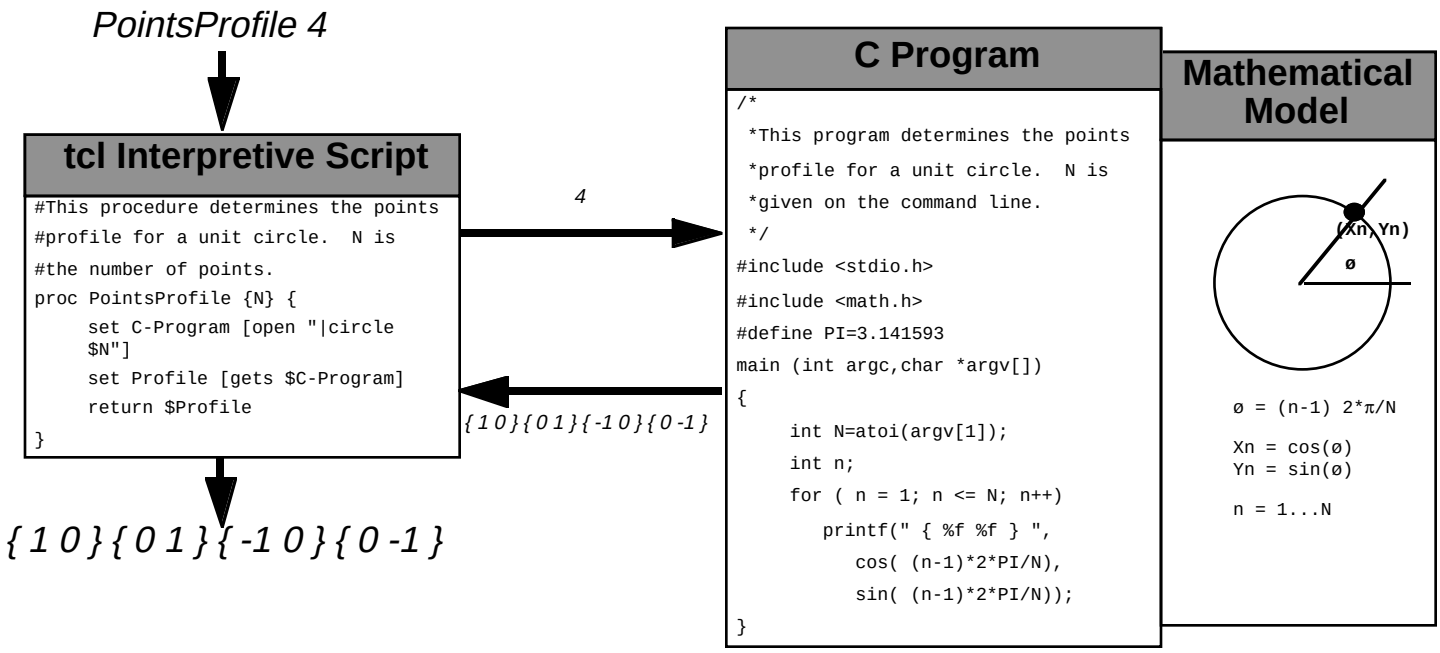


Figure 10. Points Profile - Execution

A number of important operating characteristics can be seen in this example:

1) The points profile wrap can be queried by any other agent or user on the network that understands the meaning of `PointsProfile`.
2) The resource can be changed without modifying the wrap and this can be done at run-time.
3) The model cannot be changed without modifying the resource.

Example 2: CATIA Agent

To demonstrate that IMAGE agent technology can be extended to include complex proprietary resources, CATIA was modeled and wrapped for use as a Passively Controlled Lifting Surface (PCLS) visualization agent, see Figure 11. The PCLS visualization agent was first used in LEGEND[12] to demonstrate its use in the design of a passively controlled wing. The CATIA agent was developed utilizing a model, the CATIA resources, and a wrap.

Model

The PCLS solid model is based on the CATIA concept of a volume transformation. Because of the complexity of the lifting surface definition, Boolean solids could not be used for the visualization. In a volume transformation, an object is represented by an approximate solid computed in CATIA directly from the exact volume. A volume is constructed from faces which in turn are defined by the edges that enclose simple or multiply connected regions of planar or complex surfaces. This relationship is shown in Figure 12. There are five solid entities that are used to visualize a PCLS: solid wing, solid wing with skin thickness, solid wingbox, solid wingbox with thickness, and foam filler. From the five solid entities, accountability is given to PCLS visualizations in CATIA. Not only is the PCLS modeled in CATIA, but information is made available to querying agents and tools regarding the visualization mechanism that is used. Such information would include the type of solid model representation, the solid's parametric definition, and the program used for visualization.

It should be noted that an almost unlimited number of different models could be defined for use with CATIA resources. This particular model allows for the visualization of the PCLS.

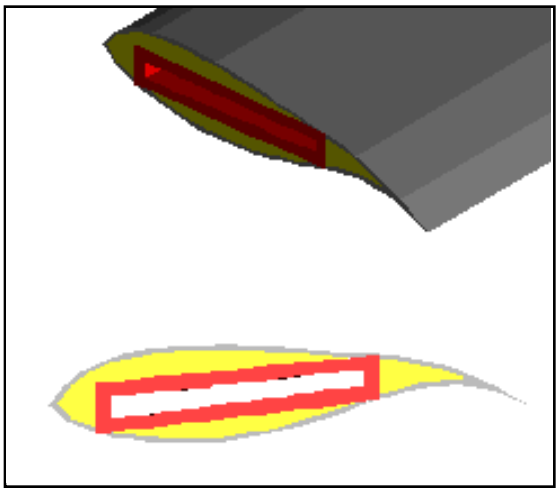


Figure 11. PCLS in CATIA

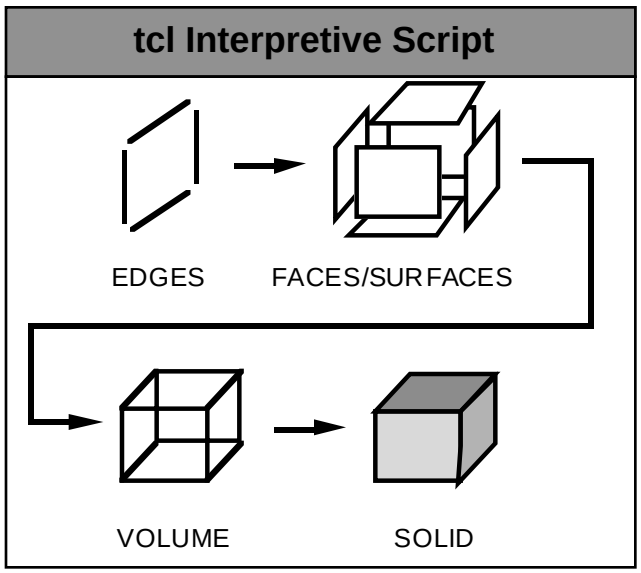


Figure 12. PCLS - Model

Resource

The CATIA GEOmetry (CATGEO) interface provides the resources required for the PCLS visualization, see Figure 13. The CATGEO interface functions allow the internal CATIA model database to be queried through user-defined, link-editable procedures. There are over 1500 CATGEO function calls available to the user. The PCLS visualization agent requires approximately 25 of them.

FORTRAN Utilities

CATIA GEOmetry (CATGEO) provides a mechanism for the user to interact with the model database. CATGEO consists of procedural calls made by linked, user-defined subroutines.

Example CATGEO call (draws a line between two points):

```
CALL GCWLN(MNUM,PT1,LIM1,PT2,LIM2,ELEM,IER,*1000)
```

Figure 13. PCLS - Resource

Wrap

The PCLS visualization agent is implemented with a Tk/tcl wrap of the model (description of the five solids entities) and the appropriate CATGEO resources in CATIA. To integrate the CATIA resources into Tk/tcl, a single function was written that dynamically accesses all of the CATGEO functions, as shown in Figure 14. The wrap is called the CATIA Interactive Interface (CII). Because the wrap accesses all 1500 of the available CATGEO functions, the wrap can readily be used to build other agents having different models.

C - FORTRAN Program
tcl Interpretive Script

A single function load, called the CATIA Interactive Interface (CII), has been developed that provides bi-directional CATGEO access to a Tk/tcl environment based on defined model utilities.

Figure 14. PCLS - Wrap

The Passively Controlled Lifting Surface visualization agent operates in the following manner, as shown in Figure 15:

1) The user or another agent queries the PCLS agent wrap with a parametric description of the PCLS.
2) The wrap gives the model the primitive PCLS solids: `SolidFromProfilesWithThickness`, `FoamFiller`, and `WingBox`.
3) The model translates the primitive ideas into a complex solid model definition.
4) The complex solid is generated in the CATIA geometry resource. At this point, the PCLS can be visualized in CATIA.
5) A unique identifier for the solid is generated in CATIA by the resource.
6) The identifier is returned to the model.
7) The model forwards the identifier to the wrap.
8) The wrap returns the identifier to the calling agent as an instantiation of the visualization in CATIA.

A number of enhancements to CATIA were discovered during the development of the PCLS visualization agent:

1) It is possible to create a "function load" without having to link the load into CATIA (with the dcg utility).
2) A CATIA session does not have to be terminated and re-started for load generation to take place. Traditional CATIA loads required that CATIA be stopped so that a link-edit procedure can be done.
3) Dependence on the CATIA model architecture becomes less stringent because more resources (for instance, a database) become available through the use of agents.

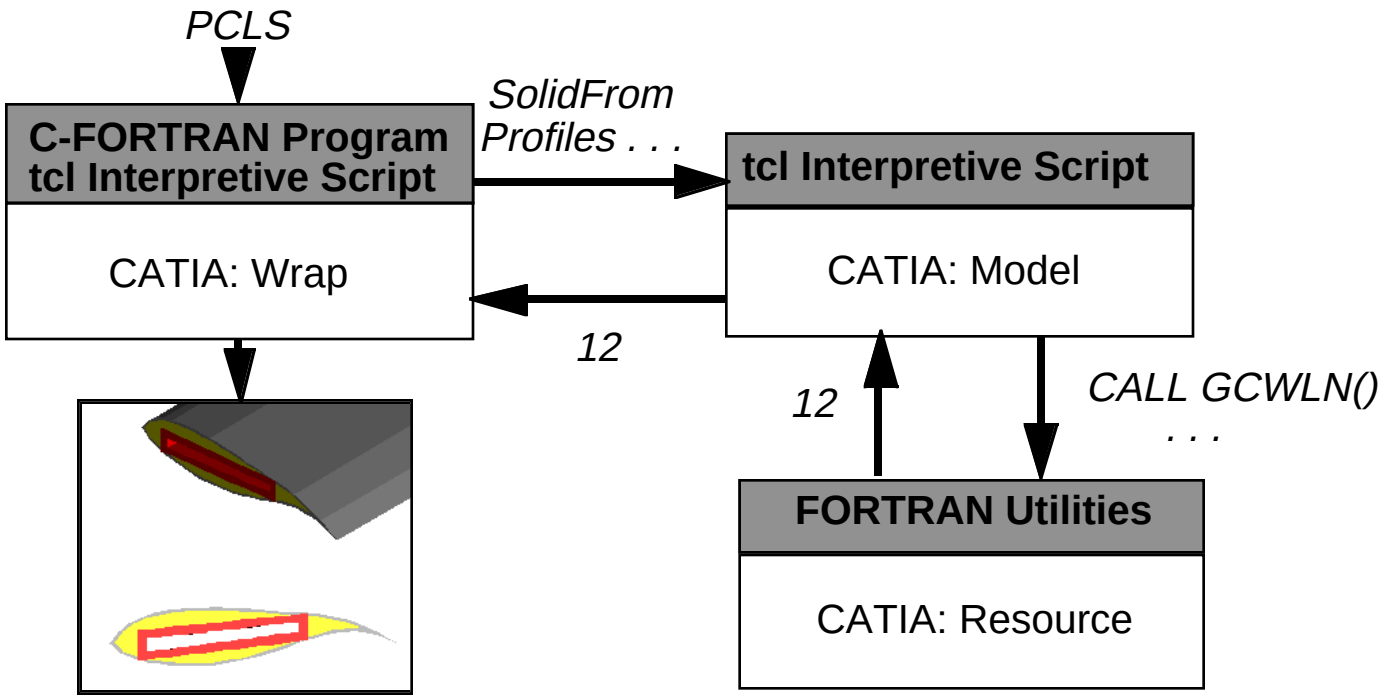


Figure 15. PCLS - Execution

Example 3: CATIA Agent / ORACLE Tool

An aircraft component visualization system, named AIRCRAFT, couples a CATIA agent with an ORACLE tool, as illustrated in Figure 17. The CATIA agent is similar to the PCLS visualization agent, except that wireframe representations are used in place of solids. The ORACLE tool is a wrap of the SQL resources in ORACLE without an associated model. The ORACLE tool is used to store parametric values associated with the various components of an aircraft. The system demonstrates the capability for integrating multiple proprietary resources in an agent-based environment.

Example 4: CATIA / FLOPS / ASTROS Visualization System

The IMAGE agent technology was used in another ASDL project to create a High Speed Civil Transport (HSCT) visualization system.[16] The system utilizes CATIA as a visualization agent, FLOPS to create HSCT geometry, and ASTROS to develop a wing structure finite element representation. A sample HSCT configuration is shown in Figure 16. A standard solid model of the HSCT and a wireframe representation of the wing structure FEM can be generated in approximately five minutes using this system. The HSCT visualization system demonstrates the capability for integrating both proprietary and nonproprietary resources in an agent-based environment.

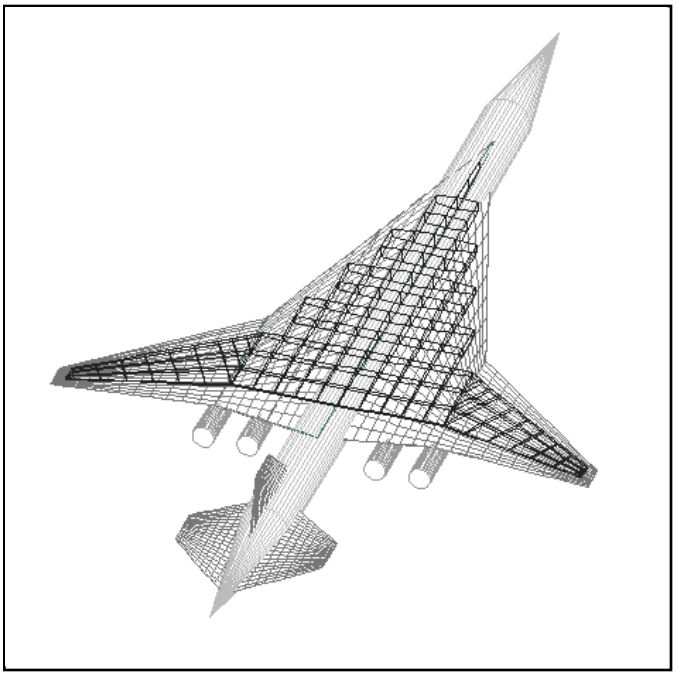


Figure 16. HSCT Visualization

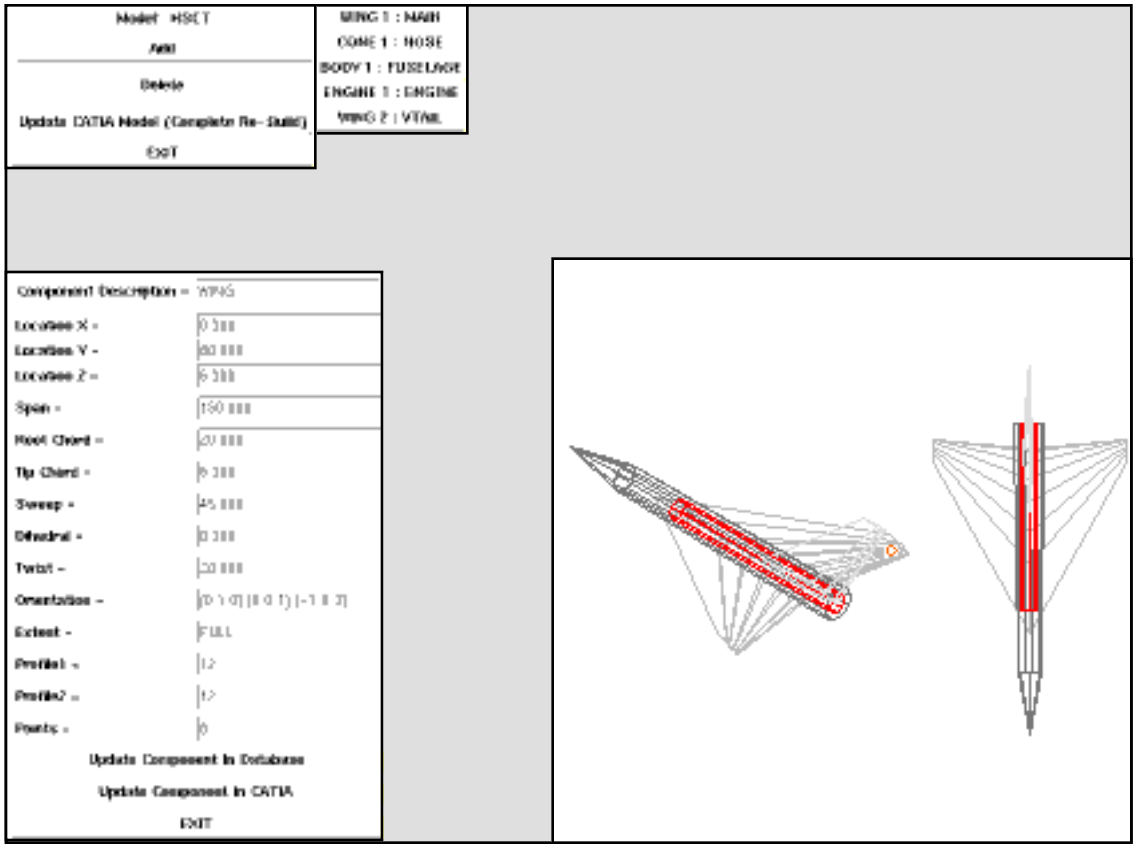


Figure 17. CATIA Agent / ORACLE Tool

Conclusions

The agent is one of the key technologies required for the implementation of integrated computing environments for Concurrent Engineering. The agent as defined in IMAGE has three basic components: the resource, the model, and the wrap. IMAGE agents formalize the role of proprietary resources in the implementation of computational design environments. Finally, IMAGE uses the agent to generate accountable information (information with context) that can be used throughout a product's life-cycle.

A PCLS visualization agent illustrates the successful development of an agent that is based on a proprietary product (CATIA). An aircraft component visualization system and a High Speed Civil Transport visualization system show that the coupling of proprietary and nonproprietary agents is feasible. These examples illustrate the important role that agents can and will play in the design of open engineering systems.

References

[1] Schrage, D.P. and J.E. Rogan, "The Impact of Concurrent Engineering in Aerospace Systems Design," course notes, 1991.

[2] Townsend, J.C. and R.P. Weston, "An Overview of the Framework for Interdisciplinary Design Optimization (FIDO) Project," NASA TM 109058.

[3] Jones, K.H., D.P. Randall and C.K. Cronin, "Information Management for a Large Multidisciplinary Project," AIAA-92-4720.

[4] A.R. Dovi, et al, "Multidisciplinary Design Integration System For a Supersonic Transport Aircraft," AIAA-92-4841.

[5] Cutkosky, M.R. et al, "PACT: An Experiment in Integrating Concurrent Engineering Systems," IEEE, Computer, January 1993.

[6] Huyn, P.N., et al, "Automated Concurrent Engineering in Designworld," IEEE Computer, January 1993.

[7] Hughes, David, "Generic Command Center Speeds Systems Design," Aviation Week & Space Technology, March 1993.

[8] "ACSYNT Overview and Installation Manual," ACSYNT Institute, May 1992.

[9] McCullers, L.A., "FLight OPtimization System, User's Guide," Version 5.41, NASA Langley Research Center, December 1993.

[10] Kroo, I. and M. Takai, "A Quasi-Procedural, Knowledge-Based System for Aircraft Design," AIAA-88-4428.

[11] Gage, P. and I. Kroo, "Development of the Quasi-Procedural Method for Use in Aircraft Configuration Optimization," AIAA-92-4693.

[12] Stephens, Eric "LEGEND: Laboratory Environment for the Generation, Evaluation and Navigation of Design," Doctoral Dissertation, School of Aerospace Engineering, Georgia Institute of Technology, September 1993.

[13] Hale, Mark A. and J. I. Craig, "Preliminary Development of Agent Technologies for a Design Integration Framework," School of Aerospace Engineering, Georgia Institute of Technology, USRA HSCT Workshop, December 1993.

[14] Ousterholt, John K., An Introduction to Tcl and Tk. Addison-Wesly Publishing Company, Inc: 1993.

[15] Bhatia, Kumar G. and J. Wertheker, "Aeroelastic Challenges for a High Speed Civil Transport," AIAA-93-1478.

[16] Marx, William J., Schrage, D. P. and D. N. Mavris, "Integrated Product Development for the Wing Structural Design of the High Speed Civil Transport," School of Aerospace Engineering, Georgia Institute of Technology, USRA HSCT Workshop, December 1993.