Architecture of Tools and Processes for Affordable Vehicles

J. Stodola*
Faculty of Military Technology, University of Defense, Brno, Czech Republic
** Corresponding author: jiri.stodola@unob.cz,*

P. Stodola
Faculty of Economics and Management, University of Defense, Brno, Czech Republic

ABSTRACT: The focus of this paper is on the description of the recommended functional requirements for the software architecture that enables the integration of tools and processes for large scale affordable vehicles and propulsion systems. These include: integration, processes, tools, affordability, repeatability, sustainability, integrity, etc. Prior to the discussion of the recommended functional requirements a brief description is given on the two types of integration environments (Monolithic Environments and Best Class Environments) along with a categorization of the different type of tools considered to be integrated within the environment. The four categories of tools addressed are Groupware, Project Management, Product Data Management or Product Lifecycle, and Engineering tools.

KEY WORDS: Software architecture, product and process representation, engineering tools, data, application

1 INTRODUCTION

When new processes are proposed, new tools discovered, mature tools exist, additional teams engaged, and distribution of work dispersed all are constrained by affordability, a description of what these new concepts are must be accompanied by a discussion of how they can be used. The “how” is the subject of this paper. What follows is a discussion of the salient features of a computing architecture that will support the Integration of Tools and Processes for Affordable Vehicles based upon the knowledge discovered in examining the unique and common needs of how product development is accomplished for vehicles and their respective propulsion systems. Once the recommended functional requirements are presented an example of how an implementation of the environment would be used to perform a simple distributed engineering analysis of an aircraft component is presented. In order for an organization to execute process du jour, it utilizes various software tools to carry out individual tasks within the process. For clarity, a software tool in this article is defined as “a software application used to perform or facilitate the execution of specified task in a process”. Here, software tools are categorized into four classes: Groupware tools (video conferencing, online meetings, teleconferencing etc.), Project Management tools (project schedules, distributing information, resource planning, team organization, process work flows etc.), Product Data management or Product Lifecycle tools (data, drawings, reports, models etc.), and Engineering tools (Computed Aided design – CAD, Computer Aided Manufacture – CAM, Computer Aided Engineering – CAE) (Sehra at all., 2006).

2 ENGINEERING TOOLS INTEGRATION

Over the past twenty five years the two primary paradigms that have emerged to perform Multidisciplinary Analysis and Design (MAD) are Monolithic Approach (MA) and Best in Class Approach (BCA) systems.

The monolithic MAD environment systems consist of a single application/tool that contains all necessary functionality to perform the desired analysis or design. If well designed, these monolithic systems contain a single database with a well-defined interface in which each functional module communicates all information through the centralized database. Each module appears independently in the system but must obtain all of its inputs and write all output that is needed by other modules to the centralized database. These systems also usually have standard Application Programming Interfaces (APIs) that enable communication with the monolithic system and at times the ability, with some effort, to add additional functional modules to the environment. In addition, these systems often have a High Order Language (HOL) that is used to combine the modules to solve a specified problem. A standard set of sequences written in this HOL are often available in these environments or a single sequence that can solve many variations of a predefined set of problems is used. The monolithic applications give users access to the HOL so they combine the available modules to customize the system to solve problems not accounted for in the standard set of sequences. These monolithic systems are typically easy to administer and are fairly robust with good error trapping and handling. However, if not well designed they can be rigid and difficult to customize and may require access to source code of modules that are being added to the system. Also, the monolithic systems do not deal well with highly distributed organizations and data. But most importantly they are usually built around a single discipline expertise, such as mechanical analysis, CAD, or KBE, and have marginal capabilities in other disciplines, such as controls, or computational fluid dynamics.

The BCA system typically uses a scripting language, such as Perl (Wall et al., 2001) or Tcl (Outsterhout, 1994) to “glue” together several independent application tools that provide the “best” functionality for a given discipline to define a process and solve a selected problem. Such existing approaches can be found in iSight (http://www.engineous.com), Model center (http://www.phoenix-int.com), MDICE (http://www.cfdrc.com), and Visual Doc (http://www.vrand.com). This approach “wraps” each application with the scripting language and defines simple input and output that a given application requires or generates. A major benefit of this approach is that it gives the end user access to the “best” technology available in a given domain and thus supports the “plug and play” Paradigm to a certain extent. This approach is much more portable to engineering domain experts since they can include their “best of class” application for a specific problem being solved. In theory this approach appears quite attractive, but in practice many problems arise with this approach. Each application has different data structures and formats. This can lead to difficult and inefficient data transfer between applications. Scripts for large-scale problems tend to become unruly/problem specific, and hence fragile, difficult to maintain, and not reusable. Also, more importantly, there is no well-defined manner to trap errors that occur during the process. This may not be an issue when combining only a few applications but when the number begins to approach the 10s or 100s the ability to determine if, when, where, and why a failure occurred becomes the critical element in the success of the BCA.

3 FUNCTIONAL NEEDS AND ARCHITECTURE

To best satisfy the requirements of product development in today's business environment a distributed BCA MAD is desired with the following qualities:

1. Product representation – a way to represent the product along with the design intent or rules;
2. Seamless access to varying fidelity best in class tools to evaluate or modify the design;
3. Process Representation with Secure Communication between all tools, data, and vested parties involved in the product development process;
4. Modularity that enables high level of reuse when moving from one application to the other.

Such a system will allow specialized communities to exist (Centers of Excellence) but will require them to publish and maintain with defined interfaces to their domain so that communications between the different domains can take place at a level that is required to evaluate the impact of one domain on another. If the interfaces are well defined and "published" on network they can be accessed anywhere, anytime allowing all participants access to the most recent product information and technology. Even though the specified functional requirements would allow communication and integration between the four classes of tools, the primary focus is on the integration of project Management tools, Product Data Management or Product Lifecycle tools, and Engineering tools.

To satisfy the integration requirements arising in the emerging product development processes in today's business environment a functional architecture is proposed. The integration of the tools will result in tangible benefits to a company and organization. It should also be evident that an ad-hoc approach to performing the integration of this plethora of widely distributed tools would not be adequate. Thus, a formal process should be adopted to developing an integration environment. This environment should enable the integration and communication across the various tools and data that are encountered during three product development processes. The functional architecture for an integration environment is henceforth presented. Here functional architecture is defined as a description of all functional activities to be performed to achieve the desired mission, the system elements needed to perform the functions, and the designation of performance levels of those system elements. Architecture also includes information on the technologies, interfaces, and location of functions and is considered an evolving description of an approach to achieving a desired mission. Throughout the remainder of the document the term Architecture and Functional Architecture will be used interchangeably. The Architecture shall be capable of supporting multiple analysis techniques and information standards for any discipline. Realizing the current investment in tools such as CAD/CAM tools and physics-based solvers (e.g., finite element modeling packages, computational fluid dynamics packages, computational electromagnetic solvers, etc.). The Architecture should not require a priori the use of any geometric representation, analysis technique or information standard. Additionally, different modules within the architecture should be easily replaceable or maintainable.

Product Representation

The current state-of-the-art in product representation is a "single parametric associative model, referred to as a Master Model. The Master Model concept traditionally contained only geometric information, but has now been extended to contain any critical information that may be needed throughout the life of a product. The Master Model is a single logical representation of the product that may be distributed geographically of between several different databases or applications, the point being that there is a single representation of a product without any duplication of information. All users begin from and update a single

representation of the product to ensure consistency. A CAD system (UniGraphics, ProE, Catia, etc.) along with a PDM (e-Matrix, Windchill, etc.) system are typically combined to create a master Model. Many companies are also coupling the Master Model with KBE systems resulting in what is called an "Intelligent Master Model" (IMM) or Smart Product model (SPM). This allows design intent and rules to be maintained with the model along with the model representation itself. Typical KBE system employed are AML, Inlet and UG Knowledge Fusion. A few features that are desirable for the IMM are:

1. Ability to quickly generate a representation of a product
2. Support parametric and topological changes
3. Maintain and document the design intent
4. Demand Driven Calculation – the product representation should perform only the calculation that are required to determine the result of a desired analysis or functional evaluation
5. Capture the knowledge and design intent of the product
6. Ability to quickly generate the domain specific analysis and design models when parametric or topological changes are made. This feature is key to supporting high fidelity analysis and numerical zooming early in the design process
7. Support Dependency Tracking - the product representation should automatically track the dependencies between various objects and properties within the model

Seamless Access to Engineering Tools and Data

Multidisciplinary. It is no longer acceptable to perform design analysis in the technical disciplines separately. Any "optimization" at the component or subsystem level will lead to a sub-optimum system. There are various approaches to this problem. It is important to remember that multidisciplinary simulation couples physical processes and the design of an interface has therefore to be based on physical understanding, and not only on implementation issues. There are many tools which assemble simulation programs used for a workflow together in an integrated environment, but even those environments need, as a core element, interfaces for the physical interaction between, the simulation programs and the models used.

Multi-level/Fidelity Zooming. The applied vehicle technology architecture, in general, should allow consistent analyses to be performed at all levels within the system. Similar multidisciplinary applications, so as to the importance of interfaces to Multi-level analysis. The term Multi-level is also known as numerical zooming or higher fidelity forward. The fundamental goal is to bring higher fidelity information up to a system view of the model where the application can "see" the effects of all the constituents interacting together. The definition of higher fidelity information includes 1-dimensional through 3-dimensional Computational Fluid Dynamics (CFD), experimental data and historical rules of development.

With a BCA approach the need to "glue" together different types of applications becomes a critical aspect of the environment. Earlier programs such as iSight and Model Center use scripting languages as this glue. This requires wrapping an application in a scripting language such as Perl, Tel, or Python (Claus at all., 1991) to expose the application or data that one wishes to bring into the environment. These applications expose parameters in this environment by parsing the applications input and output text files. With this technique the application is exposed as a single entity based on a set of input and output parameters. This technique works reasonable well when less that half a dozen applications are involved in a given process and when they reside on a single platform. Also, the size of the inputs/outputs from the application is restricted to small text files (a few megabytes). This technique tends to break down when the number of applications grows beyond 10 – 25 and the inputs/outputs of a given application are non-text files and are of a large size (100s of megabytes to

gigabytes). A great deal of knowledge has to be included in the wrapper to interact with the application on the lowest possible level that the application will support. The wrapper that exposes the object (application, hardware, etc.) will be referred to as a service provider. AQ service provider exposes discrete functionality of an object by a set of attributes to the environment; the functionality exposed is referred to as a service. The act of inserting the service into an environment is referred to as publishing. Two types of service providers are envisioned: Method Service Providers and Context Service Providers. A Method Service Provider publishes one or more methods associated with a given application where a Context Service Provider publishes one or more pieces of data associated with a product model (IMM or SPM). The following are required key features creating service providers:

1. Wrap with an object oriented language
2. Communicate intermediate results back to the client
3. Develop standard interfaces for application domains
4. Trap errors encountered within the application and pass meaningful information back to the client
5. Enable the application in a server mode to allow finer granular interaction between the wrapper and the application
6. Expose the application at a level of granularity that supports the most reusability
7. Once the service providers have been created it must then be published with attributes so the environment knows where it is and what it can do. Some examples in the world are: Sun's Jini Technology (Lutz, 2001), web Services Sun ONE, Microsoft´s.Net (http://sun.com), Globus, Integrated Virtual Product Development (iVIP) etc.

Classification of Interfaces

Simulation tools have usually been designed as stand-alone applications in a prescribed work flow. Any two tools rarely use the same native model description or data structure. Interfaces provide a means of communication between two or more coupled applications. Interfaces can be categorized in terms of work flow aspect. Here, a distinction can be made between uni-directional and bi-directional interfaces (Sehra et al., 2006), Fig. 1. A uni-directional interface is needed if one program is used as a pre-processor for a second program. Typical examples are grid generators for fine element analyses. Bi-directional interfaces handle the flow of information between two running simulations. Typical examples are co-simulation interfaces.

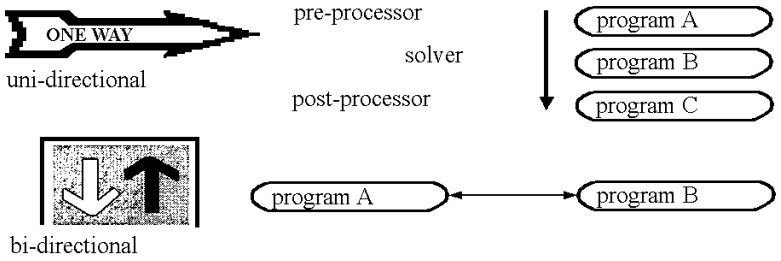

Figure 1: Uni-directional and Bi-directional Interfaces

The concepts of a connector object within the vehicle architecture is centered on providing a suite of objects that allows two components to connect together whether they are of the same/different discipline, same/different fidelity or same/different computing platform and have the object to handle all the intricacies of that connection. A way to illustrate this concept can be served by describing how this connector object would work in assembling a vehicle engine numerically. Although engine components are the basic building blocks of propulsion simulations, connector objects are the means by which components

communicate and provide the technology support for zooming and the required expansion/contraction/averaging of data. Connectors are represented in the architecture as objects. Additionally, the connector object provides a means to introduce distributed processing into the subject engine simulations. Connectors not only facilitate "what" data moves but also "how" the date moves around within the simulation. When the nature of the simulation determines that a connector doesn't add anything, or isn't needed, the effect of the connector is a data pass-through operation (Sehra et al., 2006). Pictorially these concepts can be illustrated in Fig. 2.

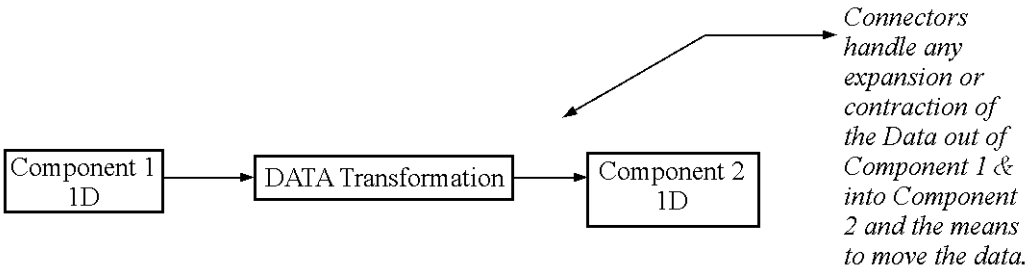

Figure 2: Effect of the connector

The connections between the engine components which are the source of the data and the methods are implemented with C++ classes called "connectors". Every such connection between a source component and a destination component uses two different connectors, a source connector and a destination connector. Thus a component has one input connector for each input and a many output connectors as there are other components which use this component's output as input data. The connectors, upon creation, determine whether the mating connector is local or remote. A local connector is the same process and hence the two connectors may directly call methods of one another in order to request and transfer data. A remote connector is a different computer and hence requires communication of messages in order to exchange data. COM (http://www.zgdv.de), COBRAQ (Pattison, 1998), Java RMI (Fintan, 2002), and PVM (Niemayer & Knudsen, 2002) are representative libraries that could be used to transport the data.

When an engine component is ready to execute, the connector transfers the data from the message to the engine component object. The very significant advantage of this approach is that there are no software concurrency problems because of multiple engine components updating concurrently in different machines. This allows all remote communication from other engine components in other computers to actually become communication calls to the single component which then distributes the data request or data reply information to the individual C++ objects representing the engine components, avoiding software concurrency issues.

4 PROCESS REPRESENTATION AND COMMUNICATION OF DISTRIBUTED DATA, APPLICATIONS, AND VESTED PARTICIPANTS

The subject applied vehicle technology architecture is comprised of the hardware and software computer systems needed to perform all the required analyses that are involved in vehicle design. This architecture must operate within a highly distributed, heterogeneous modeling and computing environment (Sehra et al., 2006). The envisioned architecture is an object oriented peer-to-peer (P2P) service based open architecture that must support the specified layers, shown in Fig. 3. Starting from the bottom of the figure, Layer 1 consists of the computer hardware in the system. Next, Layer 2 functionality addresses the need for hardware resource management, such as load balancing on the hardware. Layer 3 represents

the abstraction or the separation of the services/object from the hardware and "exposes/publishes" these services on the network as network services. In addition to the publication this layer must support the "discovery" of services by clients and the communication with or between the services. Finally, the top layer, Layer 4, represents an object model that enables the combining of the services to represent a process or multi-service transaction. This layer must also support the execution of this object representation of the process and the passing of information from service to service.

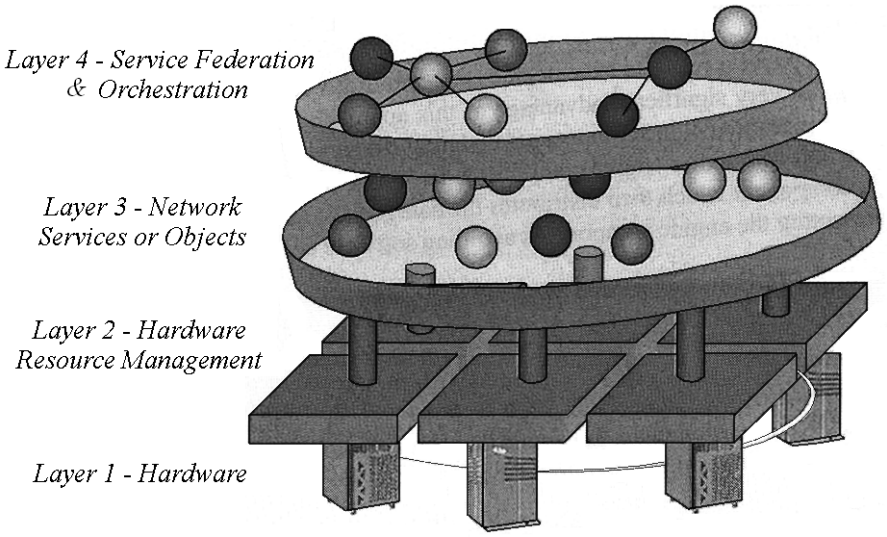

Figure 3: Recommended architectural functional layers

Level 1 Hardware Computing Environment

The architecture should run natively on multiple hardware platforms and operating systems. At a minimum it should operate under Unix, Linux, and the Microsoft Windows systems.

Level 2 Hardware Load Balancing

The architecture enables the use of available load balancing applications, such as Platform Computing´s Load Sharing Facility, IBM´sLoad Leveler, and NASA Ames´ Portable Bath System (http://www.csm.ornl.gov) etc.

Level 3 Network Services – Service Oriented Architecture (SOA)

SOA have three basic components (Service Requester, Service Registry, and Service Provider) and three basic functions (Publish, Find, and Bind).

Level 4 Service Federation and Execution Coordination – An Object Model for Service Orchestration

The P2P service-oriented architecture proposed targets multiparty service transactions. A collection of all registered service providers is referred to as a service grid. This is essentially level 3 in Fig. 3. A nested transaction is composed of a federation of providers that came together for completing a transaction. Hence, the primary function of Level 4 is to combine and orchestrate communication between the services in Level 3. The service providers do not have mutual associations prior to the transaction. They come together for a specific transaction. A standard object model representing these three components in a nested transaction is critical for the applied vehicle technology architecture. The object that represents the process, action, and data can be created by any end-user, application, or service and act as a service requestor and submit the object to the environment for execution. The object that represents the process must support different types of execution strategies for the process such as sequential, parallel, looping and conditionals. It must also account for the mapping/relationship of data between steps or services in the process. Finally, it is desirable that the process object support recursion.

5 MODULARITY – MAXIMALIZATION OF REUSE

The BCA creates the need to link together disparate applications with different needs and different data structures. This tends to create the development of applications that work on a very specific problem or narrow range of problems. One of the primary goals of an environment should be to develop a system that maximizes reuse when moving from one application, project, or product development to another. This will at least attempt to minimize the resources needed to get the next design applications to a point where it is useful in a timely fashion. An environment that promotes maximum reuse will have the following features:

1. Use an object oriented approach for the environment (Java, C++, etc.)
2. Create Common Object Models for specific domains (CAD, CAE, Optimization, etc.) and pass these objects around to the services when possible
3. Create standard interfaces for services
4. Create generic wrappers for applications
5. Separate product data from applications and their wrapper
6. Have at least one “champion” of the environment

Although all of these sound logical they are by no means trivial elements when undertaking a new development effort. A major portion of the resources used when going over to a new application is in the development of the product models (Stodola, 2007). That is the Master Model, Intelligent Master Model or Smart Product Model. Also, to modify existing components and to bring in additional applications at any given time takes a considerable amount of effort to do properly if you desire reuse. Finally, item 6 is essential but often easily over-looked. If an organization does not commit the resources to have an individual who is a “champion” of the environment the effectiveness of the use of the environment will be greatly hampered.

REFERENCES

Sehra, A., Reed, J., Hoenlinger, H., Luber, W., Stodola, J., Follen, G., Hoeninger, M., et al. 2006. *Integration of Tools and Processes for Affordable Weapons.* Final Report of the NATO Research Group AVT 093, Paris (324 p).
Wall, L., Christian, T., Orwant, J. *Programing Perl*, 2001. O´Reilly.
Outsterhout, J. K., 1994. Tcl and the Tk Toolkit. Addison – Wesley.
iSight, http://www.engineous.com/index.htm
Model Center, http://www.phoenix-int.com/products/ModelCenter.html
MDICE, http://www.cfdrc.com/bizareas/aerospace/aeromechanics/aeroelesticity/html
VisualDoc, http://www.vrand.com/
Claus, R. W.-Evans, A. L. - Lytle, J. K. – Nicholas ,L. D., *Numerical Propulsion Systems Simulation.*, 1991. Computing Systems in Engineering, Vol.2, No. 4, pp. 357 – 364.
Lutz, M., Programing Python” Object Oriented Scripting. 2001. O´Reilly.
http://www.sun.com/software/sunone/
iVIP, http://www.zgdv.de/zgdv/refprojects/ivip/index_html_en.
Pattison, T., *Programming Distributed Applications with COM and Microsoft Visual Basic 6.0*. 1998. Microsoft Press.
Fintan, B., *Pure COBRA*. 2002, Sams.
Niemayer, P.-Knudsen, J., *Learning Java*. 2002. O´Reily.
PVM, http://www.csm.ornl.gov/pvm/pvm_home_html
Stodola, J., *Modeling and Simulation in the Virtual Design of Armored Vehicles.* 2007. IVMT´07. ISBN 978-80-723-2, pp 31 – 38.