A Prototyping Effort for the Integrated Spacecraft Analysis System

Raymond Wong, Yu-Wen Tung, and Pierre Maldague
Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA, 91109

Abstract—Computer modeling and simulation has recently become an essential technique for predicting and validating spacecraft performance. However, most computer models only examine spacecraft subsystems, and the independent nature of the models creates integration problems, which lowers the possibilities of simulating a spacecraft as an integrated unit despite a desire for this type of analysis. A new project called Integrated Spacecraft Analysis was proposed to serve as a framework for an integrated simulation environment. The project is still in its infancy, but a software prototype would help future developers assess design issues. The prototype explores a service oriented design paradigm that theoretically allows programs written in different languages to communicate with one another. It includes creating a uniform interface to the SPICE libraries such that different in-house tools like APGEN or SEQGEN can exchange information with it without much change. Service orientation may result in a slower system as compared to a single application, and more research needs to be done on the different available technologies, but a service oriented approach could increase long term maintainability and extensibility.

1 Introduction

With advances in computing technology, modeling and simulations has played a vital role in the analysis of spacecraft systems for over a decade, especially as an aid for mission planning and execution.[1] Integrated Spacecraft Analysis (ISCA) is a proposed project within the Jet Propulsion Laboratory (JPL) that signifies a desire to improve spacecraft modeling and simulation techniques. This paper discusses the motivation for ISCA, a software development activity to support software design, and some technologies that may prove useful in achieving the project's goals.

2 ISCA Background

Mission planners use computer simulations to predict spacecraft behavior, which allows them to make decisions about resource allocation and usage that help achieve goals in science and planetary studies. For each project, several teams of engineers focus on a particular area of the spacecraft, and they have the responsibility to develop high fidelity models of their respective subsystems. Each model was typically implemented with a unique configuration, and they worked well for their purposes.

However, the ad-hoc manner in which some models were created tend to prevent the same model from being used again on a similar spacecraft. The Advanced Multi-Mission Operations System (AMMOS) was created to alleviate some portability issues by producing adaptable, multi-mission tools.[2]

Engineers eventually leveraged the multi-mission power (MMPAT) and telecommunications model (MMTAT) to create an integrated simulation environment for the Deep Impact (DI) mission, at least for planning activities and creating command sequences. They created interfaces to MMPAT and MMTAT such that another multi-mission tool, the Activity Plan Generator (APGEN), can communicate with those models.[3]

Previously, data from one model was written to some repository for another team to use as input for their model. For example, attitude information from the Attitude Control System (ACS) model in APGEN was written to a set of files, properly formatted, and transfered to the Thermal Subsystem and Telecom Subsystem teams for analysis. If other teams found problems with the current plan, they report their findings back to the planners for correction. Since APGEN could interface with its own telecom model for DI, planners could perform a simulation with MMTAT and adjust their plans before handing it off to the Telecom team, a much more cost-effective process than the previous approach. The downsides of the DI initiative were increased ”organizational complexity” and non-standard, non-multi-mission interfaces to the models.[3]

ISCA seeks to emulate the same capabilities from DI for future missions by providing a standardized simulation framework for different teams to ”plugin” their model and perform a simulation. Since models are ”plugged” into the framework, a team may use models from previous phases of the same mission, reducing the team’s need to reproduce a similar model from scratch nor adjust their current simulation environment. Therefore the major requirements of ISCA call for an extensible software framework to accommodate a variety of existing spacecraft models.

3 Service Orientation

One design path for ISCA leads to a paradigm known as service orientation.[4] A paradigm comprises of a set of rules or principles that help attain certain characteristics in the software solution, and most paradigms try to apply an idea called Separation of Concerns.[4][5]

Separation of Concerns ”deals with creating distance between dissimilar aspects of your code.”[6] In other words, Separation of Concerns suggest a software developer should solve a problem by breaking the problem down into smaller distinct components or ”concerns,” then tackling the problem by dealing with each ”concern.” One of the main goals for applying this concept is increased maintainability.[6]

3.1 Design Principles of Service Orientation

Service-orientation can fulfill the goals of Separation of Concerns through its set of design principles: standardized service contracts, service loose coupling, service abstraction, service reusability, service autonomy, service statelessness, service discoverability, and service composability. Each of these principles are applied to the design of software services, programs that perform a set of tasks when requested.[5]

3.1.1 Standardized Service Contracts

Standardized service contracts require services within the same group to conform to a set of standard communication protocols or schemes.[5] Standardization promotes consistency and reliability across different environments because it allows the consumer of services to understand the service’s capabilities, the data it requires, and the communication mechanism needed to transfer the data.[5] An analogy is that customers expect delivery persons to bring their package to the doorstep of the delivery address. If one delivery person brings the package to the door while another delivery person leaves the package on the sidewalk outside the home, then the customer has no expectation for where the next person may leave the package. However, a standardized practice for all delivery services, whether the package is delivered at the doorstep or the sidewalk, but preferably at the doorstep, gives the customer consistency.

3.1.2 Loose Coupling

Service loose coupling refers to the desire for service implementations to have small dependencies on the communication implementation and to its customers.[5] This allows the logic behind a service to change without forcing major changes to occur in other services or the service contracts.

3.1.3 Abstraction

Service abstraction contributes to loose coupling because ”it emphasizes the need to hide as much of the underlying details of the service as possible.”[5] The ability to hide details allows the service logic to change while still providing the capabilities it promised using the same communication standard expected from it.

3.1.4 Reusability

Service reusability stresses the importance for services to be independent of the larger application’s goals.[5] A service may perform one or a series of minute tasks, and a collection of services, used in a certain manner, achieve a larger purpose. However, a service designed to be reusable can perform the same capabilities and contribute to the achievement of another purpose, eliminating the need to produce a new service by way of an existing service.

3.1.5 Autonomy

Service autonomy suggests services should have control over their environment.[5] Restated, services should have the freedom to use resources within the bounds of the contract to complete their job description.[7]

3.1.6 Statelessness

Service statelessness says a service does not need to manage and maintain the data it used in its past invocations. Instead, the service only needs to maintain the necessary states and data to help it process the current request. The name ”Statelessness,” according to solution architect Michael Poulin, is ”misleading” because the service does have a state.[8]

3.1.7 Discoverability

Service discoverability refers to information that allows potential consumers to figure out how to use the service. This may refer to documents that describe the service contracts and capabilities, such as a diagram or text file, and the documents should, ideally, contain no ambiguity such that only one interpretation of each service exist.[9]

3.1.8 Composability

Service composability directly supports Separation of Concerns because the principle advocates that ”small problems collectively represent the big problem,” which means solutions to each of the smaller problems aggregate into a larger solution and solves the larger problem.[5]

These principles add up to an ISCA framework that would convert standalone models and tools from each team into services. Several categories of services, each with their own set of contracts and capabilities, can exist. This allows, for example, one ACS model to be exchanged with another ACS model without forcing the navigational model to readjust itself. Additional work for each model is required though. Engineers need to develop another layer on top of each model that is contract compliant, transforming each model into a service. Figure 1 illustrates the previous statement.

3.2 Remote Procedure Calls

A variety of technologies are available for developers to use and build their service oriented applications, provided they follow the design principles outlined above. At the moment, this paper will focus on one type of communication technique known as remote procedure calls (RPC).

In computer science, a procedure is a set of instructions for performing some calculation.[10] In modern computer systems, one procedure, the caller, may use another procedure, the callee, by transferring control of the central processing unit (CPU) so that the callee may run its instructions on the CPU.[11] This transfer mechanism occurs on a single computer.[11] Remote procedure calls are an extension of the previously mentioned mechanism except the called procedure can exist on a different physical machine, and the transfer of control occurs over a network.[11] When the called procedure finishes its computation, it sends the results back to the caller via the network.[11]

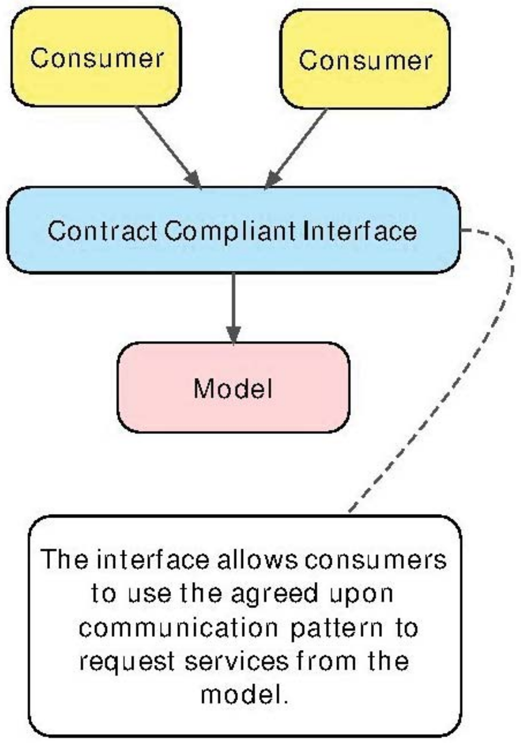

Consumer
Consumer
Contract Compliant Interface
Model
The interface allows consumers to use the agreed upon communication pattern to request services from the model.

Fig. 1. Extra development to convert a model into service.

One primary goal of RPC is ”the coexistence of independent execution environments.” In addition, it aims to let programs call procedures over the network as simple as making local procedure calls, which means the RPC protocol tries to hide as much of the network details as possible.[11]

Both of these features make them an attractive technique for service orientation because one model operating under one simulation environment may call procedures from another model acting under an entirely different environment, i.e. different modeling languages. In addition, it creates a loosely coupled system. Different models can have the same procedure names but have different methods of completing the same tasks, which means models can easily be replaced.

A few existing RPC implementations are XML-RPC, SOAP, CORBA, and DCE.

3.2.1 XML-RPC

XML-RPC is a specification and ”set of implementations” for making ”procedure calls over the Internet.” It uses ”HTTP as the transport and XML as the encoding.” The specifications define how data types should be represented in XML, and act as the driving requirements for software developers to create encoding, decoding, and network libraries for C/C++, Java, Python, etc. XML is simple to use, and parsing packages are widely available for almost every language, which essentially eliminates the communication barriers between different languages.[12][13]

Some claim the encoding of data to XML works well for more primitive data types, but it forces procedures to only handle primitive data types. In addition, it doesn’t offer as much features as SOAP, and some consider this lack of features an unworthy quality for large scale applications. Furthermore, the translation of large sets of data to and from XML requires extra information for the conversion process and creates a performance bottleneck.[13]

3.2.2 SOAP

SOAP, which stands for Simple Object Access Protocol, also uses XML as the encoding scheme.[14] It offers more features than XML-RPC, which gives the developer greater freedom to transport complex data structures. As a result, SOAP has been the more ideal choice for some larger, business oriented solutions.[13]

On the other hand, the extra features that SOAP provides are often overlooked. In many cases, XML-RPC will perform the job adequately. SOAP also comes with a higher learning curve, which detracts some people from using the protocol. Like XML-RPC, SOAP also has performance limitations.[13]

3.2.3 CORBA

CORBA stands for Common Object Request Broker Architecture, and is managed by the Object Management Group. In CORBA, services are represented as objects, and consumers (also called the clients) are given object references, values that give clients indirect access to the service objects. To use the service, the client makes a request through the Object Request Broker (ORB), and the ORB redirects the request to the service object and returns the service’s results back to the client.[15]

The Interface Definiton Language (IDL) for CORBA provides interface definitions to service objects, but the language does not allow developers to describe the service’s implementation. Instead, it provides ”language bindings for many different programming languages,” which means the IDL compiler can generate ”skeleton” code for the object in different languages like Java or C++. The programmer is then left with the task of filling in the ”skeleon” with the necessary logic that satisfy the service contracts.[15]

According to one critic, CORBA is well-supported and works nicely with the popular programming languages. However, he believes CORBA is ”complex,” ”has a steep learning curve, requires significant effort to implement, and requires fairly sophisticated clients.” CORBA is ”better-suited to enterprise and desktop applications than it is to distributed web applications.” He believes CORBA has high enough quality for small business software but believes its learning complexities outweigh its use in distributed applications, software that operate across multiple computers.[16]

3.2.4 DCE

The Distributed Computing Environment (DCE) ”is an industry-standard, vendor neutral set of distributed computing technologies” and it includes an implementation of remote procedure calls to support distributed applications.[17][18] The system is similar to CORBA and has its own IDL compiler to generate ”skeleton” code for different languages. DCE is older but considered more powerful than CORBA because it ”offers security and authentification through Kerberos,” a ”network authentification protocol.”[18][19] Thus DCE is among the favorite solutions for building large applications, but like CORBA, DCE would also involve a relatively steep learning curve.[18]

The material above is a short introduction to some available technologies for ISCA, and a more thorough analysis is required if remote procedure calls are to be incorporated as part of a service-oriented solution.

4 Software Prototyping for ISCA

In many engineering disciplines, a prototype is often created to help the engineers solidify their concepts of the end product. Software engineering is no different. Some benefits of prototyping include ”gaining understanding of the requirements, reducing the complexity of the problem and providing an early validation of the system design.”[20] For a complex system like ISCA, a prototype could vastly improve system performance by helping software engineers analyze a current solution before proceeding to the next stage of development.

4.1 Initial Phase of ISCA Prototyping

One of the most frequently used libraries at JPL is SPICE, a collection of functions for calculating positions, speeds, frame transformations, vector angle separations, etc. of Solar System bodies. Creating an interface to SPICE for other models to use is a major requirement for ISCA to succeed. Therefore, my initial step in building a prototype rested on creating a service-oriented SPICE module.

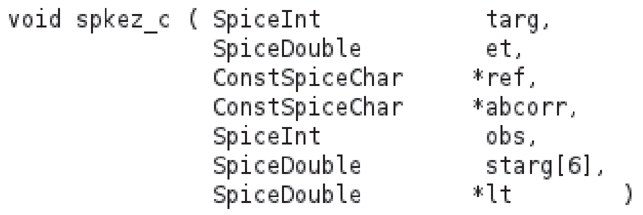
The group that manages SPICE, NASA’s Navigation and Ancillary Information Facility (NAIF), wrote the SPICE libraries in four flavors: C, Fortran, Matlab, and the Interactive Data Language. I decided to try out XML-RPC with C/C++ as my communication mechanism, mainly because XML-RPC is simple and I knew C, which saved me the time of learning a new language. A person with some programming experience could quickly learn how to build an XML-RPC server and client by looking at the documentation online.

Most of the C SPICE functions return values by address - they can return more than one value at a time - and the solution I arrived at meant the service layer needed to push all the results into an array and return the array. As I was analyzing the SPICE functions, I realized I could automate the process of writing server side code. I wrote a program that scans through the HTML documents in the CSPICE toolkit (available from the NAIF website, see [21]) and creates a file containing all the information I would need to write a code generator. Each line in the file contains meta-data for one function, which are the function return type, the function name, and the input and output parameter names and types.

Once I had the properties file, I wrote a code generator to create C++ classes and class methods to interact with the SPICE functions, one class for each SPICE function. Each of the classes define interface objects that act as interfaces between an XML-RPC server and the SPICE functions. To create a parellel with the real world, an XML-RPC server can be seen as a shipping port. The server may listen at different IP addresses, which correspond to different docks at the port. For the prototype, the server only listens at one IP address. When the server receives network data in XML form, this is the same as a ship arriving at a dock to deliver cargo.

Each of the SPICE functions can be viewed as a factory waiting for their cargo to arrive at the port. The C++ objects act like factory employees, providing the link between the port and the factory. They are responsible for transporting the cargo to and from the factory, which represent the decoding of XML data to SPICE data and the encoding of the SPICE results into XML data.

Most of the generated code focus on the XML and SPICE translation. When the SPICE function finishes its computation, all the arguments to the function, including the input arguments, are pushed onto an array. This allows the client to look at the parameters and determine if any side effects occurred with the function inputs. If the SPICE function has a void return type, the order of the values in the array follow the same order as the parameters that appear in the function prototype. If the function has a non-void return type, the SPICE return value is pushed onto the array first, followed by the arguments to the function as described previously.



```
void spkez_c ( SpiceInt          targ,
               SpiceDouble       et,
               ConstSpiceChar   *ref,
               ConstSpiceChar   *abcorr,
               SpiceInt          obs,
               SpiceDouble       starg[6],
               SpiceDouble      *lt        )
```

Fig. 2. SPICE function prototype in C.

Consider spkez_c and it’s function prototype in Figure 2. It returns the state of a target body - the position and speeds in rectangular coordinates - relative to an observer in some reference frame through the variable ”starg.” It also returns the amount of time it takes for light to travel between the observer and target through the variable ”lt.”

The function spkez_c returns 7 values (6 SpiceDoubles from ”starg,” 1 SpiceDouble from ”lt”) through 2 variables. To return these results back to the client, the interface object creates an XML-RPC array, converts ”targ” to an XML-RPC integer, "et" to an XML-RPC double, "ref" and "abcorr" to XML-RPC byte strings and so on. For "starg," the interface object converts it into its own XML-RPC array, then the object stores the array into the larger array of results.

Once I completed the SPICE server, I wrote a simple Java client to make requests to the SPICE server. Given SPICE kernels from the CSPICE toolkit, the Java client uses data from the Cassini mission to plot the state of the spacecraft from Earth. In addition, the client also plots the angular separation between the Cassini High Gain Antenna (HGA) boresight vector and the position of Earth vector relative to the HGA frame. Both plots go for 3600 seconds starting from 2004 JUN 11 19:32:00. The plots are given in Figures 3, 4, and 5, and the client uses JFreeChart to plot the data.

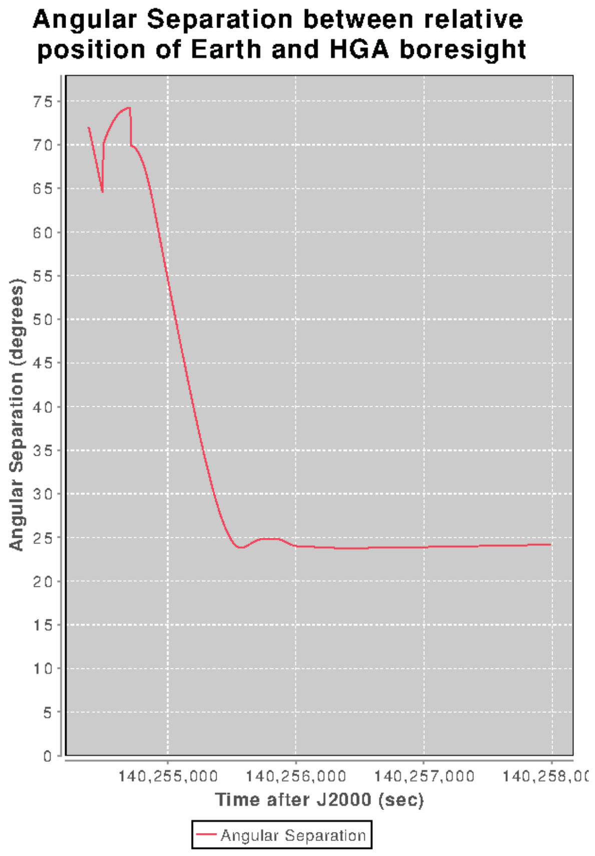


Fig. 3. Angular separation between Cassini HGA boresight and position of Earth vectors.

Unfortunately, there was not enough time to verify the correctness of these plots, but it demonstrates that models and tools can interact with the SPICE functions regardless of the underlying implementation language.

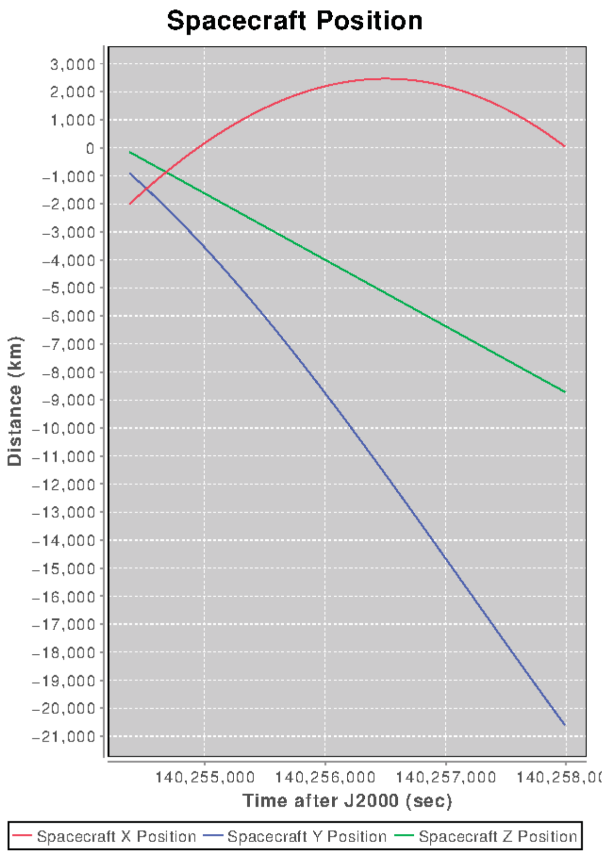


Fig. 4. Position of Cassini relative to Earth.

4.2 The Next Step

Members of the planning and sequencing team at JPL sometimes write models in one particular language like the APGEN adaptation language. However, SPICE functions only exist in C, Fortran, Matlab, and the Interactive Data Language, which means developers have to write applications in one of those four languages in order to use SPICE. With a service, it decouples the application from SPICE by resolving language dependencies between the two software entities.

The next step is to refactor one of the models written in the adaptation language to make calls to the SPICE service. A good choice is an ACS model from Deep Impact. Interface code must be written again such that APGEN will make remote procedure calls to the SPICE service, but this process can be done once. Afterwards, modelers only have to work with the adaptation language. As long as a compiler or interpreter recognizes the adaptation language and the service requests, new versions of APGEN can be deployed without modifying a model in the adaptation language.

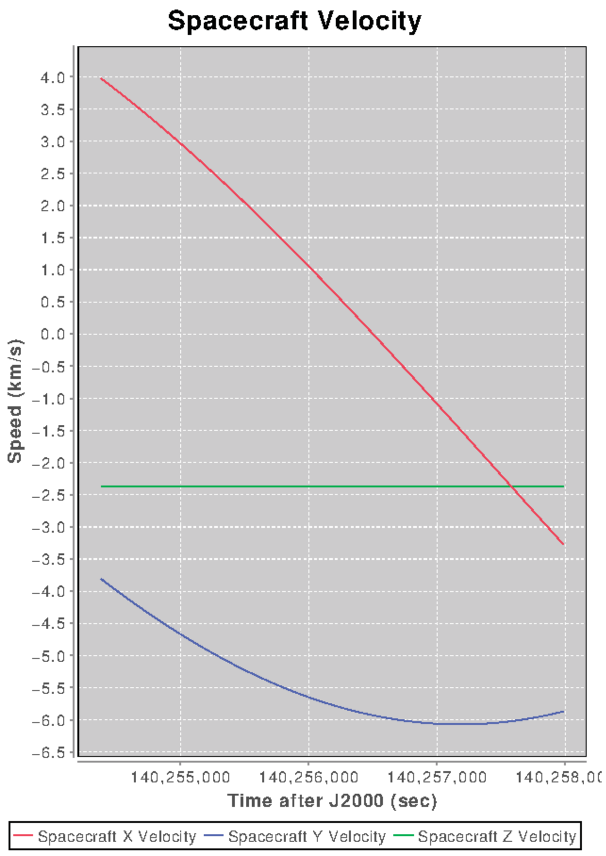


Fig. 5. The velocity of Cassini relative to Earth.

5 Concluding Remarks

As I wrote my code for the Java client, it became clear that modelers who wish to use SPICE from a different language must implement their own version of the SPICE data types, such as SpicePlane or SpiceEllipse. In addition, remote procedure calls are also intended to work across networks. As a result, the time it takes for the procedures to run and return the results may be slower than if the procedures ran locally on the same machine.

Yet, network speed has dramatically improved over the years, lowering the concern for network overhead. Additionally, object oriented programming languages make the extra development of data types an easy task, and it certainly requires much less overhead than writing wrappers to SPICE or reimplementing them in the desired language. I believe the greatest difficult in designing ISCA lies on finding a set of capabilities that can accomodate a plethora of past models without limiting the power of future models. With a service-oriented solution, this may be achievable, paving the way for a maintainable and extensible system.

6 Acknowlegements

Many thanks to my mentors, Yu-Wen Tung and Pierre Maldague, my co-interns, Cody Hyman and Erica Wicks, and Steve Wissler of the EPOXI team for the support, knowledge, and direction they provided during my tenure at JPL. This research was carried out at the Jet Propulsion Laboratory, California Institute of Technology, and was sponsored by the Space Grant Program and the National Aeronautics and Space Administration.

References

[1] N. B. R. P. on Simulation-Based Engineering Science, "Simulation-based engineering science," National Science Foundation, Tech. Rep., May 2006.
[2] "Why use ammos?" Retrieved August 10, 2011. [Online]. Available: http://ammos.jpl.nasa.gov/whyammos/
[3] S. Wissler, P. Maldague, J. Rocca, and C. Seybold, "Deep impact sequence planning using multi-mission adaptable planning tools with integrated spacecraft models," in *AIAA 9th International Conference on Space Operations*, June 2006.
[4] T. Erl, "What is soa? - an introduction to service-oriented computing," Retrieved August 10, 2011. [Online]. Available: http://www.whatissoa.com
[5] ——, "Soa principles - an introduction to the service-orientation paradigm," Retrieved August 10, 2011. [Online]. Available: http://www.soaprinciples.com
[6] K. Baley, D. Belcham, and J. Kovacs, "Separation of concerns," Retrieved August 9, 2011. [Online]. Available: http://msdn.microsoft.com/en-us/magazine/ek- stremalna-przerobka-asp-net–czesc6-podzial-obowiazkow.aspx
[7] M. Poulin, "Evolution of principles of service orientation: Service autonomy, part 5," February 2009. [Online]. Available: http://www.ebizq.net/blogs/service_oriented/20-09/02/evolution_of_principles_of_service_or- ientation_service_autonomy_part_5.php
[8] ——, "Evolution of principles of service orientation: Service statelessness, part 6," February 2009. [Online]. Available: http://www.ebizq.net/blogs/service_oriented/20-09/02/evolution_of_principles_of_service_or- ientation_service_statelessness_part_6.php
[9] ——, "Evolution of principles of service orientation: Composability and discoverability, part 7," February 2009. [Online]. Available: http://www.ebizq.net/blogs/service_oriented/20-09/02/evolution_of_principles_of_service_or-ientation_service_composability_and_discover-ability_part_7.php
[10] "Definition of procedure," Retrieved August 10, 2011. [Online]. Available: http://dictionary.reference.com/browse/procedure
[11] A. D. Birrell and B. J. Nelson, "Implementing remote procedure calls," *ACM Transactions on Computer Systems*, vol. 2, no. 1, pp. 39–59, 1984.
[12] D. Winer, "What is xml-rpc?" July 2003. [Online]. Available: http://www.xmlrpc.com

[13] M. Dehaan, "Xmlrpc vs rest vs soap vs cim vs rmi vs message- bus vs ... lots of rpc options," July 2008. [Online]. Available: http://michaeldehaan.net/2008/07/17/xmlrpc-vs- rest-vs-soap-vs-all-your-rpc-options/

[14] M. Gudgin, M. Hadley, N. Mendelsohn, J.-J. Moreau, H. F. Nielsen, A. Karmarker, and Y. Lafon, "Soap version 1.2," April 2007. [Online]. Available: http://www.w3.org/TR/soap

[15] jGuru.com, "Introduction to corba," 1998. [Online]. Available: http://java.sun.com/developer/onlineTraining/cor-ba/corba.html#anchor312271

[16] E. Kidd, "Xml-rpc howto," April 2001. [Online]. Available: http://tldp.org/HOWTO/XML-RPC-HOWTO/index.html

[17] "What is distributed computing and dce?" January Retrieved August 10, 2011. [Online]. Available: http://www.opengroup.org/dce/

[18] L. Vepstas, "Technologies similar to corba," October 2001. [Online]. Available: http://linas.org/linux/corba.html

[19] "What is kerberos?" May 2011. [Online]. Available: http://web.mit.edu/Kerberos/#what_is

[20] L. Bernstein, "Foreword: Importance of software prototyping," *Journal of Systems Integration*, vol. 6, no. 1-2, 1996.

[21] "Spice," Retreived August 11, 2011. [Online]. Available: http://naif.jpl.nasa.gov/naif/