



# **Integrated Architecture Framework for Simulation**

## **Composability Study**

Rev. 2.9  
June 30, 2004

Clear Methods, Inc

[www.clearmethods.com](http://www.clearmethods.com)

Technical point of contact

Christopher Fry  
CFry@ClearMethods.com  
One Broadway, 14th Floor  
Cambridge MA 02142  
617-475-1633

Administrative point of contact

Bob Nilsson  
bnilsson@ClearMethods.com  
One Broadway, 14th Floor  
Cambridge MA 02142  
617-475-1634

# Clear Methods®



## Table of Contents

Summary .....	4
Authors.....	6
Acknowledgement .....	6
Disclaimer .....	6
Structure of This Study .....	7
1. INTEGRATED SOLUTION.....	8
Integrated Solution Criteria.....	8
Code ↔ Data (data transformation, dynamic data) .....	8
Code ↔ Presentation (user interface, browsers).....	9
Ease of Use (training, project management, Tower of Babel).....	9
Integrated Solution Candidates .....	9
Custom Language .....	9
Multi-Language.....	10
Water .....	10
UML/MDA .....	11
2. CODE .....	12
Code Criteria.....	13
Generality (flexibility, extensibility, composability).....	13
Functionality (capability, security).....	13
Compatibility (interoperability, Web services).....	13
Ease of Use (learning, specification, testing, documentation).....	14
Performance (runtime speed, scalability, distributed computing) .....	14
Code Candidates .....	15
C, C++ .....	15
C#.....	15
Java .....	16
JavaScript.....	17
Common Lisp.....	17
Water.....	18
Other Languages .....	19
Ada .....	19
Curl .....	19
Perl .....	19
PHP .....	20
Python .....	20
Ruby.....	20
3. PRESENTATION .....	20
Presentation Criteria.....	20
Functionality .....	20
Compatibility .....	20
Ease of Use .....	20
Performance (compute power needed to render graphics) .....	20
Presentation Criteria Summary .....	21
Presentation Candidates .....	21

HTML/XHTML.....	21
SVG (2D graphics in XML format).....	21
X3D (VRML).....	22
Java (including AWT, Swing, SWT, Java 3D, Java 2D).....	24
JView .....	25
<b>4. DATA REPRESENTATION.....</b>	<b>25</b>
Data Representation Criteria.....	25
Functionality .....	25
Compatibility with Existing Practice .....	25
Conciseness.....	25
Data Ambiguity.....	26
Ease Of Use.....	26
Data Criteria Summary .....	26
Data Representation Candidates .....	26
Comma Separated Values (CSV).....	26
XML (including DTD and XML Schema) .....	27
ConciseXML.....	28
Conclusion and Summary.....	30
Appendix – Spreadsheets.....	31
Appendix – Further References .....	35

## Summary

This study compares the relative suitability of candidate technologies for composing modeling and simulation assets within an integration framework. The ability to interconnect new and existing simulations is essential to a meaningful integrated simulation architecture. Data interoperability is discussed as a necessary part of its scope. The study partitions the simulation composability challenge into the following four areas.

1. **Integrated Solution** (interoperability of data, presentation, and code)
2. **Presentation**
3. **Code**
4. **Data Representation**

Fifteen candidate technologies are evaluated in each of these areas. For each of these areas, evaluation criteria are described, the criteria are numerically weighted, and each candidate technology is rated against each criteria.

The following are the candidate technologies that have been included in this study.

### Integrated Solution

1. Multilanguage
2. Unified Modeling Language/Model-Driven Architecture (UML/MDA)
3. Water

### Code

1. C, C++
2. C#
3. Java
4. JavaScript
5. Common Lisp
6. Water

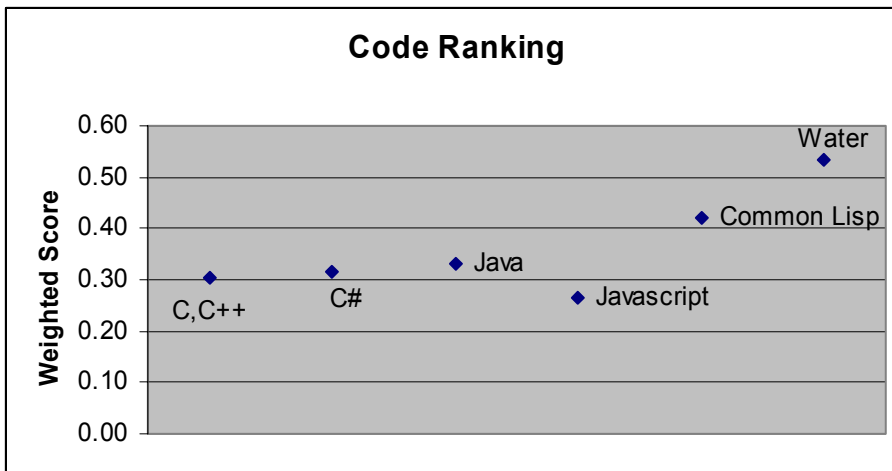
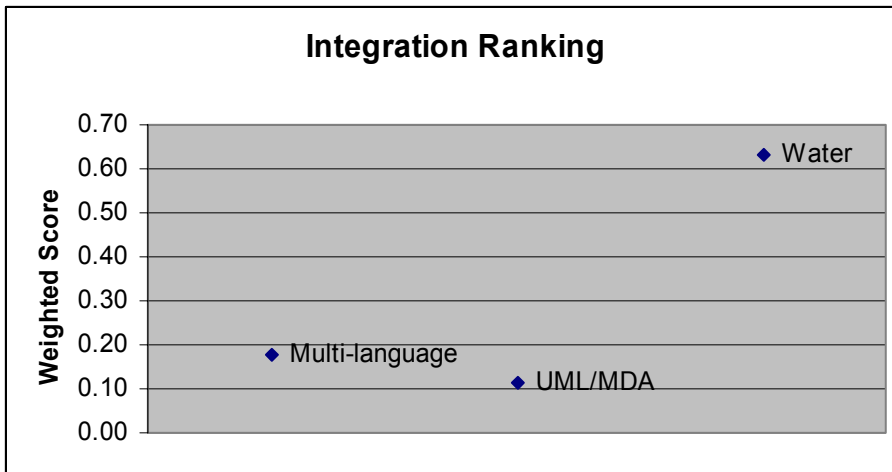
### Presentation

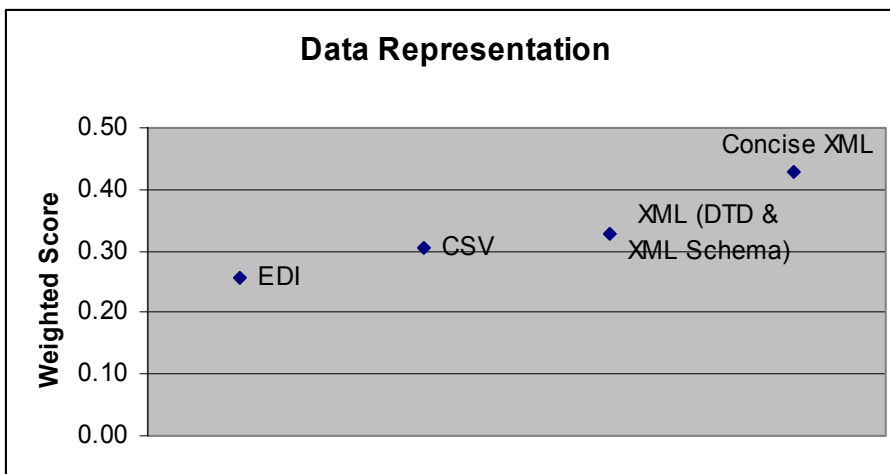
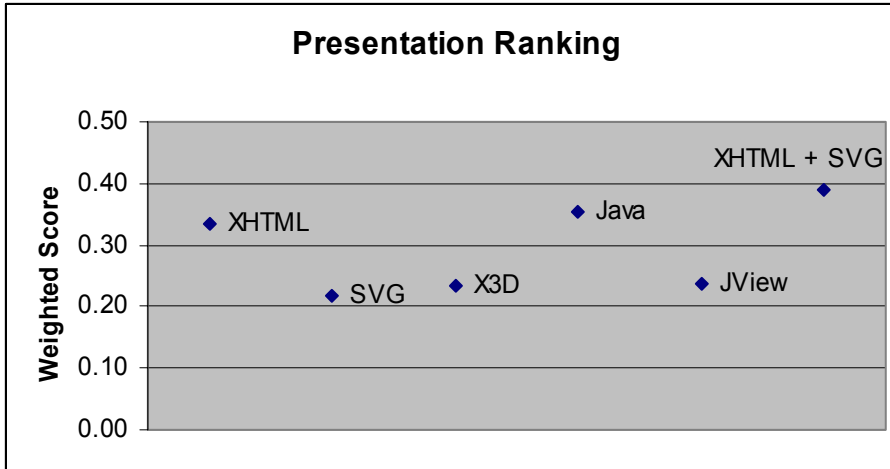
1. XHTML
2. SVG
3. X3D
4. Java
5. JView
6. XHTML+SVG

**Data Representation**

1. EDI
2. Comma Separated Values (CSV)
3. XML (DTD+XML Schema)
4. ConciseXML

An Excel workbook, reproduced in the appendix, contains the results of this evaluation in four spreadsheets. Below are the summary results of the spreadsheet analysis.





**Authors**

Christopher Fry, Bob Nilsson, and Stephen Plusch of Clear Methods, Inc.

**Acknowledgement**

The work described in this study was funded, in part, by the US Air Force Joint Synthetic Battlespace (JSB) Program with funding managed by Northrop-Grumman Corp. during October and November, 2003.

**Disclaimer**

This paper is the authors' work product and does not reflect the opinion of either the JSB Program Office or the US Government.

## Study Purpose

This study evaluates candidate technologies for composability of existing and new modeling and simulation assets within an integration framework.

## Structure of This Study

Due to the complexity of the composability problem, the study is divided into four major areas.

1. **Integrated Solution** (interoperability of data, presentation, and code)
2. **Code**
3. **Presentation**
4. **Data Representation**

Fifteen candidate technologies are evaluated in this study. Within each of the areas, the evaluation criteria are described and the technologies are evaluated against the criteria.

Summary spreadsheets, included in the appendix, present scores showing how well each technology satisfies the criterion. Note that while the underlying methodology of the spreadsheet is sound (based on summing the weighted criteria scores to yield a summary for each candidate), educated people may disagree on the raw scores assigned to each candidate-criteria cell, as well as the weights given to each criterion. The scores were assigned by Clear Methods technical personnel. An electronic copy of the spreadsheet is provided, with this paper so alternate scenarios may be tested. While it is possible to adjust the individual scores to obtain almost any outcome, it is more important to understand the reasoning for each score. That is the purpose of the discussion in this study.

### The Reality of Web-Enabled Programming

Most Web applications are programmed in several languages, each with their own idiosyncratic syntax, data and semantic representations. This introduces unnecessary complexity, which in turn increases development time and cost, as well as undermines maintainability, reliability, testability and security of the application.

### Clear Methods Full Disclosure

Clear Methods was founded to solve the problems caused by the multi-language and multi-tool approach. These same issues arise in an integration framework for composability. Clear Methods' technology provides a breadth of functionality that subsumes many individual languages and technologies. It also interoperates with the most prevalent languages. The goal of the integration framework is closely aligned with the goals of Clear Methods. In a real sense, Clear Methods has been working on a solution to this problem for years. If we did not think our solution was better than the tacked-together mish-mash of popular Web technologies, Clear Methods would not exist.

# 1. INTEGRATED SOLUTION

As described above, a good framework will need the functionalities of data, presentation and code. To achieve maximum utility; these functionalities must be integrated. For example, when presenting the results of several simulations, the output data may need to be filtered, sorted and aggregated, requiring data manipulation and potentially a wide variety of code in addition to the presentation software itself. Thus, it is important that any complete solution for a framework be considered in light of how presentation, data and code will interact. The smoother this interaction, the easier and more flexible the framework will be.

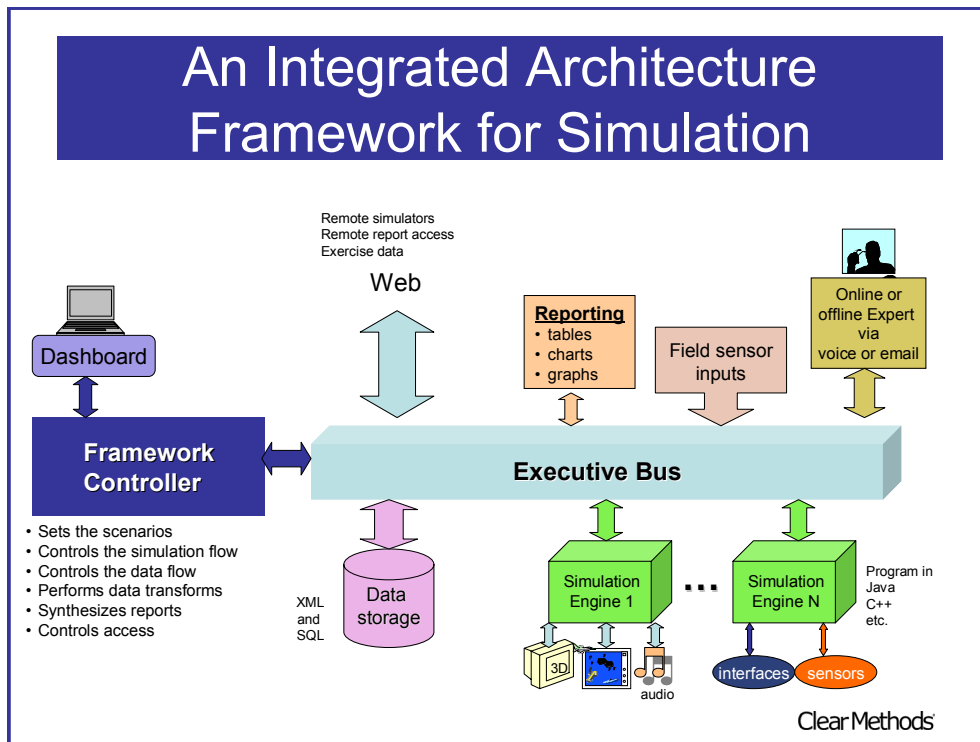


Figure 1 - A Framework Controller can provide simulation scenarios and control for multiple parallel simulations. It can also offer real-time dashboard control, a gateway to Web communication as well as report generation.

## Integrated Solution Criteria

### Code ↔ Data (data transformation, dynamic data)

Data from sensors, simulators, simulations, databases, files, etc. will be found in different formats yet are expected to be in the appropriate format when required as input to a particular simulation or presentation software application. Thus, data transformation is needed. Although some transformations are very stylized and limited, the capability for very complex, idiosyncratic transformations must be included to take advantage of all possible data sources. The more kinds of transformations that are included, the more comprehensive the framework will be.

One of the reasons transformations are required is that sometimes there is a limitation in the data

representation format that precludes it from containing some kind of information, or precludes representing a particular kind of data in the format needed by consumers of the data. If active data is placed in a data hierarchy, one can

- reduce the need for extra processing steps,
- reduce the need for extra meta data, and
- Increase the modularity of information by packaging all the essential data together.

If code can be treated as data, then code can manipulate code just as it manipulates data. This includes storage as well as all sorts of transformations. Code manipulating code is not common, but when it is relevant, it can pay huge dividends.

### **Code ↔ Presentation (user interface, browsers)**

Usually when data is to be presented, it needs some transformation. This can be in sorting, filtering, averaging, summarizing, etc. Constructing usable displays may also involve considerable computation to maximize screen real-estate for large volumes of data. Getting the data from a user and feeding it to the right piece of code is a headache in many programming environments, so simplifying that processes helps reduce user interface implementation complexity.

### **Ease of Use (training, project management, Tower of Babel)**

Application solutions that require a different language for data, presentation and code as complexity as they

- increase the difficulty in moving information from one part of an application to another,
- require expertise in several languages, and
- inhibit communication among team members not providing a common language

Some solutions, particularly in the Web space require more than three languages. This Tower of Babel increases complexity to the point of unmanageability for both the individual programmer and the manager. Having fewer languages, ideally one, allows a manager much more flexibility in assigning people to tasks and reducing overall training costs.

## ***Integrated Solution Candidates***

It is impossible to enumerate the myriad technologies used for developing and deploying complex Web applications, let alone do justice in describing them. To simplify this problem we have created the broad categories of Custom Language (discussed below and eliminated as impractical), Multi-language and UML/MDA. To this list, we have added Water, a language well equipped to solve this class of integration problem.

### **Custom Language**

Let us hypothesize a language designed specifically to solve the Simulation Composability. If it were high level, simple and only focused on a few commands for launching, routing data to and receiving data from simulators, it would appear at first glance to be an ideal solution. But take a deeper look at the real capabilities needed. It would need to be able to transform data in order to resolve incompatible data formats among simulators. That alone would require a general purpose language and thus is no longer simple. If our High Level language called out to an existing general purpose language, say Java, then one has all the problems associated with multi-language programming (see below). Furthermore, to get input from a user and present information back to the user, the language would need to interface to a windowing system. So one would end up duplicating the work already done and would not necessarily end up with a better solution than other alternatives. Such a hypothetical simulation-integration language will not be discussed further except to say that it would be very expensive as well as risky to develop.

## Multi-Language

This potential solution represents a very large family of solutions used today for most Web programming. It includes languages for code, data and presentation, often more than one language for each. For example an application might use the following.

- Java for server side processing
- JavaScript for client side processing
- HTML for client-side text and images
- Browser plug-in for additional rendering capabilities for animation, 3D graphics, etc.
- SQL for large data sets
- HTML for client-side text and images
- XML for smaller data sets
- XSLT to transform the XML

Each of these has numerous possibilities. For the server side processing see the list of candidate languages in the **Code** section below. The HTML might be augmented with CSS (Cascading Style Sheets). XML might be replaced or augmented with Comma Separated Value (CSV) data files, etc. Various configurations of these technologies will be more or less useful for various applications. But they will all have difficulties communicating with each other. Debugging will be painful at best. The programmer's bookshelf will be overflowing with thick reference manuals. The pieces of the application will be inherently hard to specify, document, and test.

### Code ↔ Data – Multi-Language

Rating: Low. Different languages make interoperability difficult.

### Code ↔ Presentation – Multi-Language

Rating: Low. Different languages make interoperability difficult.

### Ease of Use – Multi-Language

Rating: Low. The theoretical advantage of having narrow languages optimized for specific tasks is outweighed by the fact that different languages make interoperability difficult.

## Water

Water was designed to simplify the Tower of Babel of the multi-language approach. It was also designed for the Web. Its native syntax is XML. Each XHTML tag is a built-in class in Water. In order to accommodate the functionality of many different languages, Water was designed with extreme generality in mind. But, had that generality required too much learning, it would have never be used. So consistency and elegance were overriding design goals.

Water's designers built on the growing popularity of XML and its goal of becoming a common syntax. However, XML has certain limitations that make describing static data difficult and dynamic data horrendous. Water uses a superset of XML called ConciseXML that alleviates the worst of XML's limitations. With a parser that accepts both XML 1.0 and ConciseXML intermixed, and with tools that convert between the two automatically, a developer can work in whichever syntax is most convenient. ConciseXML is a superset including XML 1.0 syntax. Ref: [www.ConciseXML.org](http://www.ConciseXML.org).

Clear Methods undertook a proof-of-concept project during the summer of 2003 under contract to the MIT Engineering Systems Division. The project resulted in a discrete event simulator. The entire simulation engine, as well as specific simulations using the engine, was written in Water. See [www.WaterLanguage.org](http://www.WaterLanguage.org)

### **Code ↔ Data – Water**

Rating: High. Water's syntax is XML. There is only one parser for code and data. There is only one object system, unlike the Multilanguage approach which generally has one Object System for the language and an incompatible one, usually referred to as the Document Object Model (DOM) for parsed XML. With Water, code can be embedded in the XML for dynamic data. Code can also be manipulated as data and then used as code by calling the execution engine explicitly. (In Lisp, this facility is called *eval*. In Water it is called *execute*.)

### **Code ↔ Presentation – Water**

Rating: High. XHTML is a subset of Water. As such, all XHTML Web pages are valid Water programs. Code can be nested in HTML. It looks just like another tag syntactically. Also the value of any attribute can be expressed in Water. Water has yet to take advantage of emerging XML-based graphics descriptions such as SVG and X3D, though a proof of concept demo has been made for both of those. There is also a proof-of-concept interface between JView and Water using Water's facility for calling any methods written in Java.

### **Ease of Use – Water**

Rating: High. Water solves the multi-language problem by enabling a solution in just one language. Furthermore, the language and Steam XML IDE are designed to make incremental software development easy. There is no compile step, no need to launch a browser to see rendered HTML, and any selected code can be individually executed.

A specification can be written with Water class and method stubs. Documentation is a built-in facility using XML. The *inspector* can traverse all Water objects including the documentation. Testing is built-in and can be accomplished programmatically or via the Steam XML user interface.

Water is relatively new and while it may ultimately be able to achieve the goal of implementing all needed functionality in just one language, it also has an easy means to call all Java methods. Java objects can be passed around inside of Water. In fact, many of the Water primitive data types are Java instances. Water is also able to use SQL for database queries, though an even more elegant solution is under development. Water has a facility to use Regex for string matching in both the original Regex syntax and an expanded, human readable form. Finally, Water can use complex URLs to encode arguments in Water functions that are delivered as Web services by the Water server.

## **UML/MDA**

The combination of UML and MDA is still in its early stages. It appears that the ultimate goal is to allow the user to write an application in UML, and then automatically translate the UML into Java or C# or another language, which is then compiled to make a runtime version. UML/MDA claims that it is uniquely platform independent. Similarly, Java claims it is platform independent as does C with different compilers for each platform. In the case of UML/MDA, Platform means Java or C or another language. In Java's case, platform means Windows or Unix or an OS that supports the JVM. It is just a question of level.

What remains to be seen is whether the UML/MDA solution can make it easy to develop working applications that run on a wide variety of systems. Introducing an extra layer definitely increases complexity, unreliability, and incompatibilities of many sorts. But are these offset by other advantages of UML/MDA?

If the manipulation done to UML to get a program to work is completely transparent to the programmer, then UML becomes effectively a programming language. The question becomes, is it better than the alternative programming languages? First, it is not clear that UML/MDA has achieved automatic compilation without requiring the programmer to at least work with the underlying language. Second, is it really the expectation of the designers of UML/MDA to actually achieve this goal? The UML/MDA solution may return us to the multi-language development environment with UML itself plus an additional language for each platform. Unfortunately, these additional languages have a more complex relationship with each other than the sibling relationship that the multi-language solution has. In UML, if you have to work with one of the other underlying platform languages, then how does that information get back into the UML language, especially in a platform-language independent way?

### **Code ↔ Data - UML/MDA**

Rating: Low. Since UML/MDA produces source code in target languages, the functionality of the code cannot be greater than those languages (at least not without a lot of work). Furthermore, UML/MDA's claim is to be capable of producing source for several different platform languages. So the UML/MDA itself can only describe functionality that is the lowest common denominator of the functionalities of all the output languages. Since the functionality and generality of the popular languages is low, UML/MDA's functionality must be low.

### **Code ↔ Presentation - UML/MDA**

Rating: Low. Same rationale as Code↔Data.

### **Ease of Use - UML/MDA**

Rating: Low. UML focuses on diagrams rather than text. Diagrams are often good at creating high level descriptions for presentations. The use of diagrams for coding is mixed at best.

See Marian Petre, **Why Looking Isn't Always Seeing**, Communications of the ACM June 1995. It is difficult to find programmers who prefer UML for writing all the code for a given application. Furthermore, the process of creating a UML program, then compiling it to a platform language, then compiling that language, then running the resulting program is open to introducing errors. Even if it were not, it is a long way from Water's incremental software development ease.

## **2. CODE**

The framework must provide connections among simulators. These connections move data, transform it, and potentially control data flow among simulators. Each of these tasks could be performed simply and without executing code, but a framework that provides computational capability will be able to derive the most benefit from the component simulators. For example, transforming data is useful for passing values among incompatible simulators, for generating the data for composite presentations, and for determining which simulation to run based on the values computed by a previous simulation. To have flexibility for these and other perhaps unanticipated functionalities of the framework, a flexible language is needed.

A second reason to incorporate the ability to generate code into the framework is to make it especially easy to prototype new simulators. This might be done to

- provide a proof of concept for a new kind of simulation
- provide a *prototype* simulator until the real one becomes available
- provide a set of proposed simulators and *simulate* how they will all interact even before the real simulators are available

Though this kind of functionality is not, strictly speaking, within the scope of the framework, it can never the less speed development and testing of complex sets of interacting simulators. Rapid

prototyping gives early feedback and demonstrates new ideas which allow design flaws to be caught and corrected early.

## **Code Criteria**

Though the issues surrounding data representation and presentation are not simple, code is by far the most complex of the three areas to evaluate. Code is responsible for moving data among simulations and presentations with perhaps arbitrary transformations of the data in between. Code is responsible not just for the logic and behavior of a program, but it also takes care of deficiencies in data representation and presentation software.

### **Generality (flexibility, extensibility, composability)**

The number one code criterion is generality. Flexibility is important for code interfacing with itself. If flexibility is taken to the next level it approaches the concept of composability, that is, the ability to mix and match various components of the software in a wide variety of configurations. A programming language that is flexible and allows composability will also promote reuse.

No language can be expected to have all the functionality that will ever be needed built in from the start. That is why extensibility is critical to prevent road blocks during implementation. By making it easy to extend a language you provide a solution for

- unusual situations
- new, unforeseen requirements
- the ability to repurpose and adapt already-written components to situations different from those the component-designers had anticipated
- reconfiguring existing components to fit new situations (the core of composability)

### **Functionality (capability, security)**

The functionality of a programming language refers to its built-in capabilities. Given a language with a very few primitives, theoretically anything could be built with it. Functionality really deals with the predefined methods and classes that give the programmer a head start on domain-specific code. Groups of these functionalities are often called libraries, which hold, for example, graphics or Internet functionality.

Another important dimension is security. Is security built in or added on? How strong is it? How easy is it for the programmer to use? Security mechanisms that are built-in but difficult to use will often be circumvented, compromising the integrity of the entire project.

### **Compatibility (interoperability, Web services)**

The generality criterion covers the degree to which a programming language interfaces with itself. Compatibility refers to how well a language interfaces with other languages. There are two cases:

- A human, semantic sense of a mapping of data structures and function calls
- A programmatic sense via foreign function calls

Recently, compatibility has been extended to mean how well a language can interface with Web services. Can the language effectively implement a remote procedure call when nothing is known about the language of the other program. What is known is that it supports a particular Internet protocol such as SOAP.

A list of languages important for compatibility include the following.

- C/C++
- Perl

- Java
- XML
- HTML
- Web Services (SOAP, WSDL, UDDI, URLs)

Other programming languages, including Ada, COBOL, Basic, FORTRAN, etc. may be important for interfacing with legacy systems.

Note: The broader issue of interoperability with presentation and data is covered within the section on Integration.

### **Ease of Use (learning, specification, testing, documentation)**

Two capabilities have been grouped under the ease of use category. One is the ease of learning and using the programming language itself. This includes the central tasks of reading, writing and debugging code.

The second capability, less obvious, but no less important, is for writing the right functionality with the right level of reliability. This includes tools for managing the ancillary tasks in programming. Often, these tools are not well integrated with the core task of programming itself so they are done in a more ad-hoc fashion, costing time, lacking completeness and generally degrading overall project quality. These capabilities are

- Overall learning
- Specification
- Testing
- Detailed documentation

Economic studies have found that software maintenance accounts for the majority of the cost of software over its lifecycle. An emerging important property of systems that are easy to use is maintainability. Ease of reading, writing, debugging, specifying, testing and documenting help the task of maintaining a piece of software. Development environments that support incremental software development improve maintainability by making it easier to change the code and extend functionality without causing major disruption of the overall software system.

Java presents an example of the relationship between the programming language and its documentation. Java includes a facility called JavaDoc. It provides a way of writing documentation about individual Java classes and methods and for aggregating all the little pieces of documentation into a unified whole for printing and on-line browsing. It is a very useful utility. But it also points out a glaring omission in capability for the Java language itself. JavaDoc has a syntax all its own, unrelated to the normal Java syntax. Despite the fact that documentation needs a highly structured data set to operate, the authors of JavaDoc determined that Java itself was not sufficient to provide this and therefore required a new language and parser for the task. In contrast, Water uses a Water library to implement documentation, employing the existing Water syntax and object system for a much smoother integration.

### **Performance (runtime speed, scalability, distributed computing)**

Performance in programming languages, like conciseness in data representation, is no longer as critical due to the availability of inexpensive high-performance hardware. Distributing computationally-intensive jobs to distributed computing resources decreases the need for fast-running languages. For the purpose of real-time photo-realistic simulations, performance is still very much an issue. However, a goal of the framework is to direct traffic to and from such simulators, rather than perform the simulation locally. None the less, performance is still an issue in providing user responsiveness, especially if there are network latency issues. The category of

performance is not given a high importance but that weighting can be adjusted if the language is expected to handle tasks other than directing traffic and creating summary reports.

## **Code Candidates**

### **C, C++**

#### **Generality – C, C++**

Rating: Medium. The calling syntax of C makes it weak on flexibility. You cannot use key-worded arguments. C++ does have a mechanism for defaulting arguments, but does not allow passing arguments via keywords. The language does not permit functions that take an unlimited number of arguments. The object system does not permit the programmer to dynamically add instance variables to existing instances nor can the programmer change the parent of an object. This tightness can be thought of as an advantage to keep the code clean, but having to work around these restrictions when the need arises makes the code more cumbersome.

#### **Functionality – C, C++**

Rating: High. As one of the older languages in this section, C has plenty of libraries written for it, giving it high marks in functionality.

#### **Compatibility – C, C++**

Rating: Medium. C was the de facto standard in the 1980s and as such, is a standard that has meant compatibility with many programs. But, due to its inflexible calling and data structures, it is not particularly easy to connect C programs to programs written in other languages.

#### **Ease of Use – C, C++**

Rating: Low. C has had many years for the nurturing of development environments that would make it easier to use, but still it is not, at its core, easy to use. The precedence tables for C are difficult to master. Learning the ins and outs of C is so painful that C gets low marks here. Many users of C++ have converted back to C just because C++ is even worse than its parent in terms of ease of use.

#### **Performance – C, C++**

Rating: High. C was designed for speed. Many years of compiler optimization have made it the fastest language of the group.

### **C#**

C# is Microsoft's attempt to become independent of Java, a language they do not control. It is very similar to Java, although C# proponents claim it has improved upon a number of things in Java, but the differences are not great.

#### **Generality – C#**

Rating: Medium. Like Java. Restrictive calling conventions help tighten the code at the expense of flexibility and reuse.

#### **Functionality – C#**

Rating: Medium. Although C# can call C, being a fairly new language C#, libraries are not as extensive as those of C or Java..

### **Compatibility – C#**

Rating: Medium. Despite the fact that C# was designed post-Web for Microsoft's .NET initiative, C# itself has little to do with XML or HTML. It is fairly easy to learn for Java as well as C programmers.

### **Ease of Use – C#**

Rating: Low. Similar to Java.

### **Performance – C#**

Rating: High. C# runs on Microsoft's Common Language Runtime (CLR) platform, which is similar to Java's JVM. Over time, it will probably achieve speeds close to C.

## **Java**

### **Generality – Java**

Rating: Low. Java has been touted as a programming language for the Web, but its core was designed before the Web and as such has little to do with HTML or XML. Because of Java's fairly conventional method calls, it is not good at handling hypertext or XML.

In order to add a method to a class, the programmer must have the class source code file and compile the whole thing with the new method embedded in it. Some of these modularity limitations overcome by introducing subclasses and extending existing classes, but this has limited utility in many cases. Say for example you want to write a new method for the built in Java String class. You cannot. If you subclass String and add a method to the subclass, that method can be used only on instances of the subclass, not the String class. The new code, therefore, is not generally applicable. You can write a *static* method on your new class that takes an argument (not a *this*) of a regular string, but that breaks the whole point of object-oriented programming. You might as well use C from a modularity standpoint.

### **Functionality – Java**

Rating: High. The newer versions of Java include many built-in classes and many 3rd party libraries are available. Java still has trouble with such basic Web tasks as rendering modern versions of HTML well.

### **Compatibility – Java**

Rating: Medium. Java has a facility called JNI for interfacing native code such as C to Java. It also has add on networking support to call Web services. The Java RMI, remote messaging interface can be used to call functions over the net though you must be careful about formatting arguments correctly.

### **Ease of Use – Java**

Rating: Low. Programs must be compiled before they will run. You cannot easily run snippets of code, which is critical to reducing the edit-run-debug cycle. Java has quite a bit of idiosyncratic syntax. When defining a method, for example, you cannot use a function call. Rather this requires a rather strange construct that has a bunch of keywords at its beginning such as "public" or "static" or "void", none of which are evocative of the task at hand. Documentation is largely in Javadoc format. Often this is little more than a placeholder with only the name of the method itself. The Java classes and methods can generally be found in Javadoc but many of the most important things in the language such as control structures and operations on primitives are not in Javadoc because they are not *Objects*. This is just one of the many fallouts from the fact that Java is, in truth, only a half-object oriented language, causing all sorts of inconsistencies.

Despite these drawbacks, Java has achieved popularity. Over 1,000 books on Java are available at Amazon. One might question why so many books are necessary to learn a language.

### **Performance – Java**

Rating: Medium. Originally Java was slow due to its byte code interpreter. But as the Hotspot optimizing compiler came on-line, Java achieved speeds of within a factor of 2 or 3 of C code.

## **JavaScript**

### **Generality – JavaScript**

Rating: Low. JavaScript has an object system but it is rarely used. That is because it is not obvious how to use it, and many applications of JavaScript do not require complex data. So, JavaScript loses generality on its data structures. Its method calling is serviceable and can be used to construct rather complex programs. It does not support keywords or optional arguments that default to a programmer-specified value. As programs get large, JavaScript does not scale as well as more structured environments like Java.

### **Functionality – JavaScript**

Rating: Medium. JavaScript doesn't have nearly the library support that Java does. It is used in describing sophisticated behaviors in programs like X3D that do not have their own way to describe code.

### **Compatibility – JavaScript**

Rating: Medium. JavaScript is often embedded within HTML or XML data to give it dynamism. It's not particularly good at manipulating HTML or XML. JavaScript looks very similar to Java, so programmers tend to have an easy time moving from one to the other, though the languages are certainly not identical. JavaScript does not require the declaration of parameter types and return values of methods as with Java. So JavaScript is fairly compatible with Java from a learning perspective and there are even JavaScript interpreters that can run within Java.

### **Ease of Use – JavaScript**

Rating: Medium. JavaScript is easier to use than Java since it does not require a compile stage. You don't need to declare variables, which is a blessing for small, unsophisticated programs but may be a curse as your program gets larger and you would rather have the computer do compile-time type checking for you.

### **Performance – JavaScript**

Rating: Low. JavaScript is considerably slower than Java. For small interactive programs though, it is often plenty fast enough.

## **Common Lisp**

### **Generality – Lisp**

Rating: High. Lisp is the second most general of the programming languages described here. It has a long tradition in artificial intelligence, but in fact is a great general programming tool for many applications. Lisp's style of method calling and ability to treat code as data make it very flexible, especially for programs that write and execute code on the fly. Many Lisp programs are not so much applications as they are new domain-specific languages layered on top of the base Lisp. This style of use combined with Lisp's flexible method calling and macro system make it very extensible. The common Lisp object system has a lot of power but its complexity provides a barrier to learning.

### Functionality – Lisp

Rating: Medium. Common Lisp is a rather large language, though Java's libraries have outgrown it in sheer size. There are also not nearly so many libraries for Lisp as for Java or C, though due to Lisp's flexibilities, the libraries are often more reusable than libraries of other languages. Common Lisp never really standardized on an easy to use windows system, which is a reason it has decreased in popularity over the last couple of decades.

### Compatibility – Lisp

Rating: Medium. The generality of Lisp make it quite compatible for the serious programmer. As delivered, it is quite incompatible with other programming languages. Almost anything another language can do, Lisp can as well, albeit in a different syntax.

### Ease of Use – Lisp

Rating: Medium. Good implementations of Common Lisp have both interpreters and compilers. Using the interpreter, you can get fast turnaround on your ideas. Using the compiler you can get the code to run faster. Thus, a good Lisp has the best of both worlds. Lisp IDEs have not, in general, kept pace with IDEs for more conventional languages. Though the tools of the Lisp Machine and Macintosh Common Lisp of the 1980s were the best of their breed at the time, later IDEs of more conventional languages have surpassed them on their judicious use of graphics and color to help with the unnecessary complexities of those other languages.

### Performance – Lisp

Rating: High. High runtime performance in compiled mode. Fast for development in interpreted mode.

## Water

The Water language is an all-purpose programming language that uses an XML-based syntax. The language has been designed from the ground up to solve the Tower of Babel complexity of Web services programming. It processes XML data files directly with no extra parser required. Data, logic, and presentation are all represented and processed in a consistent manner using Water code. Water is designed to support both object-oriented programming and functional programming. See [www.waterlanguage.org](http://www.waterlanguage.org) for full language details.

### Generality – Water

Rating: High. Flexibility: the most flexible of method calling including

- optional parameters with default values
- passing of arguments by keyword or position
- unlimited number of parameters possible via *rest* parameter
- content parameter for particularly large arguments
- execution kind settable per parameter, allowing different kinds of evaluation per argument including *none*.

Extensibility: the most extensible of the languages due to:

- addition of instance variables to already created instances
- addition of methods to existing classes without a recompilation step requiring the source code

Composability

- Fully object-oriented
- Any object can be treated as a class or an instance

- On the fly changing of the parent of an object to have dynamic inheritance
- Multiple inheritance
- Designed for making special purpose languages with a very general base

### **Functionality – Water**

Rating: Medium. Water, as a relatively new language, does not yet have a large library base. It can call any Java method, thus giving it Java's functionality with essentially Water's syntax with a bit of overhead. The Water language comes with all HTML tags built in as native Water classes. It is very easy to create Web servers and services in Water, just as it is easy to consume and produce XML.

Water has a *capability-based* security model that is simple yet effective in keeping two programs running on one JVM separate from each other.

### **Compatibility – Water**

Rating: High. Runs on the Java Virtual Machine and can call and be called by Java easily. More compatible with XML and XHTML than any other programming language.

### **Ease of Use – Water**

Rating: High. IDE permits very quick edit-run cycle of segments of code as well as full programs. ConciseXML syntax gives the developer a lot of options to pass arguments by position, keyword, or letting them default. Ending tags optional.

### **Performance – Water**

Rating: Low. Slow in raw execution as it is an interpreter written on top of Java. However, effective layering of Web services in tiers with Water can cut down on round trips between client and server saving time on distributed applications. Water permits code and data to be cached near the user for high responsiveness.

## **Other Languages**

An exhaustive analysis of many other languages is not warranted. This study includes just a few points about the languages below, specifically how they compare to the best of the above languages.

### **Ada**

Very strong typing. ADA is good for writing large programs. It has been rejected by programmers because it is not easy to use.

### **Curl**

Good language from MIT and a spin-off company. Uses rather conventional programming language syntax, so Curl is poor for dealing with XML and HTML. Has nice graphics and interesting client-side processing. Curl's class-instance object system is more complex and less flexible than Water's.

### **Perl**

Lots of functionality for text processing, but has been called a write-only language for the difficulty it presents for a human to decode the terse, idiosyncratic, complex code. Like many difficult-to-learn languages, it has strong adherents.

## PHP

A favorite for Web scripting among ad-hoc programmers. It offers are lots of useful libraries. PHP is not easy to use with XML or HTML.

## Python

Some ex-Lisp programmers like Python, but it has much more complex syntax than Lisp. It uses both `foo(a, b)` style method calls as well as infix syntax. Amount of white space for indenting actually affects semantics, unlike all other popular languages. This is a double-edged sword. It has a less flexible object system and method calling than Water.

## Ruby

Some programmers like Ruby more than Python, for example. It is fully object-oriented and has some flexible features, though not as flexible as Water. Ruby is not XML-based and so deals with XML and XHTML much less smoothly than Water. It has a single inheritance object system. It does not incorporate security features.

## 3. PRESENTATION

### *Presentation Criteria*

Each simulation will generally have its own presentation capability. The framework will not need to replace each simulator-specific presentation. However, a comprehensive framework should provide a single way to display summary or aggregate data derived from multiple simulators.

### Functionality

Text (including fonts and international character sets), 2D graphics, 3D graphics, user input, pictures, audio, video may all come into play. The functionality for summarizing text and 2D graphics required in a simulation framework is not difficult to provide. High functionality is more important in 3D graphics required in individual high-performance simulations.

### Compatibility

HTML is the overwhelmingly popular display mechanism for formatted text and limited graphics. Next, there is XML representation for graphics in SVG and X3D. These are compatible only in the sense that they use XML and can thus share a parser. Since XHTML uses an XML format, it rounds out the 1D, 2D and 3D graphics formats. Java's myriad window systems provide generality at the expense of complexity in programming. JView offers yet another Java window system that is particularly good for 3D graphics. JView is compatible with Java.

### Ease of Use

To evaluate the ease of use criteria you have to decide whether *dynamism* in graphics is important. Dynamism means configuring actions or behavior to make the graphics interactive. The Java window systems are fairly difficult to program for static graphics and very difficult for connecting dynamism, though a very wide range of options are possible with the Java solutions.

### Performance (compute power needed to render graphics)

Performance is highly dependent on the amount of interactivity and responsiveness needed in a simulation. High resolution and fast response are required for flight simulation. Generally high performance is not required for a framework that configures the simulations and compiles the data generated by the simulations.

## **Presentation Criteria Summary**

For framework graphics, the main point is that 3D, highly interactive graphics are not necessary. However, good text supported by good 2D graphics will be important in summarizing quantitative data. Presenting 2D data as 3D data makes the display prettier but usually detracts from the results produced by the simulation. Generally, the user does not need real-time graphics, but rather the more typical ability to select items from menus, choose data to be presented using check boxes, and filtering data by typing data in text boxes.

## **Presentation Candidates**

### **HTML/XHTML**

HTML as originally designed has a rather non-uniform syntax. XHTML have been developed to improve this situation and should be used for all future presentations instead of HTML. XHTML and HTML have the same functionality, but XHTML is easier for tools to manipulate and has more consistent rendering across browsers. So for the purposes of this study, proper XHTML is assumed rather than the original HTML.

#### **Functionality- HTML/XHTML**

Rating: Medium. HTML/XHTML is good for text formatting. It is good for 2D tables, although it is difficult to import data into 2D tables without the right tools. It is satisfactory for displaying simple 2D (bitmapped) images and arranging them on a page, though not as good for compositing them. Real 2D and 3D graphics are not possible in HTML. Forms containing menus, radio buttons, check boxes and text entry boxes are all part of HTML. Sliders, progress bars, bar charts and graphs are not.

#### **Compatibility - HTML/XHTML**

Rating: High. HTML/XHTML is the best for Web distribution. Browsers that display HTML are ubiquitous, making broad distribution trivial. HTML, at several billion pages, is the most prevalent of any machine-readable file format.

#### **Ease of Use – HTML/XHTML**

Rating: High. Millions of people have learned HTML. The text format is a bit cumbersome but very effective for text markup. Attempting to do 2D graphics with HTML is very difficult and meets with only limited success.

#### **Performance – HTML/XHTML**

Rating: Low. On modern personal computers, with a high-speed internet connection, performance for rendering complex pages of text and images is good. Performance is not good for the much more compute-intensive, 2D and 3D graphics.

## **SVG (2D graphics in XML format)**

### **Functionality – SVG**

Rating: Medium. SVG is superior for drawing colored lines and polygons as well as some sophisticated integration of text into arbitrary shaped regions. SVG itself does not contain window system elements, but pages can be constructed that have regions of SVG and regions of HTML. SVG 1.2, currently under development, integrates audio and video into the 2D graphics core of SVG. Considerations are being made to integrate XFORMS for windowing system-like elements, though SVG itself probably will not contain them. See <http://www.w3.org/Graphics/SVG/>.

### Compatibility – SVG

Rating: Medium. SVG is an XML format, so it is syntactically compatible with XML parsers and editors. However, to view an SVG document you need a Web browser plug-in so viewing is not as easy as with HTML. The plug-ins are downloadable so deploying SVG applications is fairly easy.

### Ease Of Use – SVG

Rating: Medium. Writing an SVG document is like writing any XML document. Integration of code is tricky since SVG does not provide for real dynamism. Many SVG developers use JavaScript to provide the procedural element.

### Performance – SVG

Rating: High. Since SVG drawings can be simply line drawings, they are much faster to download and render than bitmapped images.

## X3D (VRML)

X3D is the reincarnation of VRML97 in XML syntax. VRML is designed for presenting realistic or low-resolution *virtual reality*; it is very useful for building an environment and creating *walk-throughs*.

### Functionality – X3D

Rating: Medium. X3D is good for 3D scene description and creating a *first person* experience. It provides hooks for interactive components, but real behavior must be done in a scripting language such as JavaScript.

### Compatibility – X3D

Rating: High. VRML is the most common format for virtual reality scene description. X3D will supplant VRML as it becomes more popular (X3D is quite new).

Since X3D uses XML syntax, XML parsers and authoring tools are feasible for creating and using X3D documents.

### Ease of use – X3D

Rating: Medium. X3D is as easy as XML for detailed editing. However, complex scenes will should be designed in a graphical environment.

### Performance – X3D

Rating: High. X3D renders 3-d scences well without a special purposed graphics processor..

### X3D and XML

X3D is promoted by the Web3D Consortium. Members of the Consortium that were present at the WebSim Workshop, in October, 2003, were adamant proponents of XML. This is surprising given the problems of representing X3D scenes in XML. . Understanding these problems is critical to understanding both how far XML has come and how far it has to go to achieve its goals.

The X3D CDROM handed out at the 2003 WebSim Workshop had a “Hello World” scene for illustrative purposes. The following is the core XML for that scene.

```
<Shape>
  <Text string="&quot;Hello&quot; &quot;world!&quot;"/>
  <Appearance>
    <Material diffuseColor="0.1 0.5 1"/>
  </Appearance>
</Shape>
```

There are four striking difficulties in this example. First, the value of the Hello world string is roughly three times longer than "Hello World." " " have been placed around each word. In most programming languages, including C and Java, this would be accomplished with the string "\Hello\ \world\ ". This expression is significantly more concise and readable than the XML character "entities" of " ".

Second, the representation of the value of the "diffuseColor" attribute is a string of three numbers. These numbers represent the red, green and blue values of the color they are describing. In order to use this color inside of a program, you must write a parser that:

- separates the three numbers into three strings
- trims white space
- converts each substring of digits and dots into a floating point number

That is not such a hard parser to write. However, the primary purpose of XML and X3D is to provide a standard syntax that can be parsed by standard parsers. XML and X3D fail to achieve this primary purpose; in fact, special purpose parsers are required.

Third, the string "0.1 0.5 1" carries no actual information about the type of object intended by it. Is it simply a vector of numbers? Probably not; even though it is a vector of numbers, it is not well described. Suppose you really wanted a string instead of a vector of numbers. What you probably want is a COLOR object. A human can infer that information from the name of the attribute (diffuseColor), but that is not much help to a machine. XML is supposed to be interpretable by a human as well as a machine. The schema for X3D undoubtedly describes what that string of three numbers is really supposed to be, though even if it does, you have to look in a reference document, for a long time, to find the real meaning of that string.

Fourth, one can guess that the three numbers are intended to be red, green and blue values, but suppose they are actually hue, saturation and value numbers. Again the schema or whatever program interprets this file must include the information on how to treat that vector. If the authors of X3D want the capability to express their colors in either R,G,B format or H,S,V format, they must put an indication of that fact into the string. However, this makes even more work for a special parser. In fact, the special parser should not be needed in the first place.

Now to be fair, XML could be used to structure the numbers better, as shown in the following example.

```
<Material>
  <color red="0.1" green="0.5" blue="1"/>
</Material>
```

But, suppose you want not just a diffuseColor but a shadowColor as well. The following gives us two colors, but there is no easy way to identify which color is which.

```
<Material>
  <color red="0.1" green="0.5" blue="1"/>
  <color red="0.2" green="0.6" blue="0.8"/>
</Material>
```

You could use the following to distinguish between them.

```
<Material>
  <diffuseColor red="0.1" green="0.5" blue="1"/>
  <shadowColor red="0.2" green="0.6" blue="0.8"/>
```

```
</Material>
```

However, you would then be using the tag name `diffuseColor` as a part name, not as a type, whereas, you use `Material` in `<Material .../>` to mean type.

So, in this example, a tag name is used in one context as a type and in another as a part name. This would make it very confusing since the primary function of a tag name is itself ambiguous for a very basic description. Perhaps that is why the X3D designers chose the string containing three numbers and forced every programmer to write their own parser, thus defeating the *reason for being* based on XML.

This is a simple example, but note if these kinds of problems appear in this simple “Hello World,” example, multiply them by a thousand-fold for complex data and you have an extremely complicated and confusing situation.

Although ConciseXML, as a superset of XML, can represent X3D in exactly the same way, the following is a more reasonable approach that avoids the above problems and is less verbose.

```
<Shape Text="\Hello\ \World\"  
  Appearance=<Material diffuseColor=<color red=0.1 green=0.5 blue=1/>/>  
>
```

This encodes the same information as X3D and is

- less ambiguous
- more extensible/composable
- less verbose

The X3D designers did as good a job as possible given the restrictions of XML. With ConciseXML, they would not have had to impose these restrictions.

## Java (including AWT, Swing, SWT, Java 3D, Java 2D)

### Functionality – Java

Rating: High. Java provides the skilled programmer a wide range of capability and the power to use high-level abstractions. It is not particularly good at rendering formatted text, however.

### Compatibility – Java

Rating: Medium. Java is quite incompatible with XML. However, Java has the advantage of being compatible with the large number of Java programs available.

### Ease of use – Java

Rating: Low. A Java window system expert and an experienced teacher of Java Swing has said that it takes a good two years to be comfortable with the Java language. Calling ease of use *low* for Java window systems may be an understatement.

### Performance – Java

Rating: Medium. Performance is better than SVG and HTML. Java can also take advantage of enhancements, native to the operating system, to speed up rendering.

## **JView**

This three-dimensional graphics package was written at Air Force Research Laboratory to facilitate simulations written in Java. It uses OpenGL as its underlying graphics engine.

### **Functionality – JView**

Rating: Medium. Use of Jview is restricted to 3D graphics. It provides the facility for users to select an object, but does not have 2D input features like check boxes and menus.

### **Compatibility – JView**

Rating: Low. Easy interface to Java as it is written in Java. No provisions for HTML or XML.

### **Ease of use – JView**

Rating: Low. For a Java library it is quite good, but the JView interface is hampered by Java's restrictive method calling.

Clear Methods was able to demonstrate interoperability between Water and JView quickly. Having a ConciseXML representation of Jview is expected to make presentation easier for Web applications.

### **Performance – JView**

Rating: High.

## **4. DATA REPRESENTATION**

A common data representation facilitates transferring static data among simulations. It is fundamental to deriving the greatest benefit of combining various data sources. Even if the overall system is capable of accepting data of different formats, it is still important to have an internally consistent and comprehensive data format to reduce complexity. There are two important considerations in dealing with this problem:

- reducing the number of translations necessary
- providing a common format for manipulation. (This is discussed further in the integration section of this study.)

### ***Data Representation Criteria***

#### **Functionality**

The selected technology must be able to represent the kind of data needed across numerous domain including number, string, Boolean, character, vector/array, record/structure, and hierarchical data.

#### **Compatibility with Existing Practice**

New systems will need to interface to existing systems. This means either connecting to them natively or writing code to convert legacy output to XML; thereby creating consistent definitions and reducing ambiguity

#### **Conciseness**

Disk space, processing power, and even transmission bandwidth have generally become inexpensive. Still, when these elements are taken together, the value of conciseness grows. More importantly, conciseness is critical to human readability. One can, of course, be too concise for humans, but being overly verbose gets in the way of rapidly understanding a data document.

## Data Ambiguity

Ambiguous data, especially data whose degree of ambiguity itself is uncertain, is practically worthless. Many data ambiguity issues are resolved, not in the data itself, but in processes that interpret the data. However, the processes can be error prone and separate a key part of the semantics of the data from the data itself. This inhibits data reuse because a new *interpreter* must be written for each use of the data. Furthermore, a human trying to understand the data has to learn how to read the data for each interpreter just as software would by designing rules to resolve ambiguity.

## Ease Of Use

Ease of data use includes several criteria to assess the ability of both machine and human to interpret data. If a data representation does not have the native functionality for encoding a particular kind of data, one will have to be invented even if it is cumbersome. If a representation differs greatly from a known practice, more education will be required. Verbose representations take more time and space to communicate, making it difficult to grasp the meaning of the full document. Finally, ambiguity leaves the data open to misinterpretation; Extensive training is required so that data is not misinterpreted.

## Data Criteria Summary

In this study, functionality has been rated as the most important criterion. A format that cannot represent all the required data is not acceptable. The next most important criterion is non-ambiguity. Ambiguous data cannot be processed successfully, therefore, it is of little value. The importance of conciseness is rated lower, not because it is unimportant, but because it is less difficult to manage. The ready availability of high speed computers, networks and large, inexpensive disk drives have rendered conciseness less important than it used to be.

## Data Representation Candidates

Given the variety of real world data, there will most likely always be numerous data formats. A system able to process only one format will be overly brittle. One way to deal with the problem of multiple data formats is to structure the system around a primary data format for internal use, and convert all other formats into that primary format. Each of the following candidates could serve both as a raw data input format as well as potentially a common internal format.

ASN.1 (Abstract Syntax Notation One) and EDI (Electronic Data Interchange) are not evaluated in detail here because they are largely obsolete standards.

ASN.1 was widely touted 20 years ago for concise, binary interchange of data. XML is overshadowing ASN.1 as the preferred universal data format. Two years ago a consortium was formed to promote it as a concise XML transformation for transmission. However, it does not address the issue of dynamic data.

EDI X12 is being replaced by XML representations in the healthcare industry through HL7 standards and will be reflected in evolving HIPAA mandates. EDI suffered from significant expense for direct links or Value Added Network (VAN) services to exchange EDI documents. Perhaps most importantly, EDI permitted proprietary pair-wise data formats. Note that XML also has this weakness as it uses arbitrary representation for nesting of objects.

## Comma Separated Values (CSV)

A CSV file stores *instances* of a particular record type. The name of each field of an instance can be described in the header and the values for each field can be given. Typically, the fields are separated from each other with a comma and the instances are separated from each other with a new line.

CSV is simple. It uses ASCII characters that can be edited with a regular text editor and easily read by a human. If there are a lot of fields or a lot of instances, it can get harder for a human to keep track of the instances and understand the meaning of a particular field. CSV is satisfactory for small, flat, non-self-referential data. Because of its common use, a means to read this format is essential.

### **Functionality – CSV**

Rating: Low. CSV does not scale well. It is poor for hierarchical and self-referential data. It provides only two levels: an *instance* and a *field*, but does not provide structure within a field.

### **Compatibility – CSV**

Rating: Medium. CSV files can be generated from Excel spreadsheets as well as edited by hand. CSV files of simple data are easily created by programs. It would be very hard to represent more complex data such as XML-like data in CSV files. Thus, the low score for compatibility.

### **Conciseness – CSV**

Rating: Medium. Conciseness is quite good for the kind of data that this format can represent.

### **Non-Ambiguity – CSV**

Rating: High. CSV is quite unambiguous for the kind of data that it can represent.

### **Ease of Use – CSV**

Rating: High. CSV is easy to use unless you have to make field values that are more structured than strings. There are, of course, conventions for this, but since they are not standard, they cannot be depended on. CSV gets a mediocre score for ease of use.

### **CSV Summary**

Due to low functionality, CSV is not a serious contender for a common data representation.

## **XML (including DTD and XML Schema)**

XML. XML is used more and more in distributed systems because it can describe data in a consistent format. Unfortunately, XML does not completely meet its own goal of providing a consistent format for static data. However, with some additional work, it can be used successfully. XML was not designed with *dynamic data* (variable data) in mind and therefore cannot be used to represent that type of data.

### **Functionality – XML**

Rating: Medium. Functionality for XML is medium to high. XML can represent hierarchical data. It is poor at representing non-string data, though with the addition of a schema and an interpreter that reads both the data and the schema, more complex data can be represented. It is poor at handling self-referential data and, as noted above, cannot represent dynamic data.

### **Compatibility – XML**

Rating: High. XML is highly compatible. XML is a relatively new format, having been in common use for only several years, but already it has been used to encode a tremendous amount of data. Most *new* data formats being proposed for standardization have an XML representation. XML can easily represent the simpler formats of CSV and EDI.

### **Conciseness – XML**

Rating: Low. XML is not concise. XML requires *ending tags* that duplicate the beginning tag of an element (think *instance*). In addition, each *attribute* (think *field*) of an instance also has to be named. Thus, for many instances of the same class, XML is particularly inefficient. CSV is much better than XML in this regard.

### **Non-Ambiguity – XML**

Rating: Low. This is a complex issue that the designers of XML did not fully take into account. See the Clear Methods paper, “*The Trouble with XML,*” ([http://www.waterlanguage.org/XML\\_trouble.html](http://www.waterlanguage.org/XML_trouble.html)) for a full explanation. Briefly, the problem is due to XML’s restricted representation of a field value, which must be a quoted string that does not contain angle brackets. Thus, you cannot *nest* complex structures in attribute values. In this regard, it is similar to CSV.

XML has a *content* area of an element that can take nested values. However, you cannot *name* these parts as you can attributes. You have to decide whether a given part is to be named and restricted to a string value or should remain unnamed and allowed to be a complex structure. Very often neither choice is acceptable. In order to work around these restrictions, XML designers sometimes embed a name within the *value* of the field itself. This has numerous drawbacks. Designers might use the tag name as a part name. But, that means that sometimes with a tag like `<foo .../>`, `foo` is interpreted as a type, and other times it is interpreted as a part name. Designers do develop conventions, but the inherent ambiguity of XML does not lend itself well to unambiguously named parts with complex values, so there are many different conventions which cause ambiguity in themselves.

### **Ease of Use – XML**

Rating: Medium. XML is pure text so it can be edited with a text editor; it lends itself to more structured editors. A human can read XML relatively easily because it must include attribute names and ending tags. However, including those elements, leads to verbosity, which ultimately decreases the readability of certain kinds of data, especially very regular data.

### **XML Summary**

XML has become the de facto standard for data representation. However, XML cannot represent the most important kind of computer data: dynamic data. XML can represent the kind of data used by EDI and CSV well, though it cannot contain code. The recent adoption of XHTML makes XML also good for presentation, especially over the Web.

### **ConciseXML**

ConciseXML is a superset of XML and is the syntax used by the Water programming language (see Code section for description of the Water language). It was designed to overcome the shortcomings of XML yet still be able to treat standard XML as an acceptable data representation. Automatic conversion between ConciseXML and XML 1.0 is built into the Steam XML Integrated Development Environment (IDE). See [www.ConciseXML.org](http://www.ConciseXML.org) for additional information.

### **Functionality – ConciseXML**

Rating: High. Concise XML provides for nesting complex named parts unambiguously. Self-references, as well as very complex references, can be made through the dynamism of Concise XML. Flexible code can be easily represented in ConciseXML, decreasing or eliminating the need to embed JavaScript, Perl or other special-purpose languages as a large string inside of an XML document that cannot be parsed easily.

### **Compatibility – ConciseXML**

Rating: Medium. Compatibility is a more complex issue for ConciseXML, than for other candidates. Because ConciseXML is a superset of XML, its use can be restricted to simply XML 1.0. Without following that restriction you lose direct compatibility, though automated conversion between the concise form of ConciseXML and XML 1.0 is easy to invoke. Still, ConciseXML must be given lower marks than XML in compatibility because, due to its newness, less data has been represented in ConciseXML to date. But, a direct comparison between ConciseXML and XML is not so simple because in practice XML tends to be relatively incompatible with itself. This is

because, given the absence of a *natural* or obviously unique way to code in XML, designers will likely code complex nested data in different ways. That is not the case in ConciseXML.

### **Conciseness – ConciseXML**

Rating: Medium. Despite its name, ConciseXML is not always concise. ConciseXML has the flexibility to be concise or explicit with automated conversion between these two extremes. You can mix the concise forms and verbose forms in the same document or even in the same *element*, because, in ConciseXML, (unlike XML 1.0), attribute names and ending tags are optional. Often, a ConciseXML designer will use the short form for small, common instances and the longer form for larger, more complex instances. Thus, the syntax can be tailored to match the needs of the data rather than the data being forced into a single rigid syntax.

### **Data Ambiguity – ConciseXML**

Rating: High. ConciseXML is unambiguous especially for complex named-field values, which EDI, CVS and XML were not designed for.

### **Ease of Use – ConciseXML**

Rating:Medium. ConciseXML has a more complex syntax than the other candidates, so on the surface, it appears harder to use. However, it also has more functionality. If you attempt to use the other formats for data structures that they were not designed for, then special work-arounds must be employed to overcome the limitations of those syntaxes. These work-arounds raise the complexity of the document and necessarily need outside of document knowledge for interpretation. The most flexible of the other candidates, XML, has a hard time making the answers to such basic data representation issues as “what is the class of this instance?” and “what is the name of this part?” obvious For the limited kinds of data that EDI, CSV and XML are good for, those formats are easier to use. Unfortunately a large amount of data just does not fit those types of data.

### **ConciseXML Summary**

Based on Clear Methods’ experience in programming simulators, the company anticipates the need for functionality well beyond EDI, CSV and XML. Even if the increased functionality is not immediately required, it would not be advisable to adopt a standard that can not accommodate named, nested complex parts unambiguously. For that ConciseXML stands alone compared to the other candidates. To be sure, there are work-arounds, which enable the use of the other data standards, but they involve storing meta information about the document outside of the document itself, usually in code. This greatly decreases modularity and raises complexity, and causes issues with reuse and ease of use. ConciseXML’s weakest criterion is in compatibility. That is addressed by including automatic converters between the concise form of ConciseXML and XML 1.0 in the Water language.

## Conclusion and Summary

Per the attached Excel spreadsheets, these are the results of this evaluation. The highest scoring technology in each category is as follows.

- |                              |                   |
|------------------------------|-------------------|
| ▪ <b>Integration</b>         | <b>Water</b>      |
| ▪ <b>Code</b>                | <b>Water</b>      |
| ▪ <b>Presentation</b>        | <b>XHTML/SVG</b>  |
| ▪ <b>Data Representation</b> | <b>ConciseXML</b> |

Water uses ConciseXML (which includes XML 1.0) in its syntax. Water includes all of XHTML for presentation. SVG is an XML representation for 2D graphics and X3D is an XML representation for 3D graphics. Since Water uses XML syntax, it is straightforward to integrate SVG as well as X3D. Clear Methods has completed a proof of concept of this integration for both SVG and X3D.

Although not discussed in this study, another presentation dimension is speech using VoiceXML (VXML). It is worth noting that this could likewise be easily integrated into Water to accommodate this critical dimension of training, mission planning and post-exercise review. .

The capability to compose legacy and new simulation resources for new scenarios requires complex interoperation among data, presentation and code. By using a single syntax, semantics and object model for data, presentation and code, it becomes possible to greatly simplify those complex interactions. Water is the only language that simultaneously provides this single, simplifying, comprehensive capability. This is *agility*.

By using Water to flexibly modularize a solution, normalize the disparate aggregate data, and natively communicate across the Web using XML, JSB can simplify future development while integrating older technologies.

---

## Appendix – Spreadsheets

The Excel source files are also available.

<b>Simulation Composability Framework</b>						
<b>Integrated Solution Spreadsheet</b>						
		Weighted means the raw score multiplied by the importance of the given criteria. Score equals the sum of the criteria divided by the number of criteria. Max possible value is 1.0				
<b>Criteria</b>	<b>3</b>	<b>Code ↔ Data</b>	<b>Code ↔ Presentation</b>	<b>Ease of Use</b>	<b>Raw Score</b>	<b>Weighted Score</b>
Criteria Importance		1.00	0.50	0.40		
<b>Candidates</b>						
<b>Multi-language</b>	Raw	0.30	0.30	0.20	0.27	
	Weighted	0.30	0.15	0.08		0.18
<b>UML/MDA</b>	Raw	0.20	0.20	0.10	0.17	
	Weighted	0.20	0.10	0.04		0.11
<b>Water</b>	Raw	1.00	1.00	1.00	1.00	
	Weighted	1.00	0.50	0.40		0.63
Criteria include:		data transformation dynamic data	user interface browsers	training product management Tower of Babel issue		

Simulation Composability Framework								
Code Spreadsheet								
Weighted means the raw score multiplied by the importance of the given criteria. Score equals the sum of the criteria divided by the number of criteria. Max possible value is 1.0								
Criteria	5	Generality	Functionality	Compatibility	Ease of Use	Performance	Raw Score	Weighted Score
Criteria Importance		1.00	0.70	0.60	0.60	0.40		
<b>Candidates</b>								
<b>C,C++</b>	Raw	0.20	0.80	0.50	0.10	1.00	0.52	
	Weighted	0.20	0.56	0.30	0.06	0.40		0.30
<b>C#</b>	Raw	0.30	0.50	0.50	0.50	0.80	0.52	
	Weighted	0.30	0.35	0.30	0.30	0.32		0.31
<b>Java</b>	Raw	0.30	0.90	0.50	0.50	0.30	0.50	
	Weighted	0.30	0.63	0.30	0.30	0.12		0.33
<b>Javascript</b>	Raw	0.30	0.50	0.50	0.50	0.20	0.40	
	Weighted	0.30	0.35	0.30	0.30	0.08		0.27
<b>Common Lisp</b>	Raw	0.80	0.50	0.50	0.50	0.90	0.64	
	Weighted	0.80	0.35	0.30	0.30	0.36		0.42
<b>Water</b>	Raw	1.00	0.80	0.80	0.90	0.20	0.74	
	Weighted	1.00	0.56	0.48	0.54	0.08		0.53
Criteria includes:		flexibility extensibility composability	capability security	interoperability Web services	learning specification testing detailed documentation	runtime speed scalability distributed computing		

Simulation Composability Framework							
Presentation Spreadsheet							
		Weighted means the raw score multiplied by the importance of the given criteria. Score equals the sum of the criteria divided by the number of criteria. Max possible value is 1.0					
Criteria	4	Functionality	Compatibility	Ease of Use	Performance	Raw Score	Weighted Score
Criteria Importance		1.00	0.50	0.40	0.40		
<b>Candidates</b>							
<b>XHTML</b>	Raw	0.50	0.80	0.90	0.20	0.60	
	Weighted	0.50	0.40	0.36	0.08		0.34
<b>SVG</b>	Raw	0.40	0.30	0.60	0.20	0.38	
	Weighted	0.40	0.15	0.24	0.08		0.22
<b>X3D</b>	Raw	0.30	0.30	0.50	0.70	0.45	
	Weighted	0.30	0.15	0.20	0.28		0.23
<b>Java</b>	Raw	0.80	0.50	0.10	0.80	0.55	
	Weighted	0.80	0.25	0.04	0.32		0.35
<b>JView</b>	Raw	0.30	0.50	0.20	0.80	0.45	
	Weighted	0.30	0.25	0.08	0.32		0.24
<b>XHTML + SVG</b>	Raw	0.90	0.60	0.70	0.20	0.60	
	Weighted	0.90	0.30	0.28	0.08		0.39
Criteria include:					computer power needed to render		

<b>Simulation Composability Framework</b>								
<b>Data Representation Spreadsheet</b>								
Weighted means the raw score multiplied by the importance of the given criterion. Score equals the sum of the criteria divided by the number of criteria. Max possible value is 1.0								
<b>Criteria</b>	<b>5</b>	<b>Functionality</b>	<b>Compatibility</b>	<b>Conciseness</b>	<b>Non-Ambiguity</b>	<b>Ease of Use</b>	<b>Raw Score</b>	<b>Weighted Score</b>
Criteria Importance		1.00	0.50	0.20	0.60	0.40		
<b>Candidates</b>								
<b>EDI</b>	Raw	0.20	0.50	0.80	1.00	0.20	0.54	
	Weighted	0.20	0.25	0.16	0.60	0.08		<b>0.26</b>
<b>CSV</b>	Raw	0.20	0.50	1.00	1.00	0.70	0.68	
	Weighted	0.20	0.25	0.20	0.60	0.28		<b>0.31</b>
<b>XML (DTD &amp; XML Schema)</b>	Raw	0.70	1.00	0.20	0.20	0.70	0.56	
	Weighted	0.70	0.50	0.04	0.12	0.28		<b>0.33</b>
<b>Concise XML</b>	Raw	1.00	0.50	0.50	1.00	0.50	0.70	
	Weighted	1.00	0.25	0.10	0.60	0.20		<b>0.43</b>
Criteria includes:		security	interoperability distributability			clarity		

## Appendix – Further References

- Water Language; <http://www.waterlanguage.org/about.html>
- **Water: Simplified Web Services and XML Programming**, by Mike Plusch; John Wiley & Sons, publisher; 2003
- Steam XML IDE; <http://www.clearmethods.com/products.htm>
- Concise XML syntax <http://www.concisexml.org/>
- Article: **Water's Fine for Extreme Programming**; <http://www.waterlanguage.org/XP.html>
- Article: **The Trouble With 2 + 3**; <http://www.waterlanguage.org/XMLtrouble2plus3.html>
- Article: **Water Security Architecture**; <http://www.waterlanguage.org/security.html>
- Article: Petre, Marian; **Why Looking Isn't Always Seeing**; Communications of the ACM, June 1995
- **Shaping Evolution of the Global Information Grid (GIG)**, Dawn Meyerriecks, Chief Technology Officer, Defense Information Systems Agency (CTO, DISA), presented November 29, 2003, Web-Enabled Modeling and Simulation Workshop, Reston, VA. See: <http://www.websim.net> or <http://www.vmasc.odu.edu/wsc/pres/meyerriecks.ppt>
- ***Static and Dynamic Modeling and Analysis of Architectures*** Roshanak Roshandel, University of Southern California See: <http://sunset.usc.edu/gsaw/gsaw2002/s9/roshandel.pdf>