HIGHMAST™ VERSUS "WORKBENCH" STYLE SIMULATION TOOLS

Peter C. Bosch

Highpoint Software Systems, LLC
S42 W27451 Oak Grove Lane
Waukesha, WI, 53189, U.S.A.

Introduction

This assessment is intended to provide a general survey of the strengths and weaknesses of approaching a simulation problem using a workbench-style tool such as Arena or Extend versus approaching that same problem with source code, a simulation server and class libraries (which, along with some quick-start wizards and design tools, characterizes HighMAST™).

The two approaches under consideration share some similarities.

- Both approaches require at least one developer who understands simulation. Only the most trivial simulations can be built with any degree of assurance by a simulation novice.
- Both approaches require understanding of the tools being used.
- Both entail programming of a sort.
- At a cost, each provides some entry into the other's domain. Source code can be used to create drag-and-drop tools, and most drag-and-drop tools permit some sort of direct-to-source-code access.

The remainder of this document will be focused on describing and explaining the differences between the approaches. This comparison of platforms is a survey of factors – no one factor is absolute, and each factor can be addressed in either environment, with varying degrees of difficulty.

Executive Summary

If you are building a relatively simple, standalone simulation to provide immediate and one-time answers to a specific question and the user base is familiar with the environment, a workbench tool such as Arena or Extend is probably the best fit. As the need progresses from "an answer" to "an application", the number and specificity of the requirements increases, and the integration needs and uniqueness of the desired solution grows, source-based tools such as HighMAST™ will be found to provide a better fit in the areas of price, performance, longevity and satisfaction.

What is HighMAST™?

HighMAST™ is a simulation framework that is written from the ground up as a modern object-oriented platform for the construction of simulation needs. It builds on Microsoft's .Net platform, to enable power, productivity and integration capabilities far beyond many other platforms in the simulation software world.

HighMAST™ can be used to create simulation infrastructure ranging from standalone single-decision applications to domain-level analysis tools. It can readily support usage models from the single analyst at his desk to web-deployed enterprise applications, and can be used to create embeddable services for decision-support tools ranging from individuals' spreadsheets to corporate IT applications.

With a growing library of SimStart™ wizards that plug in to Microsoft's Visual Studio, you can use HighMAST™ to achieve the quick-start that a workbench solution offers, along with the power and flexibility that a source-based tool presents.

What is a "Workbench"?

A "Workbench" Tool is a tool that abstracts the programming of a simulation solution into a primarily graphical exercise, with icons representing distributions, transformations, delays, sources & sinks, and related constructs. Simulation workbenches usually provide a flow-oriented environment, and typically bundle an executive, libraries of primitives, graphical user interface functionality and basic export/import & integration mechanisms. Most have additional sets of

blocks (primitives) that can be purchased, which provide primitives for specific domains such as supply chain and packaging simulation. Examples of such tools are Rockwell's Arena and ImagineThat's Extend.

Strategic Comparison

As with most source-based tools, HighMAST™ exposes more degrees of freedom in its implementation than a typical workbench tool, since it permits any code to be written on top of, and within, the provided mechanisms. Source-based tools are therefore usually more powerful than workbench-style tools, and allow the developer to tailor final application features, look-and-feel, and algorithms to the needs of the customer. However, they require the developer to understand a programming language, and their power can encourage the application builder to add features that he would not otherwise, sometimes resulting in a more full-featured application, and consequently a longer development cycle.

Workbench tools, being graphical, are typically easier to learn up to an intermediate level than a source-based tool, but often require equivalent effort to the source-based tool to become an expert. Workbench tools can become more difficult to master, if the expert user wishes to "drill down" to the workbench's underlying source code model. This often becomes necessary to satisfy needs for custom behaviors & analytical algorithms, application configuration, security, run-time input & output, and a wide range of other esoteric requirements.

Size Matters

There are two types of size that are relevant to this comparison. The first is project size, and the other is simulation size, and each has a rather strong impact on the choice of approaches.

It is important for the reader to keep in mind while reviewing these factors, that the less concrete a project specification is, the more likely the project is to be larger and deeper than expected.

Project Size

Project size is the scope of the development effort.

In small projects of up to perhaps five person-days' development, it is presumed that the domain expert is going to act as the modeler. Independent of complexity (which is treated separately in this paper), in these projects, the workbench has an advantage if the domain expert is familiar with it.

In a larger project, though, a wide variety of size-related factors can drive the advantage to the source-code tool. These could be factors such as having multiple team members working on the project or needing project configuration control, requirements management or a formal testing regimen.

The software industry has been working for decades on easing the difficulty of execution of non-trivial software projects, and these solutions, far more often than not, are focused on source-code projects.

An important consideration as the size of a simulation effort grows, is that not all of the effort is high-end simulation work. Work must be done on data entry and access, user interface and dialog design, behavioral constraints, security, and many other areas. Simulation tools with arcane or proprietary languages make it likely that the developers doing this work will either be expensive, or unfamiliar with the language. With HighMAST™, the developers can be "regular" developers, since the languages used are mainstream, general and well supported. In fact, it is not uncommon in large efforts for only one or two of the developers to be familiar with simulation, and for the rest to be broader developers, graphical designers, testers, etc. In short, by undertaking development with a tool like HighMAST™ your project can be staffed cheaper, with a range of people appropriate to the jobs they are doing, and familiar with the tools they are using – resulting in higher quality work performed at a better price, and with less completion risk.

Simulation Size

Simulation size refers to the number of different concepts represented in the simulation. This can be affected by variations in the behavior of specific types, such as a simulation that represents a labor force that is comprised of people possessing a range of different types and quantities of skills, or vehicles with different sizes and shapes, perhaps imposing different requirements on drivers' rest times (e.g. an eighteen-wheeler versus a railcar).

Software engineering has, again, for decades, been working on the problem of allowing one set of general behaviors to pervade a set of object types, but to allow specific types within that set to provide their own specific behaviors. The benefits of this approach, (object-oriented design), are great, and almost all mainstream software tool vendors have made it their central architectural approach. While mainstream tools and languages such as those used in HighMAST™ make it relatively simple, it is difficult and messy to implement graphically in a workbench tool.

The broader an application, the more likely that a source code tool will provide a simpler means for defining and managing those concepts, especially related concepts with minor variations in their data or behavior.

Complexity

Esoteric requirements

Simulations become necessary when the relationships between real-world entities are numerous or non-trivial. The greater the number or complexity of the interrelationships in a model, the greater the likelihood that you will need to represent those relationships in source code. And while source code can be injected into most workbench tools, a source-based tool is the simplest environment in which to perform this specification. The more of your time you expect to spend "injecting" source code, the more you ought to consider using a source-based tool in the first place.

Esoteric Structure

Models can be required to incorporate entities that flow through the model and have their own behaviors (customers who have different likes, dislikes and experiential decision-making, products that fail in different ways), and these can be difficult to model as graphical structures that use standard flow tokens. Also, idiosyncrasies such as a requirement that labor situated in France cannot be induced to work overtime, and a constraint that a factory in the U.S. may have a hard limit on emissions, can drive a great deal more complexity into a graphical solution than a source-based solution.

Often, a model is dynamically structured from some information source. Preloading a distribution system with materials in transit, configuring a factory floor with equipment and states-of-repair based on data in an ERP system or performing predictive analysis on an order stream can be difficult, and if you're working with a graphical tool, that difficulty can be compounded.

Esoteric Algorithms

Cost, routing, packaging, and floor-space optimization algorithms are all examples of algorithms that are critical to the performance or suitability of a simulation application. Often, tweaking these algorithms is a central purpose of the application, or the stakeholders want to be able to let their operations staff create and try out new algorithms. With workbench tools, these algorithms are often data-driven so that the tweaking can be done from a generic dialog (e.g. queue-selection algorithms). When faced with such tasks, for reasons of performance, flexibility and simplicity, staff members responsible for the algorithm work are likely to be more receptive to working in code rather than in dialogs or icons. The better the tools for doing so, the more productive they are likely to be, and for creating and modifying algorithms, source-based tools are likely to be far superior to graphical ones.

Lifecycle

All other things being equal, a simulation that is needed to answer a single, one-time question tends to favor a workbench solution. If you only need to know how many packing machines to put on the floor in the production line in the Tijuana plant, perhaps Extend or Arena can provide a quick, inexpensive answer.

But if you're developing software that will become a corporate asset, (one that will be used throughout your company to determine optimum layout of a range of workstation types for manufacturing of your company's product line, including new products, custom pieces, and specific rework), then developing it in a source-based tool with an eye to enabling SQL Server, Excel, Visio, or SAP integration (for example) may well be the best choice.

Another thing to keep in mind is that successful software takes on a life of its own. Software architects have, for a long time, understood the importance of separating the user interface (the "View") from the underlying engine (the "Model"), because such engines are often used later to satisfy the needs of a related or a larger problem, in which the user interface requirements are entirely different. For example, a new requirement to make the engine support a web application, bulk data entry, digital dashboard, etc., throws far less chaos into a well-conceived Model/View architecture than a Workbench-style solution.

Integration

A key consideration when choosing platforms is the degree to which you will need to integrate other software into it, and even more importantly, the degree to which you will need to integrate simulation functionality into other systems.

Database

Most workbench solutions provide data import/export capability, and some provide the capability to do so during a simulation run. Some even provide custom databases optimized for performance. But if you need to configure a model based on data in a corporate database, provide daily data feeds into a transportation optimization simulation for operational scheduling, or otherwise make the simulation an automated part of a larger system, a source-based environment with mainstream data access libraries may more readily provide you with the flexibility you will need.

3P Applications

If you need to integrate with a third party application such as a CRM or ERP system, chances are you will

need to do so in source code. And it will probably need to be standards-based code such as Java or any of the COM or .Net languages. CRM and ERP integration solutions typically require a "Model" deployment that is 24x7, GUI-less, and server based. Most workbench tools do not have such capabilities. Several workbench tools support plug-ins, but at the point that you decide you need this, you have probably realized you need a solution that is at least partly source-based.

Graphical U/I

Most workbench tools provide rudimentary user interface and animation capabilities, and with some effort, a solution can get moderately sophisticated. However, with a source based tool, the full range of the development tools and capabilities of the underlying libraries are available to a moderately skilled developer. With HighMAST™, the full capabilities of .Net and the Visual Studio environment, from dialog and image design and manipulation to high-performance 2D and 3D game-quality animation are available to the application developer.

Evolution

As the results of the specified application are required to reflect more and more closely the behavior and results of the real world, more and more of the characteristics and idiosyncrasies of the real world will need to be represented in the model. This is almost always far simpler to accomplish in source code than in a graphical representation.

As the size of the simulation increases, and performance requirements tighten up, it is likely that optimizations will need to be made. These are usually easier to perform in source code.

In addition to evolution-over-time, many corporate applications evolve to support *simultaneous* dual purposes. In our experience, it is often the case that at the same point in the life of a simulation application, some users will require high fidelity and narrow scope, and others will require higher performance and a much broader scope. An example of this would be a transportation optimization application that helped dispatchers create their daily routing assignments, and at the same time, helped long-term planners determine the optimal mix of vehicles in the fleet, over the next 5 years. Similar analogies exist in manufacturing, military, scientific and other areas. This dual-purposing is far, far easier to provide in a source-based application than a graphically constructed one.

Canned Solutions

Many workbench vendors have generic solutions that they will offer to tailor to your needs. If they do, it is wise to ask if these are simply solutions that were designed and built for someone else's problem. If it is, then there are several potential issues with this approach, and you should look deeper.

The original customer's solution may not match your solution. If it was designed and built with his unique constraints, and to solve his particular problem, or was designed as a one-off application, rather than a reusable foundation, then ripping out the other guy's idiosyncrasies and adding yours is often far more work - and therefore more expensive - than building from scratch. And with a framework as a starting point, you don't have to build from scratch.

Managed incorrectly, it can be a maintenance nightmare for you and for the company who would build it for you. If they copy the other guy's solution and start modifying it for you, there is a branch in the code's genealogy, and two (or more) copies that must be kept up to date. Bugs found in one branch (the other guy's code) often don't get fixed in yours, or worse yet, the "fix" gets implemented in your code, causing unforeseen breakage due to newly unique aspects of your branch.

In some cases, your solution could be used as a "template" for someone else's project. Suppose that "other guy" is a competitor?

Frameworks like HighMAST™ put common code in a foundation class library with clearly defined, standard behavior. Unique behavioral aspects of your domain are coded into your implementation layer, and those of the other guy's solution are coded into his implementation layer. Bugs in the foundation layer pertain to standard behaviors, and are fixed once. Bugs in the other guy's implementation layer are fixed there, and cannot cross-pollute into yours.

Summary

If you have a need to fill, and intend to do so through simulation, one of your first issues to address is likely the selection of a platform on which to build that simulation. We have discussed many factors that go into a decision of whether to use a workbench-style tool such as Extend or Arena, or a source-based environment such as HighMAST™. The size, complexity, lifecycle and integration needs of the simulation all play an important part in making that decision. It is a decision that will have an important effect on the success of your project, and which should be carefully considered. We invite you to contact us at tech@highpointsoftware.com if you would like to discuss your needs further.

About The Author

PETER C. BOSCH is a founder of Highpoint Software Systems, a small and attentive decision-support technology firm in the upper Midwest. He holds a BSEE from the State University of New York, and is a Certified Java Developer and Microsoft Certified Solution Developer. Pete has published numerous technical articles on object-oriented development in these environments, and has been designing and building simulations since 1991 for Fortune 100 firms in aerospace, medical imaging, pharmaceutical manufacturing and investment banking. He has been leading large software projects since 1995.