Vehicle Thermal Management – A Case Study in Web-Based Engineering Analysis

Michael Tiller
Xogeny Inc., USA
michael.tiller@xogeny.com

Abstract

Modelica has proven to be a compelling technology for creating sophisticated multi-domain models. It provides modern language features to promote model reuse and maximize developer productivity. These capabilities are backed up by proven simulation performance. More recently, the Functional Mockup Interface standard (FMI) has created an avenue for these models to be exported outside the model development environment as Functional Mockup Units (FMUs).

In this paper, we explore one possible way to utilize models that have been exported as FMUs. Specifically, we discuss how to incorporate these models into a web-based engineering analysis application that is designed to be accessible to non-expert users. Our goal is to show the important role that web and cloud based approaches can have in magnifying the impact of modeling activities across the enterprise.

We consider a specific engineering model and discuss exactly how we have transformed the model to make it accessible as a web-based application. This includes a discussion of the input and output data associated with the model as well as how a web based deployment (backed by cloud based services) can provide unique features compared to more conventional methods of model deployment.

Keywords: *FMI, cloud, web, HTML5, JavaScript*

1 Background

1.1 "Start the Revolution Without Me"

At the dawn of the world wide web, web servers served up static content (ordinary files) and web browsers rendered that content. What was remarkable about the web at this point was the ability to mix graphics and text using HTML and to hyperlink between pages. As the web matured, servers started to switch from serving static files to assembling content on a per request basis.

Up until 1998, the role of the web browser was simply to render this content produced by web servers. But in 1998, the "Web 2.0" revolution began. A big part of this revolution was the idea that the browser should become an application platform. Although at this point it wasn't clear what technologies would ultimate win out for "executing" content through the web (Flash, Javascript or a variety of proprietary plugins), it was clear that a browser was no longer just for rendering hypertext.

Since 1998, the changes in the web browser platform have been nothing short of astounding. In fact, most users of these web browsers are probably not even aware of all the capabilities that have not only been added to web browsers but also standardized across browsers. Modern web browsers don't just render hypertext, they can render vector graphics through SVG, detailed three dimensional scenes through WebGL, a wide variety of open ended rendering capabilities through `canvas` widgets and feature ever improving styling options through CSS. Along the way, the HTML specification has improved as well.

Behind the scenes, cloud based technologies are advancing by leaps and bounds. Every day new tools

and technologies emerge for supporting web-based applications. For example, there are now so many high-quality free options for databases that it is almost impossible to keep track of them. But this is true in nearly every area of web and cloud based technologies from Javascript frameworks to virtualization technologies.

The ubiquity of distributed computing resources is fundamentally changing the way we think about programming these large scale systems to make them more scalable and fault tolerant. This, in turn, has changed the way we think about programming. New languages and programming metaphors (*e.g.,* channels in Go[1][2], the `core.async` library in Clojure[3], actors in Akka[4] and promises[5]) are emerging to help us abstract away the behind the scenes complexity associated with these new distributed computing paradigms.

1.2 Engineering 2.0

But what does this mean for engineering? Almost nothing. While most engineers personal lives have been impacted somewhat by web based applications like Facebook, Twitter, Evernote, etc. their professional lives are still largely centered around applications, computing resources and storage tied to their local desktop. Simply put, the tools and capabilities in engineering are far from keeping pace with the proven technologies in other areas that have emerged over the last two decades.

Part of this effect is driven by the fact that engineering tools tend to be large and monolithic. This has resulted, in part, from industrial customers who approach adoption of engineering tools with a long list of requirements. This tends to foster a monolithic approach which, in an ironic twist, probably ends up hurting these customers in the long run.

One of the reasons that the technology industry seems to be better served by recent advances is that they are a more aggregated market. Engineering companies often attempt to differentiate themselves through what they feel are unique or innovative processes. The result is that their requirements are all slightly different from each other. This, in turn, means that vendors can rarely make a product that has broad market appeal in engineering. This leads to the situation that vendors often cannot justify significant development resources or reinvestment because they are, ultimately, addressing a niche market. Moving forward, engineering and industrial companies need to collaborate more to drive standards and common best practices. In this way, more resources can be brought to bear on the problems that they all share.

While the web has evolved well beyond "web 2.0", engineering is still waiting for the impact of those technologies to create an "engineering 2.0" revolution. Until then, engineering applications will remain largely centered around the desktop and subject to the same computing and storage limitations they always have and collaboration will be limited to "SharePoint" sites or network shared drives.

1.3 Xogeny

Xogeny was started to attack this problem head on. Xogeny is a new company with no legacy software to support. Everything we do starts with modern, proven technologies. There are countless technologies out there developed by technology giants like Google and Amazon that are simply there for the taking. Xogeny's mission is to help build the bridges necessary to make these technologies accessible for engineering applications.

The first step in this process was to develop a platform to easily distribute simulation of FMUs to the cloud. We call this platform FMQ.

2 Representative Model

2.1 Vehicle Thermal Management

For the remainder of this paper, we will be discussing a Modelica model used to study vehicle thermal management. This model was developed by Modelon[6][1] **without any prior consideration given to using it with the FMQ platform**.

The model itself is very sophisticated and includes detailed models of several different aspects of the vehicle. This model is used to simulate the thermal

[1]The author wish to thank Modelon for their participation in this case study and express his sincere appreciation for their support.

 DOI 10.3384/ECP140961073

response of many individual components and subsystem during a prescribed drive cycle. Such analyses are important when optimizing system level efficiency to ensure that components are not oversized for their purpose and that control strategies are effective in managing thermal loads without reaching unacceptably high temperatures in components.

2.2 Model Compilation

The model itself was written in Modelica. As part of the normal modeling process (unrelated to any specific application involving FMQ), a considerable amount of engineering information was captured. For example, most parameters in this model include a description, physical units as well as minimum and maximum parameter values. Modelica makes associating such information with the model quite easy.

This Modelica code is then compiled into an Functional Mockup Unit (FMU) that conforms to the Functional Mockup Interface (FMI) specification[7]. One important thing to understand about this process of converting Modelica into an FMU is that much of the engineering information discussed previously is propagated into the FMU. So the descriptions, physical types and limits of parameters are available via the `modelDescription.xml` file in the FMU.

The FMI specification translates Modelica parameter values and solution variables into "flattened" names. This means that while the original Modelica model was hierarchical, the parameters and variables contained in the FMU are essentially just a simple list (not a tree structure) with fully qualified names that effectively indicate their location within the model instance hierarchy. The consequence of this is that organizational information from the Modelica model is only partially preserved (via the names).

2.3 Input and Output Data

As mentioned previously, variable names convey information about the relative positions of variables within the model hierarchy, but their organization within the FMU is flat. The parameters (input data) we are interested in characterize several different subsystems in the vehicle. These can be organized roughly into parameters related to the heat exchangers, the powertrain and the chassis. Each of these different categories contains information about both the physical characteristics of components but also various control strategy calibration parameters as well. Also, some of these categories may have so many parameters associated with them that they necessitate further hierarchical organization. So, for example, the data associated with the heat exchangers is further broken down into information about stack geometry, fan control and scale factors.

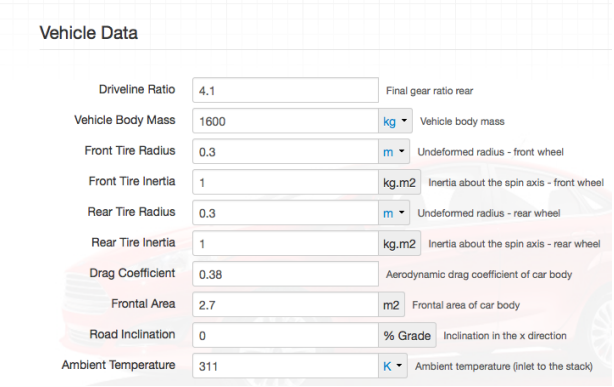

Figure 1: Input data panel

In addition to input data, there is also the question of what kinds of results can be generated from the FMU. In the case of this model, there are an enormous number of variables that are evaluated when the FMU is simulated. We won't attempt to list them all here, but the set of output variables consists of nearly any kind of information you might be interested in when exploring vehicle thermal management. As we will discuss shortly, we will only use a handful of these results. But it is worth noting that the FMQ platform itself is capable of computing and extracting whatever signals we need.

Figure 2: Output Report

3 Supporting Non-Experts

Before jumping into a discussion about web-based engineering analysis, it is worth taking some time to discuss **why** a web-based analysis tool is useful. The first thing to clarify is that this paper is not about web-based **development** tools. In other words, these tools are not designed for the creating Modelica models. The process of model development is a complex process and requires a sophisticated user interface. While creating a web-based development platform is a very interesting topic, it is not the topic of this paper. As such, for the purposes of this paper we assume that models are developed in an existing model development environment. The question we are focusing on is how to make these models accessible to non-developers or **non-experts in general**. So let us spend a little time considering a workflow that involves non-expert users.

Consider the following scenario. You are a software developer. You are writing software that you want lots of people to use. You work very hard developing the software and now you are ready to let other people use it. Imagine if circumstances required that instead of simply installing a compiled version of your code, users of your software had to buy all of your development tools (*e.g.,* Visual Studio), find a way to get the source code to your application, and then build your software for themselves.

If your "user" was an ordinary consumer (and not an open source hacker), this approach would be a complete disaster. Until recently if you were a Modelica developer, this would have been the only way to distribute your models **and** the compiler that your users would require could potentially be rather expensive. Fortunately, with the emergence of the FMI standard, we have an open and widely format for distributing models.

As a result, potential model "consumers" no longer have to compile the models themselves using expensive model development tools and their associated traps and pitfalls. However, simply having an FMU is not a complete solution for deploying models to non-expert users. There is still the issue of adding an appropriate graphical user interface. It is also important to recognize that non-expert users require an interface that is relatively straight-forward to use and warn users about potential mistakes. Such an application will typically expose a limited set of input and output data. This data should be organized in a way that is intuitive to the user. Finally, an application developed for non-expert users must focus on addressing the specific questions of that user. If this means masking some or even most of the underlying models general capabilities, then so be it.

There is an (often uncaptured) return on investment for such efforts. Many model developers are forced to spend at least some of their time justifying the resources they consume for model development (in the form of time, expensive tools, etc). This concept of model deployment (*i.e.,* having an easy path for getting analyses leveraging your models into the hands of non-experts) is one potential way to demonstrate the need for model development resources. If model developers have an easy means to deploy models to end users and, as a result, turn those non-expert users into enthusiastic customers, then they have an opportunity to create a "pull" effect for their models. This can potentially lead to greater demand from the organizations that use those models. In such cases, this can lead to a virtuous cycle where the model developers can more easily justify their resource demands and, ideally, this increased demand will allow them to focus more resource on model development.

It is worth noting that non-expert users are generally interested in performing some kind of analysis. A model may be central to this analysis, but the analysis often involves more than just a single simulation of a single model. In a sense, a model is simply a complex function (*i.e.,* you give it data and it returns data). Creating and application and an high quality user experience involves much more than providing people with a function. We have deliberately used the term web-based engineering *analysis* applications because we think it is important to explicitly recognize that these applications must be prepared to support the complete analysis, not just part of it. Leveraging one or more models, a typical analysis will involve an optimization, a sensitivity study, a Monte-Carlo analysis or some other numerical procedure.

4 Web-Based Deployment

Given the requirements for non-expert users and the representative vehicle thermal management model previously discussed, the remaining question

DOI 10.3384/ECP140961073

is what should be the medium for model deployment.

4.1 Desktop Limitations

Given the case previously made for deploying models to non-expert users, one approach could be to develop desktop based analysis tools that are distributed to end users with the necessary models embedded in them. Indeed, one could argue that this is the conventional choice. But this approach has several important drawbacks.

The first issue to consider is data management. A tool that runs on a desktop has many inherent limitations. Typically, results files are simply written to a user's desktop and managing the results is left up to them. In some cases, running an analysis simply overwrites any previous analyses (making it very easy to lose data). A desktop environment makes it hard to collaborate with others because sharing results means sharing results files which are not easily shared when they are locked up in a desktop environment. Email and shared network drives are one way to address these issues, but they still involve a lot of manual work and discipline by users.

Another issue with the desktop is limited computing power. What if your analysis deals with uncertainty and requires a Monte-Carlo approach to simulating models. Or, what if you want to perform a large scale DOE or sensitivity study. There many types of analyses that could require hundreds, thousands or even more individual simulations. For even a high-end desktop environment, this could mean long wait times for analysts and this only aggravates the data management problem previously discussed.

In most organizations, a desktop application also means involvement with IT. The application has to be installed on the user's desktop and then updated when new releases come out. Getting the necessary IT resources to support installation across desktops is logistical aspect that must be dealt with and another drain on corporate resources.

4.2 Web-Based Client

To address many of these limitations, Xogeny has developed a set of tools for building web-based engineering analysis applications. These tools extract information from FMUs and build a high-quality, intuitive user interface using this information.

As mentioned before, these FMUs frequently come from Modelica tools. As such, they include lots of useful information that can be automatically incorporated into the user interface. So, for example, such a web application can automatically incorporate logic to warn users about parameter values that are outside the excepted limits. The nice thing about how these applications are built is that if the FMUs are updated, the information captured in them (parameters descriptions, default values, physical units, upper and lower limits) can be automatically incorporated into the application.

One might assume that a web browser is not an appropriate context for engineering applications because engineering applications require rich user interfaces with support for plotting, complex diagrams, detailed human machine interfaces or 3D animation. However, modern browsers have all this in the form of HTML5 support. Widely used web browsers like Firefox and Chrome can do all of these things seamlessly and without the need for any plugins.

A web-based application is easy to deploy and update because all that work can be done centrally once for all users. This reduces the impact on both the end user and the IT infrastructure to support installation and maintenance at the desktop. In fact, "installation" is typically as simple as circulating a URL to potential users. Furthermore, it is much easier to track usage and restrict access using a web-based approach.

In the case of the vehicle thermal management application, the web application can be found at the following URL[2]:

`http://vtm.demos.xogeny.com`

This user interface demonstrates that a high quality HTML5 user interface can be synthesized from information contained in an FMU. Such an interface can incorporate the necessary business logic to support non-expert users and operate in a way that is intuitive for most users.

But in order for this application to function, it must

[2]The "simulation" capabilities of the public web application have been removed. The results presented are actual simulate results, but they are simply injected rather than simulated. As a result, they are not affected by the parameter values in the UI.

be able to perform simulations. A web-browser generally has very limited access to the user's desktop (certainly not enough to execute a simulation). So how does this interface perform the simulations required for the analysis? When the web application is compiled, support for simulation is compiled in through the `fmq.js` Javascript library. This library provides a native Javascript interface to the FMQ platforms cloud-based simulation capability. We'll return to the simulation side shortly.

One last thing to note about the web-based user interface is the presence of a "History" view. This functionality is enabled by the **data management** features provided by FMQ. This view presents the user with a graphical history of their interactions and includes a tree structure of the different data sets that the user has either saved, simulated or is currently working with. It visualizes the relationships between each data set (which ones were derived from which). Hovering over a given version provides a summary of differences between that dataset and its parent dataset. All of this data is collected automatically and passively (*i.e.,* the user doesn't have to do anything explicit or manually). Because this information is stored centrally, it would be easy to generate hyperlinks associated with each dataset visualized in the "History" view to share with colleagues. This kind of hyperlinking is the essence of the web but, unfortunately, it is not widely supported in the context of engineering tools.

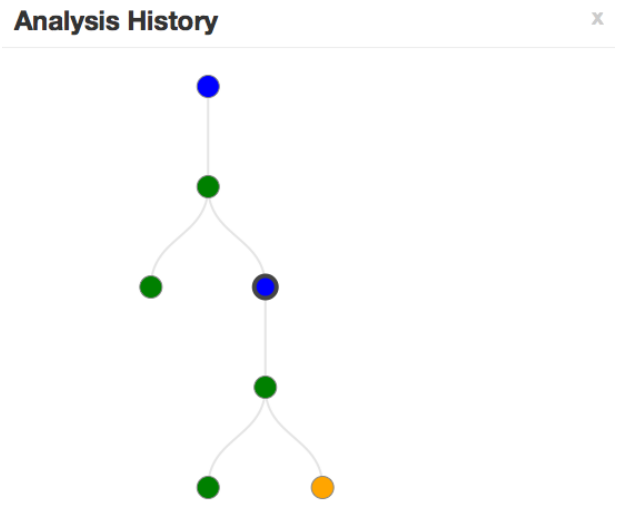

Figure 3: Visualization of data history

4.3 Cloud-Based Services

The central component of the FMQ platform (currently) is the ability to simulate an FMU. In this process, the FMU is registered in advance with the system and a RESTful web services API is used to request simulations involving specific parameter sets. This RESTful service can be accessed directly (to support large scale batch processing of FMU simulations) my making programmatic web requests or via a web-browser. For the web-based use case discussed in this paper, a special high-level Javascript library was developed that leverages the low-level RESTful API behind the scenes.

In addition to batch processing, the FMQ platform is capable of supporting interactive simulations. There are two sides to this interactivity. The first is the ability to stream simulations results interactively as the simulation is running. This means that clients can be asynchronously updated when new simulation results are generated. But it also means that the client can interactively feed **input** signals to the simulation while it is running as well. The result is the ability to create web-applications where the user manipulates the model and the model responds interactively.

Simulation results can be handled in several ways. We already discussed the ability to stream simulation results to the client as they are generated. In many cases, it may be preferable to simply wait until the simulation is complete and then provide the trajectory for key variables to the application. There are other cases where the web-based application may not have an immediate need for simulation results but it might wish to access them later. Finally, some types of analyses may generate large binary files as a by-product of the analysis (*e.g.,* a "meld" file from a simulation[8]). In those cases, the application may wish to access such artifacts. The FMQ platform supports all of these different use cases.

The FMQ services provide complete accountability for all data being processed. What this means is that for every analysis request being made, there is a record of the input data to the analysis and of all data generated from the analysis. Furthermore, the relationships between the job requests, job data, results and binary artifacts are all represented through FMQs Hypermedia API[9]. This provides a comprehensive approach to data management.

One final topic worth discussing is security. One of

DOI 10.3384/ECP140961073

the main concerns with any cloud-based solution is the security of the data. An important aspect of security is understanding exactly what "threat" is at issue and what techniques can be used to mitigate or eliminate those threats. A full discussion of such threats is beyond the scope of this paper but we are happy to discuss this topic. But it should be noted that the FMQ software is not tied to any particular platform or provider. As such, many of these security concerns can be addressed by **running the FMQ software on a computing cluster within the customer's own internal network**.

5 Conclusion

This paper presents a representative engineering model and shows how that model can be shaped into an intuitive user interface with sufficient business logic and guidance to be used by a non-expert user. Such an application can directly leverage information already available from the FMU. The application can utilize an array of technologies already available in HTML5 to provide plotting, 3D rendering, sophisticated reports and human machine interfaces.

But more importantly, by leverage the capabilities already being actively developed and supported around web and cloud-based technologies, we can **improve** the engineering process providing better data management, greater opportunities for collaboration and data sharing as well as richer views of the engineering data and process by aggregating information that be automatically and passively collected through simple use of the application.

Ultimately, the goal of this work is to underscore the importance of model development. By providing an avenue for model developers to deploy their models to more users, we hope to create a positive feedback loop that will emphasize the value of model development and model-based system engineering that will, in turn, provide greater resources to model developers for creating high-quality, high-fidelity and high-impact engineering models.

This effort is really just the beginning. The FMI initiative helped to unify previously fragmented potential markets. The adoption of FMI has just started the process of building an eco-system around the FMI standard. FMQ is simply one example of how we can leverage the power of web and cloud based resources and standardization to help drive improvements in the engineering world.

References

[1] Go Development Team. *Effective Go*. 2014. URL: `http://golang.org/doc/effective_go.html`.

[2] Caleb Doxsey. *An Introduction to Programming in Go*. Caleb Doxsey, 2012. URL: `http://www.golang-book.com/10#section2`.

[3] Rich Hickey. *Clojure core.async Channels*. 2013. URL: `http://clojure.com/blog/2013/06/28/clojure-core-async-channels.html`.

[4] Jamie Allen. *EffectiveAkka*. O'Reilly Media, 2013.

[5] PromisesA+ Organization. *PromisesA+ Specification*. 2013. URL: `http://promisesaplus.com/`.

[6] John Batteh, Jesse Gohl, and Sureshkumar Chandrasekar. "Integrated Vehicle Thermal Management in Modelica: Overview and Applications". In: *Proceedings of the 10th International Modelica Conference* (2014).

[7] MODELISAR Consortium. *Functional Mockup Interface, Version 1.0*. 2010. URL: `https://svn.modelica.org/fmi/branches/public/specifications/FMI_for_ModelExchange_v1.0.pdf`.

[8] Michael M. Tiller and Peter Harman. "recon – Web and network friendly simulation data formats". In: *Proceedings of the 10th International Modelica Conference* (2014).

[9] Roy T. Fielding. *REST APIs must be hypertext-driven*. 2008. URL: `http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven`.