MDA Journal

**David S. Frankel**

Lead Standards Architect – Model Driven Systems

SAP Labs

David.Frankel@SAP.com

Mr. Frankel's SAP WebLog is:

https://www.sdn.sap.com/fr/sdn/weblogs?blog=/pub/u/55914

Author:
Model Driven Architecture:
Applying MDA to Enterprise
Computing

# MDA's "Last Mile" Problem

**Telecommunication's Last Mile** ........ **1**

**MDA's Last Mile** ........ **1**

**Green Fields and Legacy** ........ **2**

**Model-Driven Service Orchestration** ........ **2**

Business Process Models and the Last Mile ........ 2
Composite Application Models and the Last Mile ........ 3
What About Model-Driven Service Development? ........ 3

**Not Building Applications Anymore** ........ **3**

**Product Lines and the Last Mile** ........ **3**

**Conclusion** ........ **4**

The proverbial "last mile" problem has inhibited the growth of broadband telecommunications. Analogously, a last mile problem constitutes a major challenge to realizing the promise of MDA.

## Telecommunication's Last Mile

In recent decades, technical visionaries and aggressive financiers dreamed up and spawned vast global networks of super-high throughput fiber optic cables. These networks have incredibly efficient mass switching systems based on complex applications of mathematics and advanced science, along with countless, increasingly intelligent routers that direct unprecedented amounts of data traffic.

However, the nagging problem of how to get the data to tens of millions of individual residences and businesses has been a drag on the widespread deployment of bandwidth intensive applications in the United States. For example, while the U.S. struggled with the technical and regulatory legacy of copper wire, high growth Pacific Rim countries such as South Korea surged ahead in broadband deployment, unencumbered by the past.

Today, copper is gradually giving way to coaxial cable, satellite broadband, and high power neighborhood wireless access points. Nevertheless, deployment of high speed broadband to U.S. residences and small businesses is well behind that for countries such as South Korea.

## MDA's Last Mile

Recently I read a report by IDC that analyzes the model-driven development market.[1] Overall the report is a worthwhile read, but one thing in particular caught my eye. IDC singles out the challenges that model-driven approaches face in dealing with the "last mile problem of application development – making the application come to life."

I find it intriguing to think about MDA this way. One of the main challenges MDA has to deal with is how to use models to create running systems. Good modeling languages and techniques that support abstraction and clear separation of concern have value in their own right, but the value proposition that MDA proposes goes beyond the virtues of modeling. MDA looks at models as production artifacts. MDA models should be as much a part of the software production process as are the data models that drive database management systems. But models don't achieve this status unless they drive production machinery, just as data models drive runtime database engines and WYSIWYG models of GUIs drive code generators that produce GUI code.

## Green Fields and Legacy

The vast majority of software systems that we build are not green fields, but, on the contrary, need to work with legacy systems. Just as legacy proved a drag on solving telecommunication's last mile problem, legacy complicates the task of using models to bring applications to life.

For example, when all of the data for a solution is highly normalized in a relational database, it is relatively straightforward for a model compiler to use the abstract data definitions in a model to generate code that implements data access and retrieval via SQL calls. However, when much of the data is in various legacy applications and components that offer disparate means of getting at the data, it's more difficult to generate the access and retrieval code.

## Model-Driven Service Orchestration

Often times, developers trying to get a handle on MDA approach it with the thought that they have to use it to generate the components of their applications, or at least to generate fat skeletons of the components, to which programmers add some finishing code. Then they run straight into the last mile struggles with legacy back end systems that hold so much of the data in non-normalized form.

Some of the most successful approaches that I've seen take a different route to nailing down the last mile with MDA. They use service-oriented architecture (SOA) to present existing application functionality as a set of service components whose interfaces are highly normalized via consistent use of service interface languages such as Web Services Description Language (WSDL). They use modeling to define specific orchestrations of the services. They walk the last mile by compiling or executing the orchestration models.

In sum, this approach deals with the last mile problem that non-normalized back end systems pose, by using SOA to provide a normalized intermediate tier that model compilers and model execution engines can target in a consistent fashion.

There are two kinds of orchestration models in this environment: Models of business processes and models of composite applications.

### Business Process Models and the Last Mile

Business process models receive a lot of attention in Business Process Trends. Model-driven business process management has tended to focus on process flow models expressed via Business Process Modeling Notation (BPMN) or as UML activity graphs.

Business process execution engines bring such a process model to life by "executing" it after IT architects have connected the steps of the process to IT assets. The more the interfaces to the

[1] Stephen D. Hendrick, "Worldwide Model-Driven Development Software 2006-2010 Forecast and 2005 Vendor Shares," IDC, December, 2006www.idc.com

IT assets are normalized, the more tractable it becomes to automate the establishment of these connections and the less code has to be written to do it.

In this manner, we are making progress in walking the last mile by executing BPMN/Activity style business process models.[2]

### Composite Application Models and the Last Mile

Composite application models also orchestrate service components, but at heart they are models of specialized applications, not models of business processes. The applications they define are more akin to the Web 2.0 “mashups” currently in vogue than they are to business process models.

Nevertheless, these models are at a high level of abstraction, because they need not model the basic application behavior that is encapsulated in service components. Instead, they model how to use these components together to support innovative business processes.

A composite application model feeds into a model compiler that generates the “glue” code that orchestrates the services in the manner that the model defines. There is no need to separately wire the model to IT assets, because the visual composite application modeling paradigm itself establishes these connections. The modeler drags icons representing specific services onto the modeling surface and models the interactions among the services via various kinds of visual connectors that define data flow and control flow.

In cases where the modeling language is insufficient to express all the aspects of the orchestration, programmers have to come into the picture to write 3GL code. The approach to adding such code is a critical last mile issue. In order not to break the iterative, model-driven composite application framework, this code must be packaged and normalized as a service that looks like a regular service to the modeler. Furthermore, the modeling environment and the coding environment must integrate smoothly so that the coder can flip back and forth between the modeling and code perspectives to debug the code and test how the application works with it.

### What About Model-Driven Service Development?

Model-driven service orchestration focuses on coordinating service components, and sidesteps the question of how to construct the services’ functional code. Fundamentally, it doesn’t care whether a programmer writes this code or if model-driven driven tools generate it.

From this standpoint, model-driven service construction is a “nice to have,” but it’s more difficult to achieve because at the end of that road lies the last mile of disparate legacy back ends. Model-driven orchestration is the relatively “low hanging fruit.”

## Not Building Applications Anymore

Putting a priority on model-driven service orchestration dovetails with a strong trend whereby IT organizations no longer engage in the kind of “bread and butter” application development that used to consume most of their energy. CIOs will tell you that they spend the vast bulk of their development budgets on integration rather than applications.

There has been a trend toward purchasing packaged software, and that software is gradually being encapsulated or refactored as sets of normalized services. That sets the stage for model-driven orchestration.

## Product Lines and the Last Mile

I’ve written on these pages before[3] about the Carnegie Mellon Software Engineering Institute’s approach to software engineering called *Software Product Lines*, and about the application of

[2] However, for the most part the industry has not worked out how to execute other kinds of business process models, such as SCOR supply chain models, which humans can readily consume but automated execution machinery typically cannot.

product line practices to model-driven development spearheaded by the Generative Programming[4] and Software Factories[5] initiatives.

Here I digress slightly from the focus of this article on model-driven orchestration, but I am struck by the fact that a product line approach to model-driven development resembles model-driven orchestration in an important sense: Its model compilers generate glue code that invokes pre-provisioned, reusable components.

Product line practices force you to sharply focus your scope to a well-defined product line when undertaking model-driven development. You build a set of components (by hand or by model-driven techniques) that are specific to the product line's domain, and you create a domain-specific language (DSL) for modeling individual products. The DSL can be a UML profile or something else entirely.

Thus, the DSL compiler brings the individual product to life by generating connections to the pre-provisioned, domain-specific components. The components encapsulate and normalize whatever backend functions are required.

## Conclusion

The IDC research report includes figures that demonstrate that the lion's share of revenue growth in the past few years in the model-driven development space is in business process modeling tools. Thus there is hard data to corroborate anecdotal observations that the trend for IT development budgets to fund integrations rather than classical application building means that the most lucrative market for MDA is in model-driven orchestration in an SOA environment. Not coincidentally, this is the environment in which the MDA's last mile problem is the most manageable.

*David Frankel is Lead Standards Architect for Model-Driven Systems at SAP Labs. He has over 25 years of experience as a programmer, architect, and technical strategist. He is the author of the book," Model-Driven Architecture®: Applying MDA® to Enterprise Computing." He also is lead editor of the book, "The MDA Journal." He served several terms as a member of the Architecture Board of the Object Management Group (OMG), the body that manages the MDA standards, and he has co-authored a number of industry standards. Recently, he has been publishing and speaking about the role of model-driven systems in enterprise SOA.*

[3] "The MDA Marketing Message and the MDA Reality," Business Process Trends, March 2004, pp. 3-4.

[4] Krzysztof Czarnecki and Ulrich W. Eisenecker, *Generative Programming: Methods, Tools, and Applications*, Addison-Wesley, 2000.

[5] Jack Greenfield and Keith Short, *Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools*, John Wiley & Sons, 2004.