MDA Journal

David S. Frankel
David Frankel Consulting

david@dFrankelConsulting.com
http://www.linkedin.com/in/davidsfrankel

Author:
Model Driven Architecture:
Applying MDA to Enterprise
Computing

UML Profiling Comes of Age

Realizing the Potential of Domain-Specific Modeling

Over ten years ago, when the OMG's Model-Driven Architecture (MDA) initiative was just getting started and I was working on my first book on MDA, I and some of my industry colleagues had a dream that someday we would have frameworks that would allow us to create domain-specific graphical modeling languages and tools with orders of magnitude less effort than it required at that time. Since then, the industry has followed this dream down several different paths. In this Column I have the pleasure to report to you that to a significant extent this dream is now being realized.

Paths to Domain-Specific Modeling

One of the directions that domain-specific modeling advocates pursued is based on the Eclipse Modeling Framework (EMF). Eclipse has a stack of tools built upon EMF that together provide frameworks for domain-specific modeling, including textual modeling languages and graphical modeling languages. The EMF stack is more or less based on the the OMG's MDA standards, with some deviations.

The Microsoft Software Factories initiative also embraced both textual and graphical domain-specific modeling languages. Although some companies have done quite extensive software development with the Software Factories framework, overall Micorosoft decided a few years ago to base its software modeling efforts on UML; substantial parts of the Software Factories framework survived, repurposed toward supporting UML modeling.

UML tools that implement UML's profiling capabilities represent another key category of support for domain-specific modeling. Although UML profiling cannot be used to define and provide tool support for all graphical modeling languages, it can nevertheless support many such languages. Recently, I've been working with UML tools of this category, and thus this Column concentrates on that experience.

Before I dive into the UML profiling subject, I should point out that the three paths I outlined above

Copyright © 2013 David S. Frankel. All Rights Reserved.

are not entirely mutually exclusive. The Eclipse UML 2.0 project builds on the EMF stack to some extent, and Microsoft has developed and shipped a UML tool that supports UML profiling.

UML Profiling

The architects of UML never expected it to satisfy industrial modeling requirements “out of the box.” Rather, UML is a base language with extension mechanisms. Virtually all serious applications of UML use the extension mechanisms to define modeling constructs that are not part of the base UML language.

A UML Profile is a package of UML extensions that address a particular purpose. A modeling construct that a profile defines can have properties not supported by the UML construct that it extends, and/or can constrain the values of pre-existing properties. Some UML Profiles have been standardized by various standards bodies, while others are proprietary.

For example, the base UML language does not have constructs for modeling many aspects of distributed enterprise systems, including security, distribution tiers, quality of service, and so forth. Standardized and proprietary UML Profiles have thus been created to enhance expressiveness regarding these subjects.

The Origin of the UML Profiling Specifications

After the release of the original UML 1.0 specification in 1997, the industry gradually started to use UML’s profiling mechanisms. This experience proved frustrating for many. Generally speaking, the UML tools did not support the mechanisms in a straightforward or truly UML-compliant fashion. The tools realized only a small fraction of the potential power of UML profiling.

Part of the problem was that the UML specification said very little about the overall purpose of the profiling mechanisms and their potential power when combined synergistically with other UML mechanisms such as the Object Constraint Language (OCL). When the effort to define UML 2.0 began in 2001, one of the key objectives was not only to make the profiling mechanisms more robust, but also to provide more extensive explanation of their purpose.

The Vision

The vision we had as we began this task was that with a UML tool an architect could create a profile that defines a set of domain-specific modeling constructs, and that the tool would use the profile definition to make the desired modeling constructs available to the modeler, i.e. available to the user of the profile.

The profile definition would contain constraints written in OCL, in order to rule out combinations of elements, element values, and relationships among elements that would not make sense for the specific domain. Furthermore, the tool would use these constraints to identify offending elements of models based on the profile, and to guide the modeler to rectify the problems.

All of the capabilities would be contained in the tool – the ability to define the profile’s new modeling constructs; the ability to serve as the modeling tool using those constructs; and the ability to serve as the debugger identifying illegal modeling.

We felt that, armed with tools providing such support, architects could create all kinds of domain-specific modeling languages and tools, even ones that aren’t fundamentally about object-oriented software modeling. Many modeling languages and tools that graphically require shapes (such as boxes, ovals, etc.) and connectors between the shapes could be defined this way. This would exclude languages that need complex graphical elements such as spin dials, but that still would leave many languages that just require simple shapes and connectors.

UML 2 Profiling

In keeping with that vision, UML 2.0 and the subsequent minor 2.x version updates have a complete chapter devoted to profiling. A key improvement introduced in UML 2.0 was the introduction of a standard mechanism for representing a profile’s definition. There is actually a standard way to model a profile graphically, so that any conforming UML 2 tool can undersand the definition of the profile. The role of constraints in profiles was also spelled out more explicitly.

When the first crop of UML 2.0 tools emerged, many of us were delighted to find that the tools' creators were actually making an effort to faithfully implement the UML 2.0 specification. The new tools were also more mature as desktop tools, having implemented multi-level undo and redo, font and color management, and other such key usability features at a higher level of functionality and robustness than was generally the case in the UML 1 tools. The good UML 2 tools understood a profile's definitions of new modeling constructs and how to use them to provide the modeler with the new modeling constructs within the tool.

Still, it took a while for OCL support to evolve in the tools. But we are now at a place where more and more of the top tier UML tools support the validation of a model by checking whether the model conforms to the constraints of the profile upon which it is based.

For example, in my last MDA Journal column[1] I described some aspects of a domain-specific language for modeling the semantic structure of an element of an SOA system. This language has been developed for a standards organization named the Banking Industry Architecture Network (BIAN). Figure 1 below is reproduced from that Column. The modeling of the semantic structure of the CreateNewCurrentAccountInventory service operation is really quite different conceptually than traditional object-oriented modeling, and yet UML's profiling mechanism works very well for it because graphically the language fits the shape and connector paradigm. Furthermore, the definition of the language includes constraints that the modeling tool uses to rule out illegal use of the relationships; for example, constraints require ActionTerm relationships to point to a VocabularyEntry, and if the relationship points to something else the tool cries foul and directs the modeler to correct the transgression. There is much more to the domain-specific graphical modeling language that this is part of, yet it took only a few weeks to put the entire domain-specific language and modeling tool together, and the tool has the usability virtues of good desktop tools because the enveloping generic UML tool does.

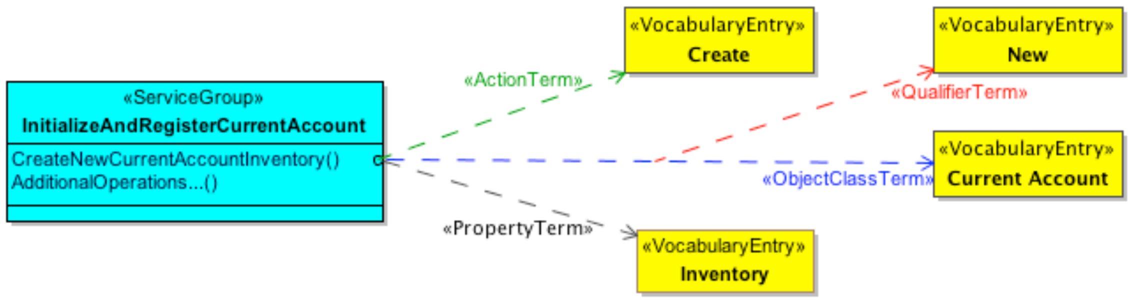
«ServiceGroup»
InitializeAndRegisterCurrentAccount
CreateNewCurrentAccountInventory()
AdditionalOperations...()
«ActionTerm»
«VocabularyEntry» Create
«VocabularyEntry» New
«QualifierTerm»
«VocabularyEntry» Current Account
«ObjectClassTerm»
«PropertyTerm»
«VocabularyEntry» Inventory

Figure 1: Using a Domain-Specific Modeling Language

UML Profiling Gaps

Despite substantial progress, there are some notable gaps in UML's profiling capability that are worth summarizing.

One of the gaps is best grasped by imagining that you have defined a metamodel for some language, using MOF, which basically is the UML subset used for metamodeling. Imagine further that you want to use UML profiling to create a domain-specific modeling tool that operates in

[1] "Semantic Interoperability for SOA," originally published as an MDA Journal Column in BP Trends, September 2011, updated version available at http://www.dfrankelconsulting.com/Articles.html

conformance with the metamodel. The problem is that there is no standard mechanism, that would be understood as part of the profile definition, to map the metamodel to the profile. Why is this a problem?

The profiling mechanism works well in cases where UML has nothing out of the box that you can use to express some concept in the metamodel and you thus clearly need to create a new modeling construct. You declare which out-of-the-box UML construct your new construct is based upon and define any additional properties and constraints the new construct needs that the out-of-the-box construct doesn't have. All is well in this scenario.

However, for some other construct of your metamodel, UML may have a similar construct that would serve perfectly well without extension. For example, suppose your metamodel defines the notion of Rule, and you discover that UML's Constraint construct is perfectly suitable and has all of the properties you need for representing a Rule in line with your metamodel. There is no standard way to declare, as part of your profile definition, that the Rule construct in your metamodel maps to the Constraint construct in UML. Therefore tools either do this differently, or, probably more often, don't support it at all and you have to express the mapping in English, which is tedious and not exploitable by tools.[2]

The profiling mechanism also does not support execution of dynamic models that are based on profiles. If the domain-specific modeling language expresses dynamic things, like the progression of some procedure from one step to the next, the profiling mechanism has nothing to say about how a tool would execute the procedure. The OMG has made a lot of progress in defining standard execution semantics for executable models in its Foundational UML specification[3], but so far it does not cover the execution semantics of profiles. Addressing this gap is on the roadmap for OMG, but it will probably be a few years until we will see anything like that.

It's also worth mentioning that one of the purposes of UML profiling is to enable domain-specific graphical models to be the basis for generating various kinds of production artifacts such as code, database definitions, test suites, and so on, but artifact generation itself is outside the scope of the profiling mechanisms. Arguably this is not really a gap, just a separation of concerns.

Conclusion

UML tools have come a long way since the early days, and good support for profiling is one of the main reasons why. There are quite a few serious industrial projects that have taken advantage of UML profiling to good effect. UML profiling is not appropriate for all domain-specific graphical modeling languages, but I suspect that many users of UML tools are not fully aware of the power that is there to exploit.

David Frankel has over 30 years of experience in the software industry as a technical strategist, architect, and programmer. He is recognized as a pioneer and international authority on the subject of model-driven systems and semantic data modeling. He has published two books and dozens of trade press articles, and has co-authored a number of industry standards including XBRL, ISO 20022, BIAN, and UML®.

[2] I felt this gap when working on the UML Profile for ISO 20022. ISO 20022 is an important standard in the financial services sector, and has a metamodel and a UML Profile that realizes the metamodel for UML.

[3] http://www.omg.org/spec/FUML/