Applying MDD, Generative Programming, and Agile Software Techniques to the SDR Domain
What is a Domain Specific Language?

- In order to understand DSLs, one must understand levels of modeling
- DSLs are defined using Meta-Models
- Meta-Models are defined using even higher level modes

So, a Domain Specific Language is...
- A language targeted to a particular problem
  - Such as Software Radios
- Not a general purpose language aimed at any kind of problem
  - Such as UML
DSLs allow simplified modeling in the **Problem Space** vs. complex modeling in the **Solution Space**
Model Driven Development

- **Model Driven Development**
  - Models are used to express the design of a system
  - Models are transformed to create implementations
- Model Driven approaches combined with *Generative Programming* technologies allow developers to “create designs that write code”
- Developers use *Domain Specific Languages* and *Models* to efficiently and automatically map to platform-specific technologies
- Model Driven approaches allow us to effectively and practically move from making one-time concrete systems to *families of systems*
  - Since the meta-models capture the system family rules
Model Driven Software Development

- Allows developers to **weave various aspects** of the solution domain together automatically
- Increases **productivity and correctness** in complex systems by **simplifying** development
- Puts tools in the hands of developers so they can properly **capture the commonalities and variabilities** of their domain
- Captures the “**sweet-spot**” of many areas of software development
  - Including modeling, code generation, coding, testing
  - Could program in assembly or C++, what is the best combination of software tools to get the job done and done correctly
- Supports the creation of development **processes** for the domain
  - Explain exactly what is meant here
- Supports the creation of **Domain Specific Tools**
- These Tools further **eliminate the complexities associated with development** in a particular domain
Levels of Abstraction

Higher Order Languages

Assembly

Opcodes

Custom Hardware

Flexibility

More Flexible

More Productive

Productivity

Less Flexible

Less Productive

Less Flexible More Productive

More Flexible Less Productive

Flexibility

Productivity
Generative Programming

- The process of moving from a higher level abstraction to a lower level abstraction automatically.
- Specification of transformation rules support this paradigm.
  - e.g. C++ to Assembly to Opcodes.
- Domain Specific Models and Languages work in concert with generative technologies.
  - The whole is the worth more than the sum of the parts.
- Thereby increasing the productivity.
To summarize...

- Model driven development defines higher levels of domain specific abstractions and combines these with generators that automatically transform these abstractions to lower level **executable** artifacts.
- An Engineering Trade-Off
  - Sacrificing flexibility for productivity.
- Not a value judgment
  - Its payback is in the cost reduction found in developing **Families Of Systems** vs. Single Concrete System.
Focus on System Families allows one to identify the commonalities and variabilities found across family members.

And develop DSLs to:

- **Factor out** common behavior into parameterizable abstractions
- Provide **extension mechanisms** to incorporate variation points found across family members

And further develop generators to synthesize concrete functionality for a particular family member.
Radio Families

Radio Family Members

- **Commonalities**
  - Properties
  - Tests
  - Life Cycle
  - Communications Path
  - Deployment
  - Functionality (Routing, Networking)
  - Basic architecture

- **Variabilities**
  - Functionality
  - RF or SiS characteristics
  - Processing Elements (HW)
  - Size weight and power constraints
What the SCA has done … and has not

- SCA isolated the commonalities and variabilities but did not provide a DSL in which to program these things
- Also no generators
- So to really complete the picture, need generators to handle practical use and to map against the variabilities
- The SCA provides the necessary abstraction and framework of patterns (Extension Object with component Configurator) as well as the deployment and configuration engine.
- What is needed now are the remaining artifacts to make this particular solution complete.
Providing the remaining steps

- Allow programmers to program in the higher order domain by
  - Providing a domain specific grammar
  - Graphical representations of this grammar
  - Automatic constraint engines to ensure the use of the grammar is correct
  - Automatic generations engines
    - transform the resulting model to various targets along varying dimensions
    - Weave together various complex aspects of the domain
## The steps

<table>
<thead>
<tr>
<th>In general</th>
<th>In our domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>Isolate the abstractions and how they work together</td>
<td>The SCA</td>
</tr>
<tr>
<td>Create a formalized grammar for these - DSL</td>
<td>Create a formalize SCA meta-model</td>
</tr>
<tr>
<td>Create a graphical representation of the grammar – GDSL</td>
<td>Create a SCA specific graphical tool</td>
</tr>
<tr>
<td>Provide domain-specific constraints – GDSCL, DSCL</td>
<td>Program into the tool the constraints</td>
</tr>
<tr>
<td>Attach generators for necessary transformations</td>
<td>C++, C and VHDL generators</td>
</tr>
</tbody>
</table>
The SCA
The Metamodel

- Component
  - ID: EString
- Factory
  - Name: EString
- Resource
  - Port
- Property
  - Type: EString
  - name: EString
- ProvidesPort
- UsesPort
<?xml version="1.0" encoding="ASCII"?>
<com.prismtech.spectra.sdr.sca2_2.models:Assembly
    xmi:version="2.0"
    xmlns:xmi="http://www.omg.org/XMI"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:com.prismtech.spectra.sdr.sca2_2.models="http://com.prismtech.spectra.sdr.sca
2_2.models">

    <components Name="BitFlipper" organization="PrismTech" id="DCE:8f647411-91a1-4295-bbc6-6d3eff4982f7">

        <ports xsi:type="com.prismtech.spectra.sdr.sca2_2.models:UsesPort"
            instanceName="TX" name="Data"/>

        <ports xsi:type="com.prismtech.spectra.sdr.sca2_2.models:ProvidesPort" instanceName="RX"
            name="Data"/>

    </components>

</com.prismtech.spectra.sdr.sca2_2.models:Assembly>
A Graphical Domain-Specific Language

Images, layout, organization based on meta-model
Domain-Specific Constraints

Enforce structural composition, direction, etc.
Generators

Translate from declarative to imperative

VHDL

Code Coverage

C++

Test Cases
The Benefits

- The SCA
  - portability
  - standardized development
- Meta-model – Domain Specific Language
  - more productivity
- GDSL
  - easy to use and communicate to others
- Constraints
  - left shift defects from run-time to modeling time
- Generators
  - Productivity
  - Correctness
  - Architectural consistency
  - Requirements traceability
  - Synchronization of software artifacts
    - e.g. documentation
  - Automated testing = increased robustness
Contact Info

Stop by our booth at this conference

www.prismtech.com

Dominick - dp@prismtech.com

Bruce – bt@prismtech.com