Minimal Modeling Language Specification

By George Craig

MML Example:

Creates objects: `BaseObject, Events, EventsDetails, Contacts.`

```
Events > BaseObject
    *EventsDetails
    <>EventsDetails
    -name
    +name
    -description
    :bool verbose
    :long attempts
    ::DateTime startDate ;language specific target, limits output, but warns during generation

Contacts > BaseObject
```

Usage:

```
Generate code:
mml -o python|java|c++ MyModel.mml

Generate diagram:
mml -d MyModel.mml

Options:
-o output language
-d generate diagram
```

Language Specic Example (Python):

```
Events > BaseObject
    EventDetails[py:class]
    Name
    Description
```

Will output Python code:

```
class Events(BaseObject):

    __init__(self, event_details, name, description)
        self.event_details = event_details
        self.name = name
        self.description = description

    __tos__(self):
        return "foo"

    __run__(self):
        pass
```

Consider:

- typed vs. dynamic languages
- for languages that want a specific case: will autotranslate between camelCase and regular c/python
- no class variable support for now, unless I come up with another symbol

Language specific markup:

```
[py:class]
[py:instance]
[java:static]
[pat:proxy]
[pat:visitor] (pattern)
```

py = python
java = java

DSL Header Commands in MML:

```
generate web layer use bottle {framework | toolset | pattern} e.g., bottle, appfuse, front controller
generate db layer use dao
generate business layer
```

Full Specification:

```
Events{required class Name} >{optional inheritance} BaseObject{optional inherited class name}
    *{optional multiplicity}EventsDetails{optional instance variable}
    <>{optional aggregation}EventsDetails{optional instance variable}
    -{optional private scope}name
    +{optional public scope}name
    #{optional protected scope}name
    -description
    :bool{optional type}verbose
    :long{optional type}attempts
    ::DateTime:bool{optional language specific type} startDate ;language specific target, limits output, put out a warning when
generating
    process_business(){optional function indicator}
    =
        print("Hello, World"){optional implementation code}
    =
    process_ranges(:long{optional parameter type}description)
```

```
Contacts > BaseObject
```

Legend:

```
{} designates whether model or symbols are required or optional.
> inheritance
<> aggregation, if available
* multiplicity, results in collection container
- private property
+ public property
# protected property
: type designator for generating typed language output
; inline comment(s)
:: language specific type designator
= provides ability to add implementation code
```

Specification Notes:

1. DSL (Domain Specific Language) will help taylor to specific target, but not required.
2. DSL will help generate scaffolding, but not required
3. Convention over configuration
4. UML - Unified Modeling Language
5. MML - Minimal Modeling Language (inspired by Markdown)
6. Write the spec in Markdown, generate HTML, PDF
7. Diagram needed to show outputs to: dynamic languages, typed languages, java, python – ala a mind map – many options
8. How to designate additional relationships. I.e., dependencies
9. Instance variables by default
10. No types, on a typed language will result in code: Object description. So it can at least compile.
11. Hooks for `preGenereate()`, `postGenerate()` during the code generation process.
12. Plugins for user defined scripting.
13. Leverage existing code generators: appfuse, rails, etc.
14. Provide easy layer to call into MML to be leveraged by above code generation tools.
15. Generate simple unit, functional, and load tests(?)
16. Create a Markdown of What, Why, How, MML came to be and how to use.
17. Filetypes:
    1. .mml = minimal modeling language (speficies the models)
    2. .msl = modeling specific language (DSL included in .mml or separate file)
18. Need table lookup for reg-ex rules; consisting of key = rule, value = lambda of processing
19. Add Domain specific datatypes ala Sparky
20. Provide a simple diagramming tool. Initially output a static diagram to showcase model.
21. Leverage diff tools to re-run mml on existing code bases. I.e., full round trip engineering.

Syntax Highlighting:

Provide syntax highlighting, for MML files, for editors:

- Notepad++
- EditPlus
- Sublime