Translating Executable Software Models with *micca*

Andrew Mangogna
Model Realization
24th Annual Tcl/Tk Conference
October 16-20, 2017
Houston, TX

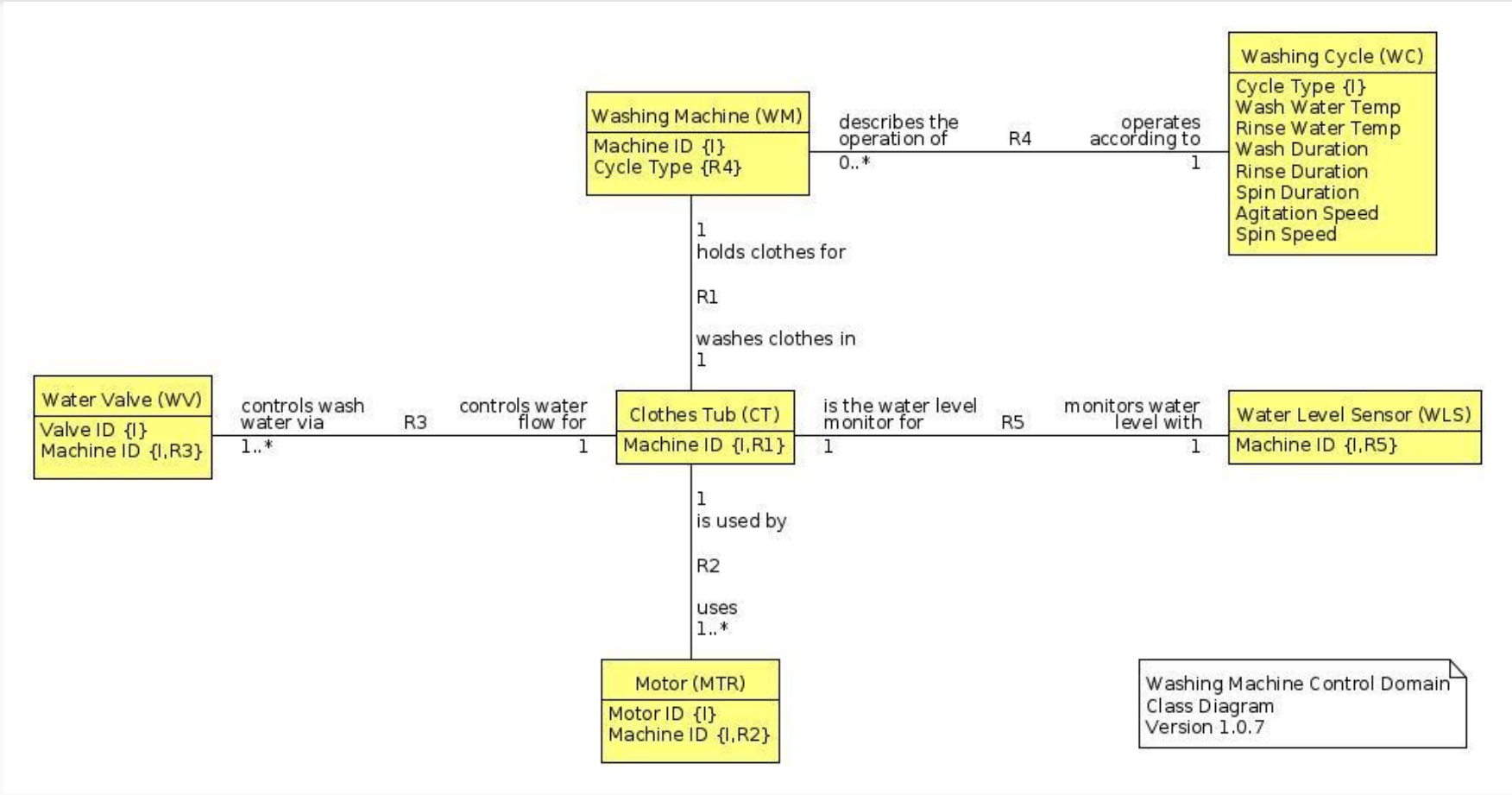
Introduction

- Micca is a program to aid in translating executable software models
  - Micca targets embedded platforms and small to medium scale POSIX systems and uses “C” as the implementation language.
  - Follow on to pycca (presented at the 2010 Tcl/Tk conference).
  - Micca is built using rosea (presented at the 2015 Tcl/Tk conference).
- Presentation today is focused on Tcl features used to implement micca.
  - Micca is approx. 8600 lines of Tcl code, plus approx. 475 lines of PEG grammar.
- Example model of an automatic washing machine.
  - The example is fully worked out in the micca documentation.

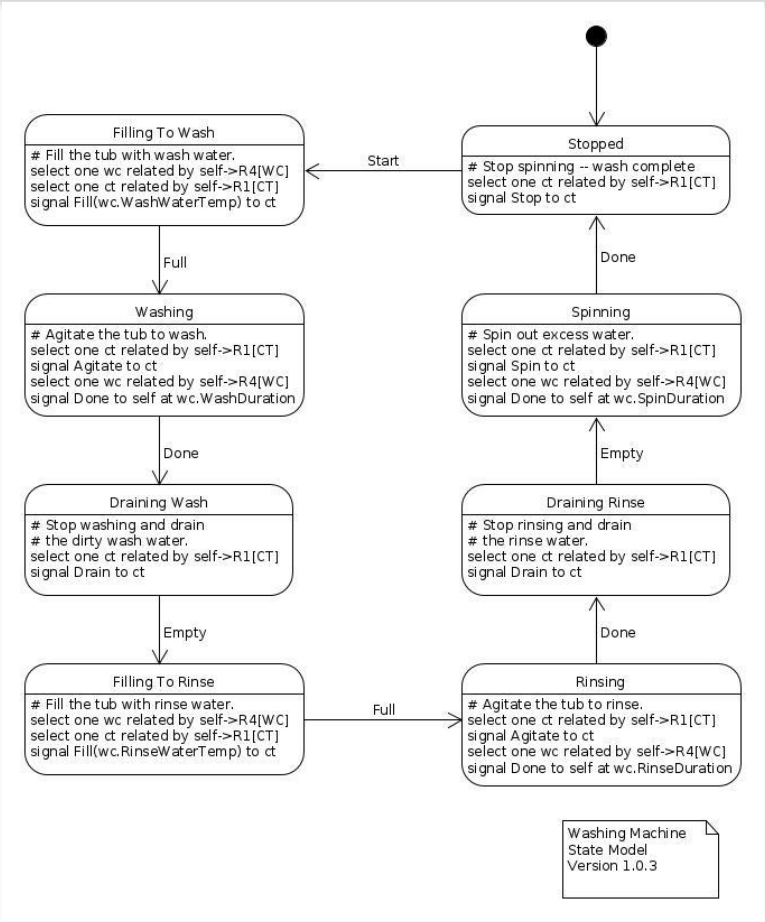
Tcl Features used in micca

- Domain specific language for specification
- Relationally structured data
- Parsing with PEGs
- Code generation by template expansion

Example Model



Washing Machine state model



Washing Machine domain specification

```
domain wmctrl {
    class WashingMachine {
        attribute MachineID {char[32]}
        statemodel {
            transition Stopped - Start -> FillingToWash
            transition FillingToWash - Full -> Washing
            # ... and other transition commands
            state Stopped {} {
                // "C" code for the Stopped state
            }
            # ... and other state commands
        }
    }
    class WashingCycle {
        attribute CycleType {char[32]}
        attribute WashWaterTemp WaterTemp_t
        attribute RinseWaterTemp WaterTemp_t
        attribute WashDuration unsigned
        attribute RinseDuration unsigned
        attribute SpinDuration unsigned
        attribute AgitationSpeed WashSpeed_t
        attribute SpinSpeed WashSpeed_t
        # ... and other Washing Cycle class properties
    }
    association R4 WashingMachine 0..*--1 WashingCycle
    # ... and the specification of the other classes in the diagram
}
```

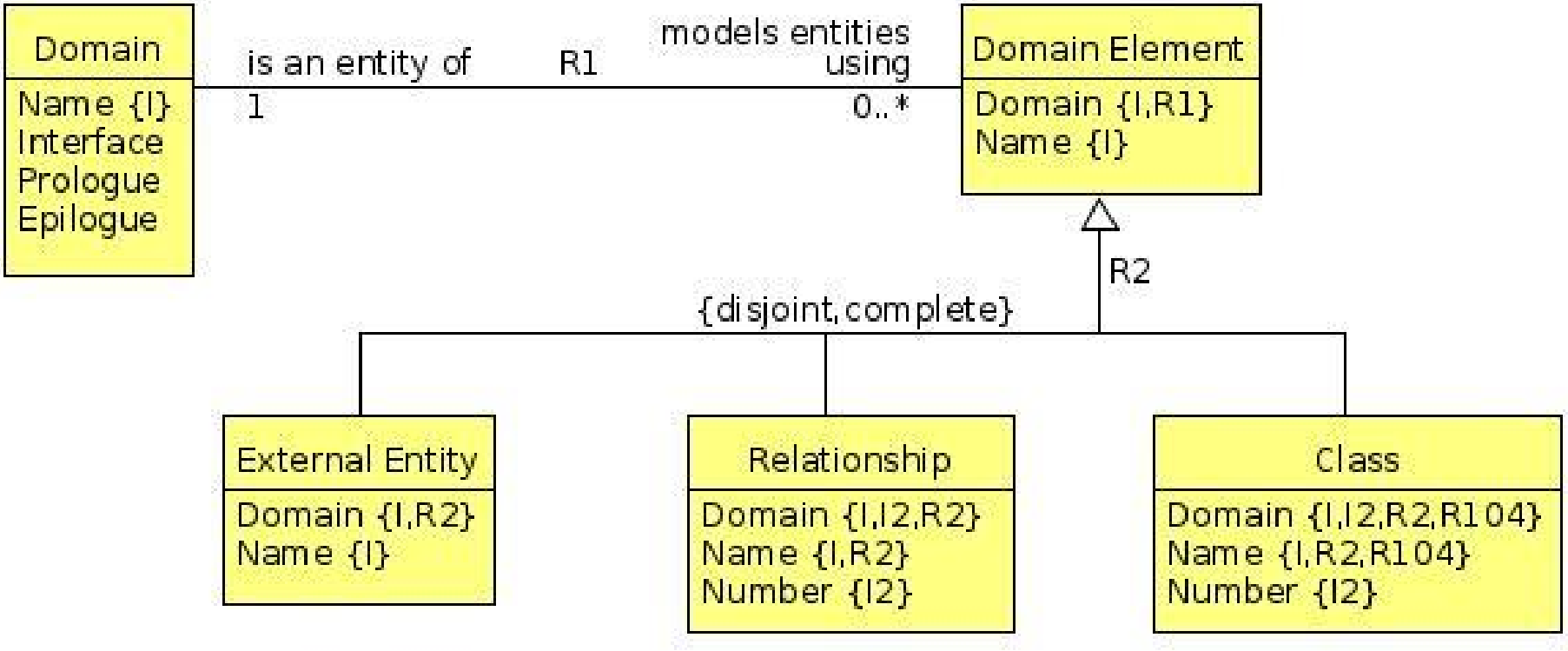
Micca DSL

- The DSL is a valid Tcl script.
- DSL commands, by design, are declarative in nature.
- Namespaces are used to insure DSL commands resolve properly.
- DSL commands are evaluated line-by-line, using **info complete** to determine command boundaries, to do better error handling.

Relationally structured data

- Micca is a rosea based application
  - Rosea presented at the 2015 Tcl/Tk conference
  - Rosea is to Tcl as micca is to “C”
  - Yes, I eat my own dog food!
- Micca DSL is a text-based interface to the underlying platform model.
  - Populating the platform model is done in a single transaction to insure consistent data.
  - There are 86 classes and 78 relationships in the micca platform model.
- Relational integrity checks insure data consistency is achieved.
  - Declarative constraints require no additional code.

Micca Platform Specific Model



Micca platform model encoded in Rosea DSL

```
class Domain {
    attribute Name string -id 1\
        -check {[::micca::@Config@::Helpers::isIdentifier $Name]}
    attribute Interface string -default {}
    attribute Prologue string -default {}
    attribute Epilogue string -default {}
}
class DomainElement {
    attribute Domain string -id 1
    attribute Name string -id 1\
        -check {[::micca::@Config@::Helpers::isIdentifier $Name] &&\
            ![::regexp -- {__[A-Z]+\Z} $Name]}
    reference R1 Domain -link {Domain Name}
}
association R1 DomainElement 0..*--1 Domain
class Class {
    attribute Domain string -id 1 -id 2
    attribute Name string -id 1
    attribute Number int -id 2
    reference R2 DomainElement -link Domain -link Name
    reference R104 ValueElement -link Domain -link Name
}
```

Parsing C type names

- Micca has some knowledge of “C” type names.
  - For example, the code generator has to create variable declaration statements.
  - Parser tools in tcllib provide the parser generator.
- “C” types names have an inherent ambiguity.
  - “C” allows new type names to be introduced via the “typedef” statement.
  - Micca resolves the ambiguity using a naming convention.

```
typedef_name <-
    <upper> <alnum>* '_t' WHITESPACE /
    'MRT_' <alnum>+ WHITESPACE /
    TYPENAME LPAREN identifier RPAREN ;
```

PEG for “C” type names

PEG datatype (type_name)
type_name <- specifier_qualifier_list abstract_declarator? EOF ;
abstract_declarator <- pointer? direct_abstract_declarator / pointer ;
direct_abstract_declarator <- direct_abstract_declarator_head direct_abstract_declarator_tail* ;
direct_abstract_declarator_head <- LPAREN abstract_declarator RPAREN / direct_abstract_declarator_tail ;
direct_abstract_declarator_tail <- array_declarator / LPAREN parameter_type_list? RPAREN ;
pointer <- (STAR type_qualifier_list?)+ ;

- Plus many, many more production rules.
- Derived from a full C99 PEG written by Ian Piumarta.

AST for “int (*)(void)” type

```
<type_name> :: 0 12
    <specifier_qualifier_list> :: 0 3
        <type_specifier> :: 0 3
            <int> :: 0 3
    <abstract_declarator> :: 4 12
        <direct_abstract_declarator> :: 4 12
            <direct_abstract_declarator_head> :: 4 6
                <LPAREN> :: 4 4
                <abstract_declarator> :: 5 5
                    <pointer> :: 5 5
                        <STAR> :: 5 5
                <RPAREN> :: 6 6
            <direct_abstract_declarator_tail> :: 7 12
                <LPAREN> :: 7 7
                <parameter_type_list> :: 8 11
                    <parameter_list> :: 8 11
                        <parameter_declaration> :: 8 11
                            <declaration_specifiers> :: 8 11
                                <type_specifier> :: 8 11
                                    <void> :: 8 11
                <RPAREN> :: 12 12
```

Code generation by template expansion

- Two types of code generation
  - Data structures and initialized variables
  - Activity code for model level operations
- Micca uses **::textutil::expander** from tcllib to perform the code generation.
  - Two different expander instances for the two types of code generation
- Expanding a template allows the generated code to be ordered properly.

Header file template

```
set headerTemplate {
    <%banner%>
    #ifndef <%headerFileGuard%>
    #define <%headerFileGuard%>
    #include "micca_rt.h"
    #include <assert.h>
    <%interface%>
    <%interfaceTypeAliases%>
    <%domainOpDeclarations%>
    <%externalOpDeclarations%>
    <%eventParamDeclarations%>
    <%portalIds%>
    <%portalDeclaration%>
    #endif /* <%headerFileGuard%> */
}
```

Operation Declarations

6 Find the parameters of the Domain Operation by traversing the R6 relationship. In the micca platform model, R6 associates a Domain Operation to zero or more formal Domain Operation Parameters.

7 This series of commands creates a relation value with the data needed to generate a function declaration.

22 The resulting declaration is created from data obtained by the query over each operation.

```
proc domainOpDeclarations {} {
    variable domain
    set result [comment "Domain Operations External Declarations"]

    set opRefs [DomainOperation findWhere {$Domain eq $domain}]
    set params [deRef [findRelated $opRefs ~R6]]
    set ops [pipe {
        deRef $opRefs |
        relation project ~ Domain Name ReturnDataType Comment |
        relation rename ~ Name Operation |
        ralutil::rvajoin ~ $params Parameters
    }]

    relation foreach op $ops {
        relation assign $op
        if {$Comment ne {}} {
            append result [comment $Comment]
        }
        set plist [relation list $Parameters DataType -ascending Number]
        set pdecl [expr {[llength $plist] == 0 ?\
                "void" : [join $plist {, }]}]
        append result "extern $ReturnDataType\
                ${Domain}_${Operation}\($pdecl\) ;\n"
    }

    return $result
}
```

Domain operation declarations

```
/*
 * Domain Operations External Declarations
 */
extern int wmctrl_createWasher(char const *) ;
extern bool wmctrl_deleteWasher(char const *) ;
extern bool wmctrl_startWasher(char const *) ;
extern void wmctrl_selectCycle(char const *, char const *) ;
extern void wmctrl_init(void) ;
```

Summary

- None of the ideas in micca is particularly novel.
  - Constructing DSLs as Tcl commands using namespaces.
  - Structuring complicated data models using relational techniques.
  - Parsing “C” type names using PEGs
  - Generating “C” code using template expansion.
- Micca is structured similar to a database CRUD application.
  - Populate a data model.
  - Generate a report from the data.
- All is done in Tcl.

Resources

- Micca is freely available
  - Same license as Tcl/Tk
  - Model Realization Tools
  - Chisel app (mrtools)

Micca and rosea resources

- Literate program document http://repos.modelrealization.com/cgi-bin/fossil/mrtools/doc/trunk/micca/doc/micca.pdf
- http://repos.modelrealization.com/cgi-bin/fossil/mrtools
- http://chiselapp.com/user/mangoa01/repository/mrtools

TclRAL resources

- http://repos.modelrealization.com/cgi-bin/fossil/tclral
- http://chiselapp.com/user/mangoa01/repository/tclral

Questions?

Andrew Mangogna
Model Realization
amangogna@modelrealization.com