# Taking a peek inside the cmlFramework

E.D. (Eric) Schabell
Institute for Computing and Information Sciences (ICIS)
Radboud University Nijmegen
http://www.schabell.com

**Abstract**

This paper will walk you through a demonstration run of the Collaborative Modeling Lab (CML) Framework prototype. This will start with a conceptual overview of the component structure. The reader will then be walked through a processing run in which a Niam Normal Form (NNF) grammar is used to *Filter* a *Logbook*, produce a *Contract*, *Distill* this into an *Essence*, and finally using the *Builder* to generate an eventual *Model* in ORM.

## 1 Introduction

The *Collaborative Modeling Lab Framework (cmlFramework)* prototype is the result of a desire to make a start at providing the IRIS department with a model generation validation playground. This initial *cmlFramework* is based closely on the groundwork laid out in [Bv05] and will provide further validation to continue the evolution of our research. This is an ongoing project which will require close attention to continue to provide a generic and flexible component architecture for the main *IRIS* research lines[IRI07].

This paper will outline the *cmlFramework* and provide the reader with a global view of the components and their interaction. This interaction will be discussed in enough detail to provide the reader with a conceptual understanding of how the *cmlFramework* processes a given *logbook*.

## 2 The framework

The *cmlFramework* is a Java application, written to be run in the *Eclipse* framework[Ecl07]. There has been very little work to date on the user interface as such, assuming the user will be able to run the *CmlManager* from within an *Eclipse* setup. Some of the other project dependencies are the java connector for *MySql* and for the *Antlr* parser generator project[Ant07]. The source code for this project is availabel online[Sub07] and includes api documentation in the form of *javadocs*.

The *cmlFramework* consists of several components that have been defined to work as independently as possible. The central component in this architecture is the *CmlManager*, which coordinates all work and message flow between the other components. This class contains the *main* method which runs as the

frameworks entry point. It takes one parameter as input and that is the name of the logbook file to be processed.

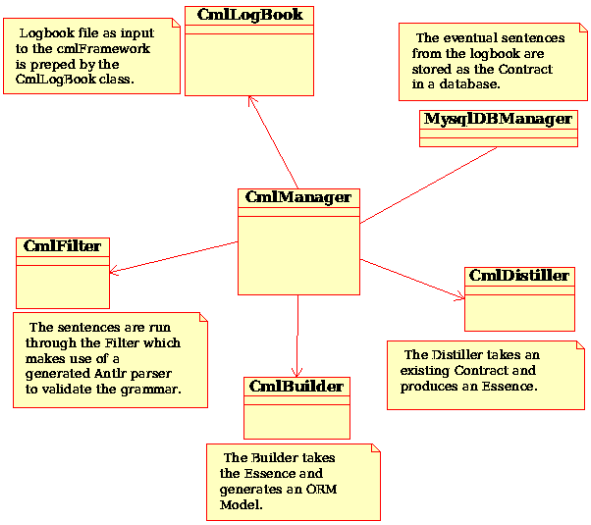


Figure 1: The cmlFramework architecture.

The rest of this paper will discuss the components managed by the *CmlManager*, see Figure 1, as they process the example logbook data.

# 3 Filtering a contract

The *CmlLogBook* components is responsible for the initial collection of the data from the given logbook file. This class arranges and makes it possible to provide the next step with the grammar sentences for parsing and validation.

We assume a certain format for our logbook files, as can be viewed in Section 5. It can therefore be noted that should a logbook input file deviate from the current format that one would need to refactor this class.

Beginning with a structured request from the *CmlLogBook*, the *CmlFilter* class then makes use of the generated parser for sentence validation. This parse is generated from our simple NNF grammar as can be seen here in *Antlr* notation:

```
start    : sentence ;
sentence :
labelTYPE LEFTPAR ROLENAME standard_name
COMMA ROLENAME standard_name RIGHTPAR ;
standard_name : ENTITYTYPE PRONOUN labelTYPE label ;

// Specify terminals here.
//
QUOTE         : '"' ;
LEFTPAR       : '(' ;
```

```
RIGHTPAR      : ')' ;
UNDERSCORE    : '_' ;
LT            : '<' ;
GT            : '>' ;
COMMA         : ',' ;

ROLENAME : UNDERSCORE ('a'..'z')+ (UNDERSCORE ('a'..'z')+)* ;

ENTITYTYPE    : ('A'..'Z')+ ;
PRONOUN       : LT ('a'..'z')+ GT ;
labelTYPE     : ('A'..'Z') ('a'..'z')+ ;
label         : QUOTE labelTYPE QUOTE ;

WS : (' ' | '\t' | '\r' | '\n') { $setType(Token.SKIP); } ;
```

All *cmlFramework* grammar files need to define a **start** entry point for the grammar. This is integrated into the filtering process and allows for seemless swapping of grammars in the framework. As an aid to help the reader visualize the process we include the following diagram is as follows:

```
Logbook -> Filter -> Contract
```

This process ends with a set of sentences that are valid and either in or out of the current running model, known as our *Contract*.

# 4 Distilling an essence

The *CmlDistiller* picks up the existing *Contract* and distills an *Essence*. This consists of the various components that can be given to the next step for model building based on the given *CmlBuilder*. Again, as an aid to help the reader visualize the process, we include the following diagram is as follows:

```
Contract -> Distiller -> Essence -> Builder -> Model
```

This component makes use of the same generated parser, but it has been modified to include Builder elements leading to a model. Currently this process leads directly to the ORM model of the current existing model from our previously compiled *Contract*.

# 5 Demo results

A sample of the demo running as described above is included here to complete this papar:

```
The contents of the Cml logbook file CmlLogBookNNF.log are:

1: [2006-08-10 11:23:12] [part1] [prop]   [S1]
[Inhabitance ( _lives_in PERSON <with> Name "John" ,
_has_inhabitant CITY <with> Name "Nijmegen" )]
2: [2006-08-10 11:23:56] [part2] [accept] [S1] []
```

```
3: [2006-08-10 11:24:34] [part1] [prop]   [S2]
[Experience ( _has_age PERSON <with> Name "John" ,
_is_of AGE <of> Years "Forty" )]
4: [2006-08-10 11:25:04] [part2] [reject] [S2] []
5: [2006-08-10 11:27:56] [part2] [prop]   [S3]
[Inhabitance ( _lives_in PERSON <with> Name "Billy" ,
_has_inhabitant CITY <with> Name "Utrecht" )]
6: [2006-08-10 11:28:34] [part1] [accept] [S3] []
7: [2006-08-10 11:29:56] [part2] [prop]   [S4]
[Inhabitance ( _lives_in PERSON <with> Name "Mary" ,
_has_inhabitant CITY <with> Name "Amsterdam" )]
8: [2006-08-10 11:29:56] [part2] [prop]   [S5]
[Loving ( _loves PERSON <with> Name "Mary" ,
_loved_by PERSON <with> Name "Billy" )]
9: [2006-08-10 11:30:34] [part1] [accept] [S5] []
10: [2006-08-10 10:50:34] [part1] [reject] [S6] []
11: [2006-08-10 10:51:34] [part1] [withdr] [S6] []
12: [2006-08-10 10:52:34] [part1] [accept] [S6] []
13: [2006-08-10 11:27:56] [part2] [prop]   [S4]
[Borking ( PERSON Name "John" has AGE <of>
Years "Thirty" )]
14: [2006-08-10 11:48:34] [part1] [withdr] [S3] []


Processing log line: [2006-08-10 11:23:12] [part1] [prop] [S1]
[Inhabitance ( _lives_in PERSON <with> Name "John" ,
_has_inhabitant CITY <with> Name "Nijmegen" )]
Parsed OK: Inhabitance ( _lives_in PERSON <with> Name "John" ,
_has_inhabitant CITY <with> Name "Nijmegen" )

*** The current model is complete at this time. ***

Processing log line: [2006-08-10 11:23:56] [part2]
[accept] [S1] []
Parsed OK:

*** The current model is complete at this time. ***

Processing log line: [2006-08-10 11:24:34] [part1] [prop] [S2]
[Experience ( _has_age PERSON <with> Name "John" ,
_is_of AGE <of> Years "Forty" )]
Parsed OK: Experience ( _has_age PERSON <with> Name "John" ,
_is_of AGE <of> Years "Forty" )

*** The current model is complete at this time. ***

Processing log line: [2006-08-10 11:25:04] [part2]
[reject] [S2] []
Parsed OK:
Bumped S2 out of the current model.
```

```
Processing log line: [2006-08-10 11:27:56] [part2] [prop] [S3]
[Inhabitance ( _lives_in PERSON <with> Name "Billy" ,
_has_inhabitant CITY <with> Name "Utrecht" )]
Parsed OK: Inhabitance ( _lives_in PERSON <with> Name "Billy" ,
_has_inhabitant CITY <with> Name "Utrecht" )

Processing log line: [2006-08-10 11:28:34] [part1]
[accept] [S3] []
Parsed OK:

Processing log line: [2006-08-10 11:29:56] [part2] [prop] [S4]
[Inhabitance ( _lives_in PERSON <with> Name "Mary" ,
_has_inhabitant CITY <with> Name "Amsterdam" )]
Parsed OK: Inhabitance ( _lives_in PERSON <with> Name "Mary" ,
_has_inhabitant CITY <with> Name "Amsterdam" )

Processing log line: [2006-08-10 11:29:56] [part2] [prop] [S5]
[Loving ( _loves PERSON <with> Name "Mary" ,
_loved_by PERSON <with> Name "Billy" )]
Parsed OK: Loving ( _loves PERSON <with> Name "Mary" ,
_loved_by PERSON <with> Name "Billy" )

Processing log line: [2006-08-10 11:30:34] [part1]
[accept] [S5] []
Parsed OK:

Processing log line: [2006-08-10 10:50:34] [part1]
[reject] [S6] []
Parsed OK:

Processing log line: [2006-08-10 10:51:34] [part1]
[withdr] [S6] []
Parsed OK:

Processing log line: [2006-08-10 10:52:34] [part1]
[accept] [S6] []
Parsed OK:

Processing log line: [2006-08-10 11:27:56] [part2] [prop] [S4]
[Borking ( PERSON Name "John" has AGE <of>
Years "Thirty" )]
Parse FAILED: Borking ( PERSON Name "John" has
AGE <of> Years "Thirty" )
Has been REJECTED by the Grammar Parser...

This entry already exists in the database,
continuing with the next entry...
Problem inserting line into the statements table!
This entry already exists in the database,
```

```
continuing with the next entry...
Problem inserting line into the participants table!

Processing log line: [2006-08-10 11:48:34] [part1]
[withdr] [S3] []
Parsed OK:
Bumped S3 out of the current model.


Resulting model (accepted by all participants):

ID: TIME:
=== =====
S1 2006-08-10 11:23:12.0
S4 2006-08-10 11:29:56.0
S5 2006-08-10 11:29:56.0


Building the current model:

*******************************
orm-entity-type(PERSON)
orm-label-type(Name)
orm-entity("John")
orm-label("John")
orm-type("John",Name)
*******************************
orm-entity-type(CITY)
orm-label-type(Name)
orm-entity("Nijmegen")
orm-label("Nijmegen")
orm-type("Nijmegen",Name)
*******************************
orm-relation-type(Inhabitance)
orm-relation(_lives_in)
PERSON
<with>
Name
"John"
*******************************
orm-relation-type(Inhabitance)
orm-relation(_has_inhabitant)
CITY
<with>
Name
"Nijmegen"
*******************************
Distilled OK: Inhabitance ( _lives_in PERSON <with>
Name "John" , _has_inhabitant CITY <with> Name "Nijmegen" )
```

```
*******************************
orm-entity-type(PERSON)
orm-label-type(Name)
orm-entity("Mary")
orm-label("Mary")
orm-type("Mary",Name)
*******************************
orm-entity-type(CITY)
orm-label-type(Name)
orm-entity("Amsterdam")
orm-label("Amsterdam")
orm-type("Amsterdam",Name)
*******************************
orm-relation-type(Inhabitance)
orm-relation(_lives_in)
PERSON
<with>
Name
"Mary"
*******************************
orm-relation-type(Inhabitance)
orm-relation(_has_inhabitant)
CITY
<with>
Name
"Amsterdam"
*******************************
Distilled OK: Inhabitance ( _lives_in PERSON <with>
Name "Mary" , _has_inhabitant CITY <with> Name "Amsterdam" )

*******************************
orm-entity-type(PERSON)
orm-label-type(Name)
orm-entity("Mary")
orm-label("Mary")
orm-type("Mary",Name)
*******************************
orm-entity-type(PERSON)
orm-label-type(Name)
orm-entity("Billy")
orm-label("Billy")
orm-type("Billy",Name)
*******************************
orm-relation-type(Loving)
orm-relation(_loves)
PERSON
<with>
Name
"Mary"
*******************************
```

```
orm-relation-type(Loving)
orm-relation(_loved_by)
PERSON
<with>
Name
"Billy"
******************************
Distilled OK: Loving ( _loves PERSON <with> Name
"Mary" , _loved_by PERSON <with> Name "Billy" )
```

Development is ongoing and will be reported as appropriate milestones are achieved in either technical reports or as integral validation reporting in other departmental papers.

# References


[Ant07] Antlr project team. *Antlr parser generator*, 2007.

[Bv05] S. Bosman and Th.P. van der Weide. Towards formalization of the information modeling dialog. Technical Report ICIS–R05012, Computing Science Institute, University of Nijmegen, 2005.

[Ecl07] Eclipse project team. *Eclipse*, 2007.

[IRI07] IRIS department. *Collaborative Modeling Lab*, 2007.

[Sub07] Subversion project team. *Subversion code repository for cmlFramework*, 2007.