



# DSL – Brains & Tools

Addressing **Development Asynchronicity**  
Using **Domain Modeling**  
With a **Domain Specific Language**



Frank Pijpers

September 18, 2009

## Technology Contribution Sioux (WP3)

Apply Model-driven  
Technologies:

- Domain Specific Modeling,
  - Model-Driven Software Development)
- Facilitate
  - Co-design



# Today: DSL – Theory & Practice

## Domain Specific Language

A **language** designed to be useful for a specific set of tasks, as opposed to a general purpose language

- UML is a general purpose modeling language
  - Similar to general purpose programming languages
  - Can be seen as a collection of DSLs
  - Can be used to define a DSL (i.e. using profiles)

- MDA is a trademark of the OMG
  - A collection of standards
  - Models defined in MOF or UML, refined/constrained/queried (OCL), transformed (QVT), and used to generate text (MOF2Text), ...

- The UML metamodel can be the starting point of your DSM tool
  - How important are standards to you?
  - How much complexity do you need in a metamodel or language?

- Technical dialects (e.g. SQL)
- Notations (e.g. music score)
- Business languages
- Textual ↔ Graphic

```

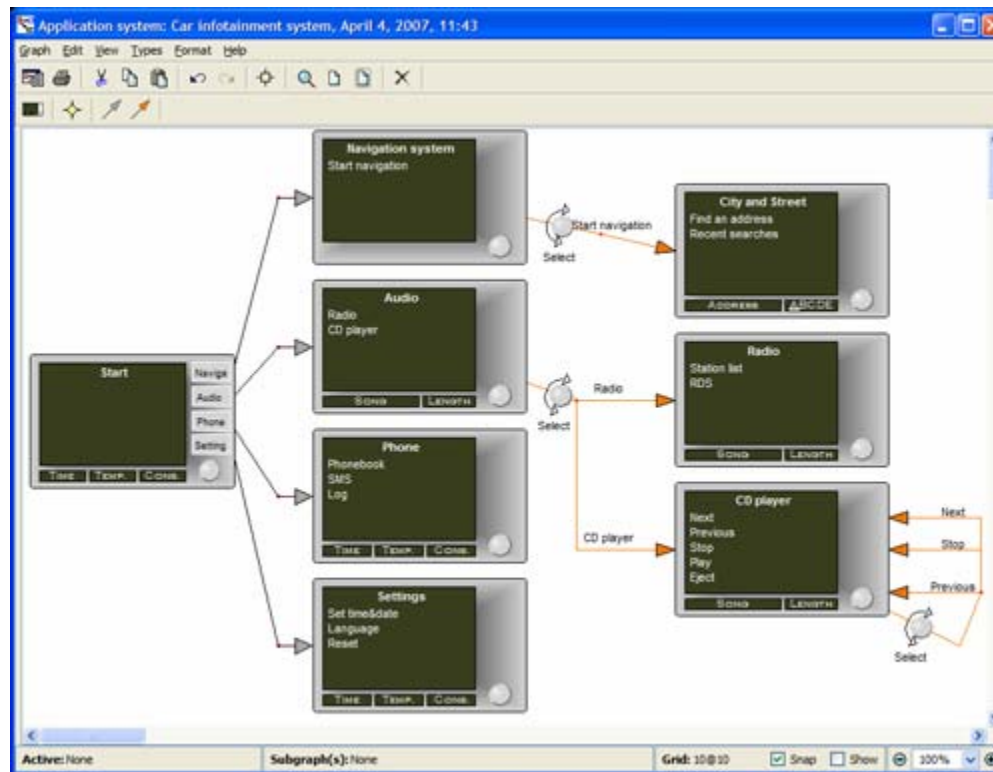
NAME VARCHAR(15) NOT NULL,
INIT CHARACTER,
NAME VARCHAR(15) NOT NULL,
INIT CHARACTER,
ENO CHARACTER(4),
DATE DATE,
CHARACTER(8)
VIL CHARACTER(4) NOT NULL,
CHARACTER,
HDATE DATE,
RY DECIMAL(9,2),
S DECIMAL(9,2),
DECIMAL(9,2),
ld01" DB2ADMIN.ADDRESS I INL
ld02" DB2ADMIN.T NAME,
keys */
TRAI NT PK_EMPLOYEE
IMARY KEY (EMPNC),
hecks */
'RAINT NUMBER
* (PHONE NO >= '0000' AND

```



# Example Graphical DSL (1/3)

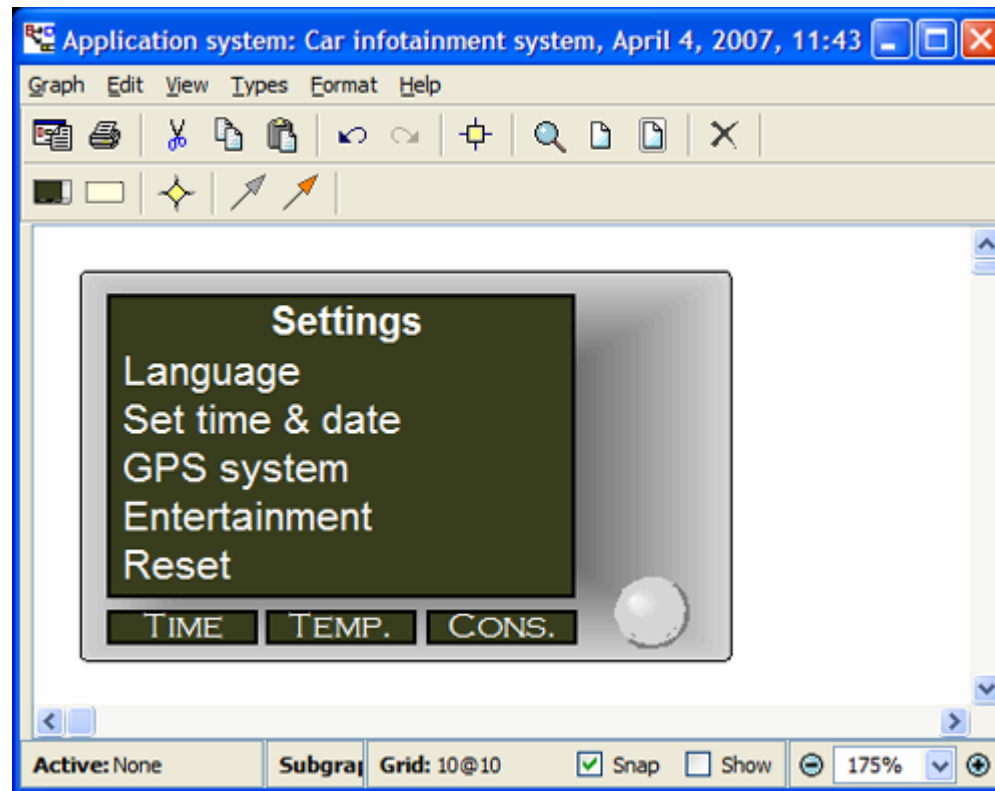
A DSM language language based on the AUTOSAR standard.



# Example Graphical DSL (2/3)

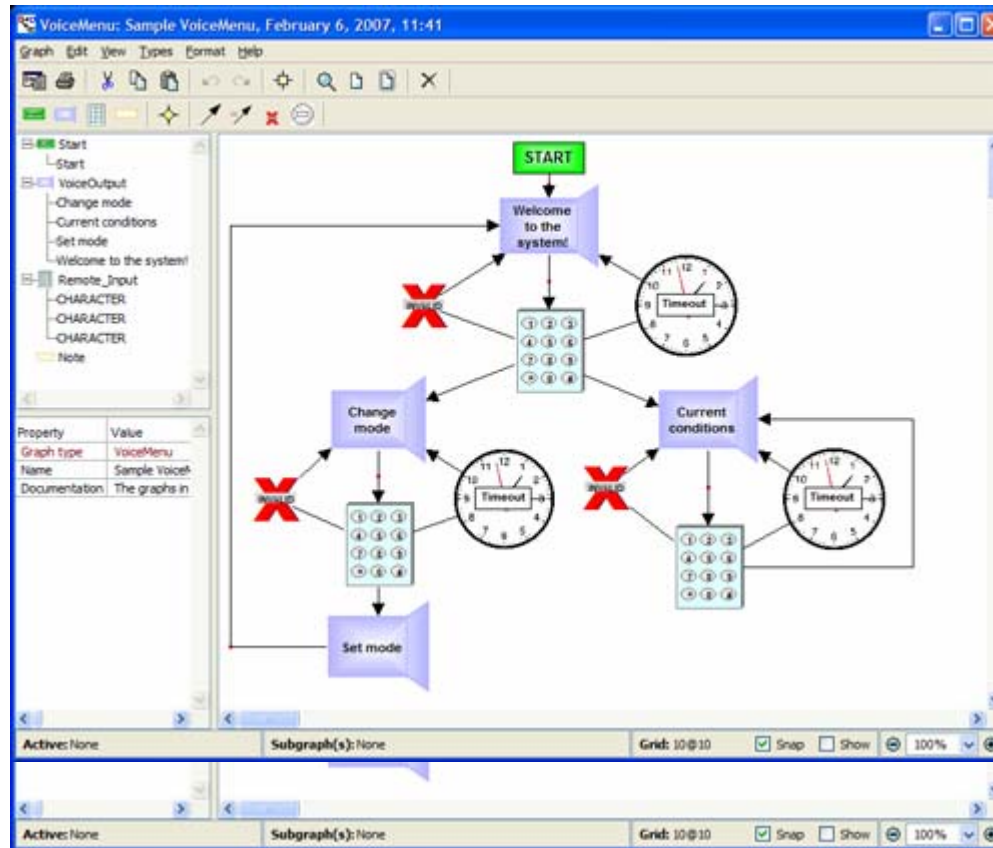


# Example Graphical DSL (3/3)

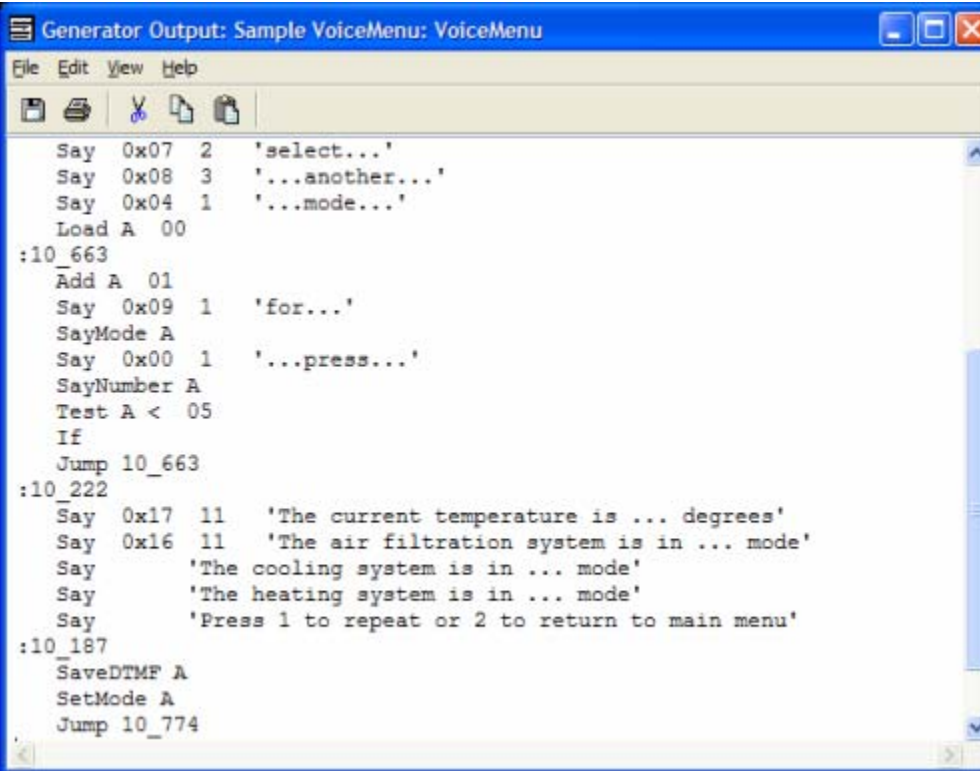


# Example Graphical DSL (1/2)

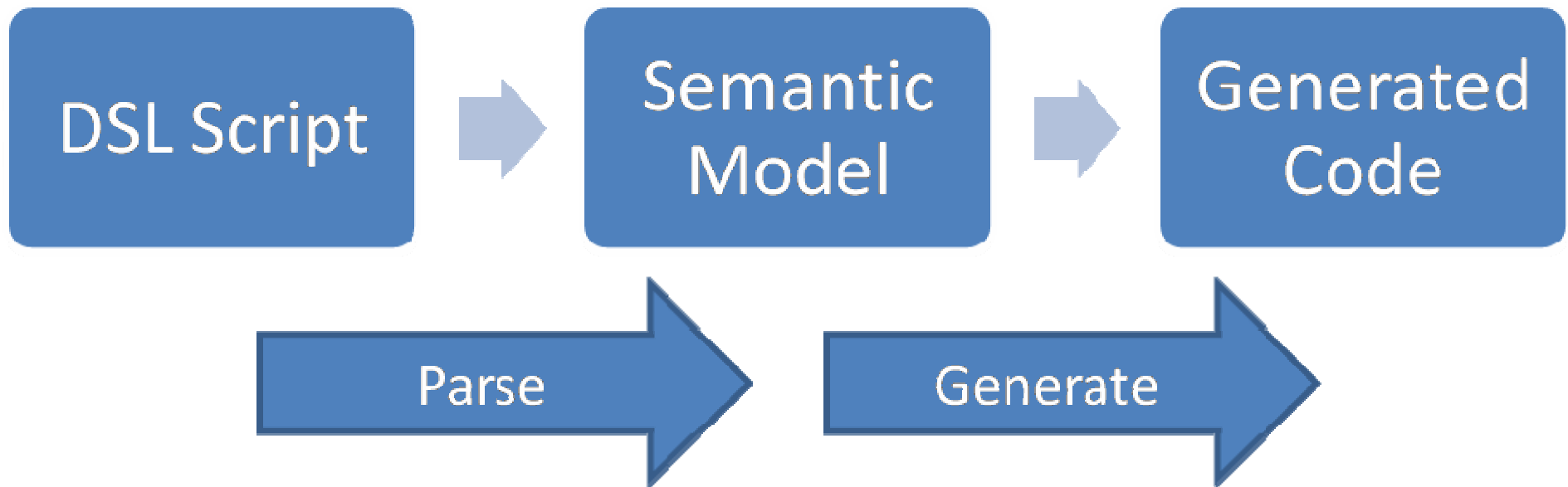
A voice menu system for an 8-bit microcontroller.



# Example Graphical DSL (2/2)



```
Generator Output: Sample VoiceMenu: VoiceMenu
File Edit View Help
Say 0x07 2 'select...'
Say 0x08 3 '...another...'
Say 0x04 1 '...mode...'
Load A 00
:10_663
Add A 01
Say 0x09 1 'for...'
SayMode A
Say 0x00 1 '...press...'
SayNumber A
Test A < 05
If
Jump 10_663
:10_222
Say 0x17 11 'The current temperature is ... degrees'
Say 0x16 11 'The air filtration system is in ... mode'
Say 'The cooling system is in ... mode'
Say 'The heating system is in ... mode'
Say 'Press 1 to repeat or 2 to return to main menu'
:10_187
SaveDTMF A
SetMode A
Jump 10_774
```



- A language to build, configure or do in your domain
- Not necessarily text, could be graphical
- Might reuse syntax of host language

- An in-memory representation of the subject the DSL describes
- Sometimes this is the AST
- Separates parser and generation

- Executable representation of the DSL
- Evaluation during parsing
- Interpretation of semantic model

- External
- Internal
- Workbenches

- DSLs that use a different syntax to the main language that uses them
- Examples
  - make, flex, yacc, bison
  - XPath, SQL, regexp
  - Sed, awk

- Pros & Cons
  - Pros
    - Language Designer
    - Tools for compiler construction can be used
    - Language User
    - Simpler to use than a GPL

- Pros & Cons
  - Cons
    - Language Designer
    - Weak tool support
    - Yet another language to learn
    - Often target to a particular GPL
    - Often difficult to closely integrate with GPL


## Style – Internal (1/3)

- DSLs that share the same syntax to the main language that uses them
- Examples
  - PetitParser (Smalltalk)
  - rake, rspec (Ruby)
  - jQuery (JavaScript)
  - Rpython
- A subset of the host language is used
- Popular on Lisp, Scheme, Ruby, Smalltalk and JavaScript

- Pros & Cons
  - Pros
    - Language Designer
      - No special tools required
      - No new grammar required
    - Language User
      - Intermixable with GPL
      - Tools continue to work
      - No new language to learn

- Pros & Cons
  - Cons
    - Limited expressivity
    - Unnecessary syntactic noise
    - Constrained by host language

## Today's Language Workbenches

Today's Language Workbenches		
Intentional Software	<a href="http://intentsoft.com/">http://intentsoft.com/</a>	
Meta-Programming System	<a href="http://www.jetbrains.com/mps">http://www.jetbrains.com/mps</a>	
Software Factories	<a href="http://www.softwarefactories.com/">http://www.softwarefactories.com/</a>	
Model Driven Architecture (MDA)	<a href="http://www.omg.org/mda/">http://www.omg.org/mda/</a>	



# Basic Example

- **A**nother **T**ool for **L**anguage **R**ecognition  
Written (Java) by Terence Parr

Supported by ANTLRWorks  
graphical grammar editor and debugger  
Written by Jean Bovet (Swing)

Used to implement  
"real" programming languages  
domain-specific languages

<http://www.antlr.org>

- Uses EBNF grammars
  - Extended Backus-Naur Form
  - Can directly express optional and repeated elements
  - Supports subrules (parenthesized groups of elements)
- Supports many target languages for generated code
  - Java, Ruby, Python, Objective-C, C, C++, C#
- Provides infinite look-ahead
  - most parser generators don't
  - used to choose

- - LL(k) parsers are top-down parsers that
    - parse from **Left** to right
    - construct a **Leftmost** derivation of the input
    - look ahead k tokens
  - LR(k) parsers are bottom-up parsers that
    - parse from **Left** to right
    - construct a **Rightmost** derivation of the input
    - look ahead k tokens
  - LL parsers can't handle left-recursive rules
  - Most people find LL grammars easier

# Three Main Use Cases

- **Implementing validators**
  - generate code that validates that input obeys grammar rules
- **Implementing processors**
  - generate code that validates and processes input
    - could include performing calculations, updating databases, reading configuration files into runtime data structures, ...
- **Implementing translators**
  - generate code that validates and translates input into another format such as a programming language or byte code

- Lexer
  - Converts a stream of characters to a stream of tokens
- Parser
  - Processes a stream of tokens, possibly creating an AST
- Abstract Syntax Tree (AST)
  - An intermediate tree representation of the parsed input that
    - is simpler to process than the stream of tokens
    - can be efficiently processed multiple times
- Tree Parser
  - processes an AST*
- StringTemplate
  - a library that supports using templates with placeholders for outputting text (e.g. Java source)

- Write grammar
- Optionally: write StringTemplate templates
- Debug grammar with ANTLRWorks
- Generate classes from grammar
- Write application that uses generated classes
- Feed the application, text that conforms



Demo