Rapid Development of Dialogue Systems by Grammar Compilation

Björn Bringert, bringert@cs.chalmers.se
Department of Computer Science and Engineering, Chalmers University of Technology and Göteborg University

GF Grammar

```
abstract Pizza = {
  flags startcat = Order;
  cat Order; Number; Size; Topping;
  fun pizza        : Number → Size → Topping → Order;
      one, two     : Number;
      small, large : Size;
      cheese, ham  : Topping;
}

concrete PizzaEng of Pizza = open SyntaxEng, ParadigmsEng in {
  lincat Order = Utt; Number = Det;
         Size  = AP; Topping = NP;
  printname cat Order   = "What would you like to order?";
                Size    = "What size pizzas do you want?";
                Topping = "What topping do you want?";
  lin pizza n s t =
        let pizza_N = mkN "pizza";
            sp : CN = variants{mkCN s pizza_N; mkCN pizza_N}
        in mkUtt (mkNP n (variants{
              mkCN sp (mkAdv with_Prep t);
              sp
            }));
      one    = indefSgDet;
      two    = mkDet n2_Numeral;
      small  = mkAP (mkA "small");
      large  = mkAP (mkA "large");
      cheese = mkNP massQuant (mkN "cheese");
      ham    = mkNP massQuant (mkN "ham");
}
```

VoiceXML dialogue manager

```
<?xml version="1.0" encoding="utf-8"?>
<vxml version="2.0" xmlns="http://www.w3.org/2001/vxml">
  <form>
    <subdialog name="sub" src="#Order_cat">
      <param name="old" expr="{ name : '?' }" />
    </subdialog>
  </form>
  <!-- Number category. -->
  <form id="Number_cat">
    <var name="old" />
    <block cond="old.name != '?'">
      <assign name="term" expr="old" />
    </block>
    <field name="term">
      <prompt>Number</prompt>
      <grammar src="PizzaEng.gram#Number_cat" />
    </field>
    <!-- one : () Number -->
    <!-- two : () Number -->
    <block>
      <return namelist="term" />
    </block>
  </form>
  <!-- Order category. -->
  <form id="Order_cat">
    <var name="old" />
    <block cond="old.name != '?'">
      <assign name="term" expr="old" />
    </block>
    <field name="term">
      <prompt>What would you like to order?</prompt>
      <grammar src="PizzaEng.gram#Order_cat" />
    </field>
    <!-- pizza : (Number, Size, Topping) Order -->
    <subdialog name="pizza_0" src="#Number_cat" cond="te
      <param name="old" expr="term.args[0]" />
      <filled>
        <assign name="term.args[0]" expr="pizza_0.term"
      </filled>
    </subdialog>
    <subdialog name="pizza_1" src="#S
      <param name="old" expr="term.arg
      <filled>
        <assign name="term.args[1]" ex
      </filled>
    </subdialog>
    <subdialog name="pizza_2" src="#To
      <param name="old" expr="term.arg
      <filled>
        <assign name="term.args[2]" ex
      </filled>
    </subdialog>
    <block>
      <return namelist="term" />
    </block>
  </form>
  <!-- Size category. -->
  <form id="Size_cat">
    <var name="old" />
    <block cond="old.name != '?'">
      <assign name="term" expr="old" /
    </block>
    <field name="term">
      <prompt>What size pizzas do you
      <grammar src="PizzaEng.gram#Size
    </field>
    <!-- large : () Size -->
    <!-- small : () Size -->
    <block>
      <return namelist="term" />
    </block>
  </form>
  <!-- Topping category. -->
  <form id="Topping_cat">
    <var name="old" />
    <block cond="old.name != '?'">
      <assign name="term" expr="old" /
    </block>
    <field name="term">
      <prompt>What topping do you want
      <grammar src="PizzaEng.gram#Topp
    </field>
    <!-- cheese : () Topping -->
    <!-- ham : () Topping -->
    <block>
      <return namelist="term" />
    </block>
  </form>
</vxml>
```

SRGS+SISR speech recognition grammar + semantic interpretation

```
#ABNF 1.0 UTF-8;
meta "description" is "Speech recognition grammar for PizzaEng. Original start category: Order{}.s";
meta "generator" is "Grammatical Framework 2.7";
tag-format <semantics/1.0>;
root $Order_cat;

public $Number_cat = $PizzaEng_0 { $ = $PizzaEng_0 }
                   | $PizzaEng_1 { $ = $PizzaEng_1 } ;
public $Order_cat = $PizzaEng_2 { $ = $PizzaEng_2 } ;
public $Size_cat = $PizzaEng_3 { $ = $PizzaEng_3 } ;
public $Topping_cat = $PizzaEng_4 { $ = $PizzaEng_4 }
                    | $PizzaEng_5 { $ = $PizzaEng_5 } ;

// Number{.n=Pl;}.s
$PizzaEng_0 = (two)
            {!{ $ ={ name: "two", args:[]} }!} ;

// Number{.n=Sg;}.s
$PizzaEng_1 = (a | an)
            {!{ $ ={ name: "one", args:[]} }!} ;

// Order{}.s
$PizzaEng_2 = { var a =[] }
              ($PizzaEng_0 { a[0] = $PizzaEng_0 }
               $PizzaEng_3 { a[1] = $PizzaEng_3 } pizzas with
               $PizzaEng_5 { a[2] = $PizzaEng_5 }
               | $PizzaEng_1 { a[0] = $PizzaEng_1 }
                 $PizzaEng_3 { a[1] = $PizzaEng_3 } pizza with
                 $PizzaEng_5 { a[2] = $PizzaEng_5 })
              {!{ $ ={ name: "pizza", args:[a[0], a[1], a[2]]} }!}
```

JavaScript output realizer

```
var Pizza = new Abstract();
Pizza.addType("cheese",[], "Topping");
Pizza.addType("ham",[], "Topping");
Pizza.addType("large",[], "Size");
Pizza.addType("one",[], "Number");
Pizza.addType("pizza",["Number", "Size", "Topping"], "Order");
Pizza.addType("small",[], "Size");
Pizza.addType("two",[], "Number");
var PizzaEng = new Concrete(Pizza);
PizzaEng.addRule("__Number", function(cs){ return new Arr(new Int(2), new Seq());});
PizzaEng.addRule("__Order", function(cs){ return new Arr(new Seq());});
PizzaEng.addRule("__Size", function(cs){ return new Arr(new Int(2), new Arr(new Seq(), new Seq(), new Seq(),
    new Seq(), new Seq(), new Seq()));});
PizzaEng.addRule("__Topping", function(cs){ return new Arr(new Rp(new Int(6), new Arr(new Int(2), new
    Int(3))), new Arr(new Seq(), new Seq(), new Seq()));});
PizzaEng.addRule("_dNumber", function(cs){ return new Arr(new Int(0), cs[0]);});
PizzaEng.addRule("_dOrder", function(cs){ return new Arr(cs[0]);});
PizzaEng.addRule("_dSize", function(cs){ return new Arr(new Int(0), new Arr(cs[0], cs[0], cs[0], cs[0],
    cs[0], cs[0]));});
PizzaEng.addRule("_dTopping", function(cs){ return new Arr(new Rp(new Int(0), new Arr(new Int(0), new
    Int(0))), new Arr(cs[0], cs[0], cs[0]));});
PizzaEng.addRule("_0", function(cs){ return new Arr(new Int(0), new Int(2));});
PizzaEng.addRule("_1", function(cs){ return new Rp(new Int(2), PizzaEng.rule("_0", cs));});
PizzaEng.addRule("_2", function(cs){ return new Arr(new Str(""), new Str(""), new Str("'s"));});
PizzaEng.addRule("_7", function(cs){ return new Arr(new Str(""), new Str(""), new Str(""), new Str(""), new
    Str(""), new Str(""));});
PizzaEng.addRule("cheese", function(cs){ return new Arr(PizzaEng.rule("_1", cs), new Suffix("cheese",
    PizzaEng.rule("_2", cs)));});
PizzaEng.addRule("ham", function(cs){ return new Arr(PizzaEng.rule("_1", cs), new Suffix("ham",
    PizzaEng.rule("_2", cs)));});
PizzaEng.addRule("large", function(cs){ return new Arr(new Int(0), new Suffix("large", PizzaEng.rule("_7",
    cs)));});
PizzaEng.addRule("one", function(cs){ return new Arr(new Int(0), new Seq("a"));});
PizzaEng.addRule("pizza", function(cs){ return new Variants(new Arr(new Seq(cs[0].sel(new Int(1)), new
    Variants((new Arr(new Seq((new Arr(new Seq(cs[1].sel(new Int(1)).sel(new Rp(new Int(2), new Arr(new Int(0),
    new Int(2)))), new Str("pizza")), new Seq(new Str("pizza"), cs[1].sel(new Int(1)).sel(new Rp(new Int(2), new
    Arr(new Int(0), new Int(2)))))).sel(cs[1].sel(new Int(0))), new Seq(new Str("with"), cs[2].sel(new
    Int(1)).sel(new Int(1)))), new Seq((new Arr(new Seq(cs[1].sel(new Int(1)).sel(new Rp(new Int(5), new Arr(new
    Int(1), new Int(2)))), new Str("pizzas")), new Seq(new Str("pizzas"), cs[1].sel(new Int(1)).sel(new Rp(new
    Int(5), new Arr(new Int(1), new Int(2)))))).sel(cs[1].sel(new Int(0))), new Seq(new Str("with"),
    cs[2].sel(new Int(1)).sel(new Int(1))))).sel(cs[0].sel(new Int(0))),(new Arr(new Seq(new Str("pizza"), new
    Seq(new Str("with"), cs[2].sel(new Int(1)).sel(new Int(1))))), new Seq(new Str("pizzas"), new Seq(new
    Str("with"), cs[2].sel(new Int(1)).sel(new Int(1))))).sel(cs[0].sel(new Int(0)))), new Arr(new
    Seq(cs[0].sel(new Int(1)), new Variants((new Arr((new Arr(new Seq(cs[1].sel(new Int(1)).sel(new Rp(new
    Int(2), new Arr(new Int(0), new Int(2)))), new Str("pizza")), new Seq(new Str("pizza"), cs[1].sel(new
    Int(1)).sel(new Rp(new Int(2), new Arr(new Int(0), new Int(2)))))).sel(cs[1].sel(new Int(0))),(new Arr(new
    Seq(cs[1].sel(new Int(1)).sel(new Rp(new Int(5), new Arr(new Int(1), new Int(2)))), new Str("pizzas")), new
    Seq(new Str("pizzas"), cs[1].sel(new Int(1)).sel(new Rp(new Int(5), new Arr(new Int(1), new
    Int(2)))))).sel(cs[1].sel(new Int(0))))).sel(cs[0].sel(new Int(0))),(new Arr(new Str("pizza"), new
    Str("pizzas"))).sel(cs[0].sel(new Int(0)))))));});
PizzaEng.addRule("small", function(cs){ return new Arr(new Int(0), new Suffix("small", PizzaEng.rule("_7",
    cs)));});
PizzaEng.addRule("two", function(cs){ return new Arr(new Int(1), new Str("two"));});
```

Grammatical Framework (GF)

GF is a framework for developing high-precision natural language applications. It is based on **type theory** and **functional programming**. It supports **multilingual** and **multimodal** grammars, where a **semantic representation** is mapped into multiple languages or modalities.

There are currently GF grammar **libraries for 10 languages**: Danish, English, Finnish, French, German, Italian, Norwegian, Russian, Spanish, and Swedish. GF grammars can be compiled into many formats, including SRGS, GSL, JSGF, and SLF speech recognition grammars.

GF is **open source** and available from **http://gf.digitalgrammars.com/**

Information gathering dialogue = Proof editing in GF

$\Downarrow$

GF grammars can be compiled to VoiceXML dialogue systems

Web-based multimodal demo system

Example dialogue

?
S: What would you like to order?
U: a large pizza
order [pizza one large ?, ?]
S: What toppings would you like?
U: ham and cheese
order [pizza one large [ham,cheese], ?]
S: What else would you like?
U: a large beer
order [pizza one large [ham,cheese], drink one large beer, ?]
S: What else would you like?
U: nothing else
order [pizza one large [ham,cheese], drink one large beer]
S: Thank you for ordering one large pizza with ham and cheese and one large beer. Your order will be ready in two minutes.

References

Björn Bringert. 2007. Rapid Development of Dialogue Systems by Grammar Compilation. *SIGDIAL 2007*.

Aarne Ranta and Robin Cooper. 2004. Dialogue Systems as Proof Editors. *Journal of Logic, Language and Information*, 13(2).

CHALMERS | GÖTEBORG UNIVERSITY