A Development Environment for Multimodal Functional Unification Generation Grammars

David Reitter
Media Lab Europe
Dublin, Ireland
reitter@mle.media.mit.edu

1 Introduction

When grammar-based techniques for natural language generation (and analysis alike) find their way into collaborative projects or actual application, big grammars tend to become hard to extend and debug. The MUG system represents a new tool set with a graphical debugging environment for functional unification grammars, which is designed to help grammar developers inspect the results of their work.

The particular formalism supported is Multimodal Functional Unification Grammar (MUG, [4]), which is similar to Functional Unification Grammars (FUG: [2], [1]), but supports several coordinated modes, such as voice prompts or structural and/or language-based screen displays. For each input description, the grammar can generate a range of coherent realization variants, which are ranked by a scoring function in order to optimize the output towards situational and device-related factors.

2 Development with MUG

2.1 System overview

The MUG System is a development tool that consists of several components. The *MUG Formalism* is a grammar specification syntax. The *MUG Engine* handles the generation and adaptation process and offers interfaces to connect external components. *MUG Workbench* is a graphical development environment. The workbench works as an inspection tool, which runs a test case, generating all possible *variants* of the output, and then gives access to the various steps taken during content realization. We found this a faster method than the step-by-step execution in a graphical debugger.[1]

[1]The generation system including the workbench will be available shortly as open source, with documentation and for free.

2.2 MUG Formalism

The MUG formalism is close to Prolog syntax. One or more grammar files contain components, which are usually made up of one big FD (additional disjunctive or conjunctive unifications are allowed). Variables can be named and always start with an upper-case letter. The grammar writer is allowed to add detailed comments about components or their parts. MUG rules (*components*) are attribute-value matrices (AVM), which regularly have internally shared substructures propagating information between the different levels of linguistic representation. The grammar formalism implements this functionality by using named variables.

Unification-based grammars just like other object-oriented formalisms need to balance off the safety of strongly typed classes, which give error messages at an early stage, and the fact that no typing supports exploratory programming and quick prototyping. We address the issue with a type hierarchy that is used to issue non-fatal warnings in case of possibly ill-formed substructures in dialogue act input and grammar rules.

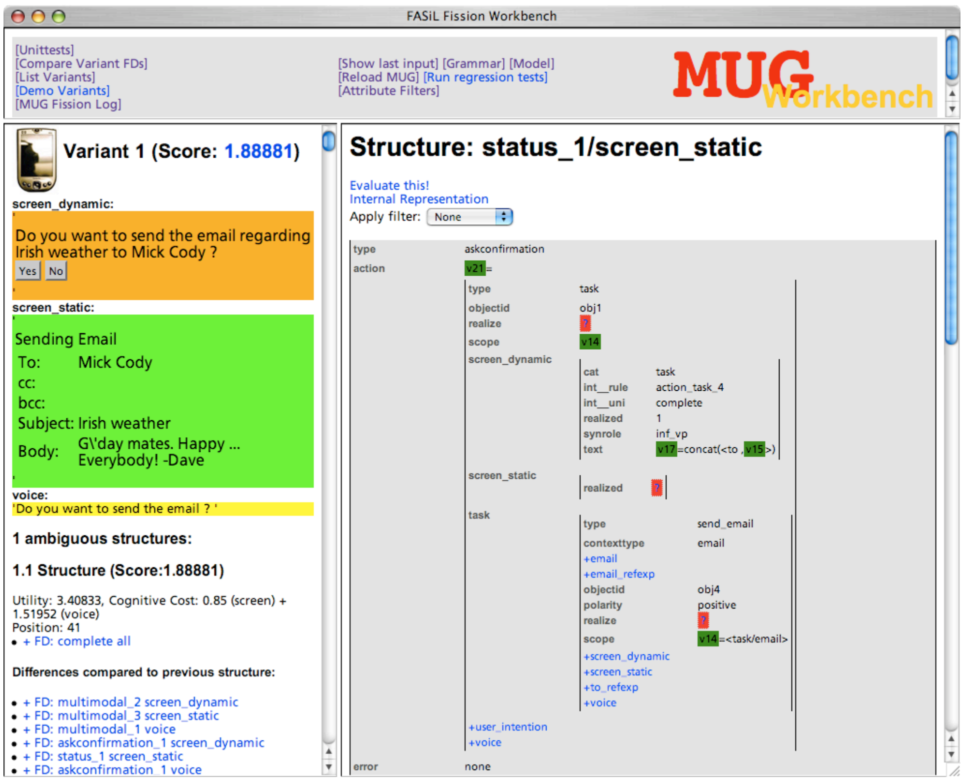
2.3 Inspecting variants of output

In the *variants view* (Fig. 1), these variants may be inspected and compared: the workbench lists, for each variant, the components used. Variants with the same generated text, but different structures, are automatically flagged, as we found this to be a common problem during the development of grammars. Generally, the variants view is a good way to deal with faulty or extraneous variants.

Misspelled variable names, but also variables in the wrong positions in AVMs are a common source of errors. Such variables remain unbound. The workbench marks them clearly in the display. The developer can also inspect variables easily and collapse or filter the rather large feature structures. Syntax errors are shown, when the grammar is loaded via the workbench user interface.

2.4 Log view

This view of the process is purely logical: there is no conceptual time-line in unification-based grammars as in procedural programs. We found that a more procedural view may help to spot problems with variants that failed to come up, furthermore it is a way to spot efficiency bottlenecks or to simply learn about how the formalism works. We offer a *log view* (Fig. 2) that enumerates all the steps that the MUG interpreter takes to apply a grammar to the input. These steps are shown for each variant of the output. This view allows inspection of the state of the sub-substructures as they were before and after a component (for a given mode) was applied.



FASiL Fission Workbench

[Unittests]
[Compare Variant FDs]
[List Variants]
[Demo Variants]
[MUG Fission Log]

[Show last input] [Grammar] [Model]
[Reload MUG] [Run regression tests]
[Attribute Filters]

MUG Workbench

Variant 1 (Score: 1.88881)

screen_dynamic:

Do you want to send the email regarding Irish weather to Mick Cody ?
Yes No

screen_static:

Sending Email
To: Mick Cody
cc:
bcc:
Subject: Irish weather
Body: G\'day mates. Happy ... Everybody! -Dave

voice:
'Do you want to send the email ? '

1 ambiguous structures:

1.1 Structure (Score:1.88881)

Utility: 3.40833, Cognitive Cost: 0.85 (screen) + 1.51952 (voice)
Position: 41
- + FD: complete all

Differences compared to previous structure:

- + FD: multimodal_2 screen_dynamic
- + FD: multimodal_3 screen_static
- + FD: multimodal_1 voice
- + FD: askconfirmation_1 screen_dynamic
- + FD: status_1 screen_static
- + FD: askconfirmation_1 voice

Structure: status_1/screen_static

Evaluate this!
Internal Representation
Apply filter: None

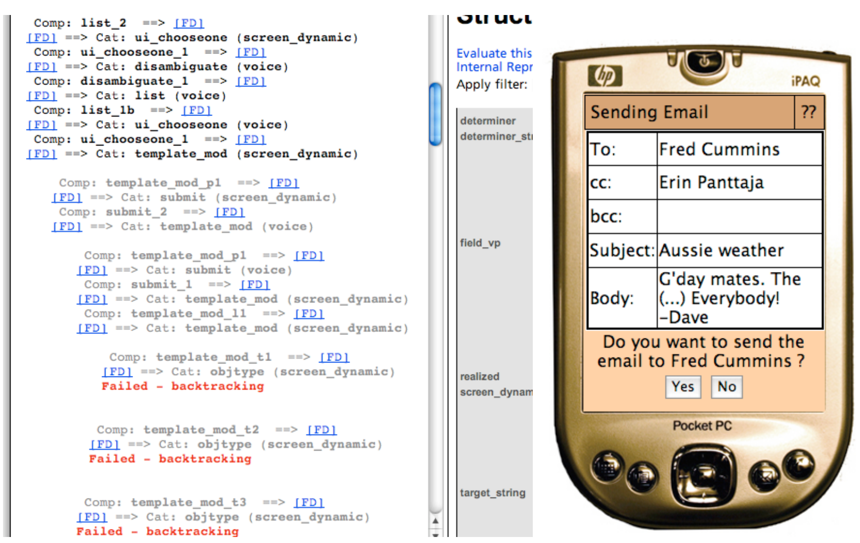
```
type            askconfirmation
action          v21=
                type            task
                objectid        obj1
                realize         ?
                scope           v14
                screen_dynamic
                                cat         task
                                int__rule   action_task_4
                                int__uni    complete
                                realized    1
                                synrole     inf_vp
                                text        v17=concat(<to ,v15>)
                screen_static
                                realized    ?
                task
                                type            send_email
                                contexttype     email
                                +email
                                +email_refexp
                                objectid        obj4
                                polarity        positive
                                realize         ?
                                scope           v14=<task/email>
                                +screen_dynamic
                                +screen_static
                                +to_refexp
                                +voice
                +user_intention
                +voice
error           none
```

Figure 1: Variants and a large AVM. Attributes can be collapsed.

A useful feature in this view is the marking of steps that were undone by means of backtracking, because – at a later stage in the generation process – an application of a rule failed. In many cases, the cause of the failure is a bug in the grammar. In other cases, it is desired behavior, but computationally inefficient. Such effects are visualized in the log view.

3 Applications

The MUG Workbench aided two experienced and one novice grammar writers to create a multimodal UI for personal information management (handling e-mail), which contains 126 components (190 with disjunction compiled), and a second, smaller MUG (39 comp.), which generates a short coherent discourse with pronouns. To test (see [3]) and also demonstrate the grammar, multimodal output can be made on any networked device (e.g. PDA) with an HTML client, with text-to-speech voice rendered on the server, as well as on simulated devices on the local workstation. A dialogue system in Java will demonstrate the use of MUG a complete environment.



```
 Comp: list_2  ==> [FD]
[FD] ==> Cat: ui_chooseone (screen_dynamic)
 Comp: ui_chooseone_1  ==> [FD]
[FD] ==> Cat: disambiguate (voice)
 Comp: disambiguate_1  ==> [FD]
[FD] ==> Cat: list (voice)
 Comp: list_1b  ==> [FD]
[FD] ==> Cat: ui_chooseone (voice)
 Comp: ui_chooseone_1  ==> [FD]
[FD] ==> Cat: template_mod (screen_dynamic)

     Comp: template_mod_p1  ==> [FD]
    [FD] ==> Cat: submit (screen_dynamic)
     Comp: submit_2  ==> [FD]
    [FD] ==> Cat: template_mod (voice)

         Comp: template_mod_p1  ==> [FD]
        [FD] ==> Cat: submit (voice)
         Comp: submit_1  ==> [FD]
        [FD] ==> Cat: template_mod (screen_dynamic)
         Comp: template_mod_l1  ==> [FD]
        [FD] ==> Cat: template_mod (screen_dynamic)

             Comp: template_mod_t1  ==> [FD]
            [FD] ==> Cat: objtype (screen_dynamic)
            Failed - backtracking


           Comp: template_mod_t2  ==> [FD]
          [FD] ==> Cat: objtype (screen_dynamic)
          Failed - backtracking


         Comp: template_mod_t3  ==> [FD]
        [FD] ==> Cat: objtype (screen_dynamic)
        Failed - backtracking
```

Sending Email ??

To:	Fred Cummins
cc:	Erin Panttaja
bcc:	
Subject:	Aussie weather
Body:	G'day mates. The (...) Everybody! –Dave

Do you want to send the email to Fred Cummins ?

Yes No

(a) (b)

Figure 2: a) In the log view, steps taken back during backtracking (because they didn't lead to valid solutions) are greyed out. b) life-size results can be demonstrated from the workbench (design: E. Panttaja)

Acknowledgements

The author would like to thank Erin Panttaja for her substantial assistance in the development of the MUG System, as well as Fred Cummins, Eva Maguire, Vitor Santos Costa and Jan Wielemaker. Part of this research was funded by the European Commission under the FASiL project, contract number: IST-2001-38685.

References

[1] Michael Elhadad and Jacques Robin. Controlling content realization with functional unification grammar. In *Proc. of the 6th International Workshop on NLG*. Springer, Lecture Notes in AI, 1992.

[2] Martin Kay. Functional grammar. In *Proceedings of the Fifth Meeting of the Berkeley Linguistics Society*, pages 142–158, Berkeley, CA, 1979.

[3] Erin Panttaja, David Reitter, and Fred Cummins. The evaluation of adaptable multimodal system outputs. In *Proceedings of the Workshop on Multilingual Linguistic Resources (MLR2004), at COLING*, to appear 2004.

[4] David Reitter, Erin Marie Panttaja, and Fred Cummins. UI on the fly: Generating a multimodal user interface. In *Proceedings of HLT-NAACL-2004*, Boston, 2004.