Design Guidelines for Domain Specific Languages

Gabor Karsai², Holger Krahn¹, Claas Pinkernell¹, Bernhard Rumpe¹, **Martin Schindler¹**, Steven Völkel¹

1. Department of Computer Science 3 (Software Engineering) RWTH Aachen University, http://www.se-rwth.de/
2. Institute for Software Integrated Systems Vanderbilt University Nashville, USA

Outline

1. Introduction
2. DSL Design Guidelines
3. Discussion
4. Conclusion
Design Guidelines for Domain Specific Languages

1. Introduction

Why Design Guidelines for DSLs?

- Designing a new DSL
  - needs experience and
  - is sometimes error-prone and time consuming

- Existing tools simplify technical aspects
  but lacks support for a good language design

- Guidelines based on
  - our experience in developing languages
  - relying on existing guidelines on general purpose and modeling languages

Target:
Support DSL developers to achieve better quality of the language design and acceptance among its users.
Categories of DSL Design Guidelines

- 5 categories along the language development process:

1. **Language Purpose**: 
   *What is the aim of the language?*

2. **Language Realization**: 
   *How to implement the language?*

3. **Language Content**: 
   *Which elements should be included?*

4. **Concrete Syntax**: 
   *How to define a readable representation of the elements?*

5. **Abstract Syntax**: 
   *How should the language represented internally?*
1. Language Purpose

- Guideline 1: *“Identify language uses early.”*
  - many forms of usage:
    - documentation, analysis, configuration, code generation, …
  - differences strongly influence needed language concepts

- Guideline 2: *“Ask questions.”*
  - Who is going to model in the DSL?
  - Who is going to review the models? When?
  - Who is using the models for which purpose?
    ➔ Identify the domain, its experts, and the development process

- Guideline 3: *“Make your language consistent.”*
  - DSLs are typically designed for a specific purpose
  - each feature of a language should contribute to this purpose

2. Language Realization

- Guideline 4: *“Decide carefully whether to use graphical or textual realization.”*
  - both approaches have advantages/disadvantages
  - weight and match against end users' preferences and uses

- Guideline 5: *“Compose existing languages where possible.”*
  - by embedding, using extendable languages, or referencing
  - concepts of the composed languages need to fit together

- Guideline 6: *“Reuse existing language definitions.”*
  - by language extension or language specialization
  - or taking existing definitions as a starter ("language pattern")

- Guideline 7: *“Reuse existing type systems.”*
  - improves comprehensibility and avoids misinterpretations
3. Language Content (1/2)

- Guideline 8: "Reflect only the necessary domain concepts."
  - by validating the language definition against the domain (e.g. using examples)
  - to ensure expressiveness for all necessary domain concepts

- Guideline 9: "Keep it simple."
  - one of the main targets
  - eases implementation, introduction, understandability, …
  - achieved by guidelines 10-12

- Guideline 10: "Avoid unnecessary generality."
  - by preventing generalization or parameterization not yet needed

3. Language Content (2/2)

- Guideline 11: "Limit the number of language elements."
  - sublanguages can cover different aspects of the system/domain
  - libraries extend expressiveness based on basic language elements

- Guideline 12: "Avoid conceptual redundancy."
  - concepts with none or slightly differences are often source of confusion

- Guideline 13: "Avoid inefficient language elements."
  - efficiency of a model should be transparent to the language user
  - should not depend on specific elements used within the model
4. Concrete Syntax (1/3)

- Guideline 14: "Adopt existing notations domain experts use."
  - inventing a new concrete syntax raises the barrier for domain experts
  - chose syntax close to existing notations
    (within the domain or other common used languages)

- Guideline 15: "Use descriptive notations."
  - supports learnability and comprehensibility

- Guideline 16: "Make elements distinguishable."
  - basic requirement to support understandability
  - usually a document is written only once but read many times
  - efficiency for the reader more important than for the writer

4. Concrete Syntax (2/3)

- Guideline 17: "Use syntactic sugar appropriately."
  - syntactic sugar improves readability
  - but an overuse can hide the important content

- Guideline 18: "Permit comments."
  - enables explanation of design decisions
  - for better understanding or even documentation

- Guideline 19: "Provide organizational structures for models."
  - possibility to arrange models in hierarchies
  - to handle complex systems
  - requires definition of references
4. Concrete Syntax (3/3)

- Guideline 20: *Balance compactness and comprehensibility.*
  - compact notations enables productivity while writing but can hinder comprehensibility
  - short notations are more preferable for frequently used elements

- Guideline 21: *Use the same style everywhere.*
  - improves understandability
  - eases identification of language elements
  - user can obtain some kind of intuition for a new language

- Guideline 22: *Identify usage conventions.*
  - not every aspect should be defined within the language definition (e.g. a certain layout)
  - conventions describe more detailed regulations that can, but need not be enforced

5. Abstract Syntax

- Guideline 23: *Align abstract and concrete syntax.*
  - eases automated processing, transformations, and presentation (pretty printing) of the model

- Guideline 24: *Prefer layout which does not affect translation from concrete to abstract syntax.*
  - otherwise using different editors or arranging the model might change its meaning without purpose

- Guideline 25: *Enable modularity.*
  - enables incremental processing of the models
  - important for comprehensibility and efficiency in handling large systems

- Guideline 26: *Introduce interfaces.*
  - to increase flexibility and hiding complexity
3. Discussion

Design Guidelines for Domain Specific Languages

- Depending on language purpose and domain, guidelines might be:
  - contradicting: e.g., combining existing languages may introduce conceptual inconsistencies
  - unimportant: e.g., none executable DSLs for documentation cannot introduce inefficient elements
  - too cost or time intensive: e.g., for small DSLs with few users some improvements might not amortize the costs/time

Guidelines have to be matched against purpose, complexity, and number of users of the resulting language.
4. Conclusion

- Discussion of 26 guidelines
  - as a basis for language design and development
  - categorized along the development phases

- Guidelines need to be
  - weighted and balanced specific to domain and purpose

- Other guidelines are needed for
  - integrating DSLs in a software development process,
  - deploying it to new users, and
  - evolving syntax and existing models in a coherent way