Language Design Tools

The following three design tools offer a safe and reliable step-wise approach into obtaining a custom configuration language that addresses your customers needs, yields a significant & measurable business value and that seamlessly fits with your product.

1. Decider

The configuration language decider reduces the risk of accidently implementing the wrong kind of configuration language.

Building a configuration language is expensive; there are many decisions to be made all resulting in very different architectures and implementation strategies. What purpose does your language serve? How frequently does your product change and along side its configuration language?

The result of the decider is an architectural reference frame and design decisions to guide you in selecting the proper implementation techniques ranging from:

1. **Interface** of the configuration language to determine the abstract representation of a configuration
2. **Mechanism** to create a new product variant that will determine the engine of the configuration language
3. **Language origin** of the configuration language to determine the syntactic and semantic abstraction level of the configuration and its relationship with the implementation language.
4. **Artifact type** to determine the output(s) of a configuration.
5. **Artifact model** to determine the semantic density of configuration.
6. **Specification time** to determine the relationship of the configuration language with the development cycle
7. **Extensibility** of the configuration language to determine the required maintainability in the face of changing requirements.

Offer:

- Analyze your existing language architecture with respect to the language space.
- Identify problems and pitfalls to propose the best fitting language architecture

2. Formulator

The configuration language formulator allows you to realize the full potential of your configuration language by finding the balance between easy but not simplistic, expressive but not complex.

For building the right configuration language you need to balance expressiveness between specificness of your application domain (easier, reliable) and freedom of generic programming languages (extensible, not constrained). Too specific, you can't express sufficient variations and there is constant maintenance. Too generic, you involve too much complexity and there are plenty of existing languages already.

The result of the formulator is a formal description of the syntax of your configuration language such that your configuration language is a (1) small, (2) concise programming languages or executable specification languages containing the (3) appropriate notations and (4) abstractions, (5) expressive power (6) tailored and (7) optimized towards the specific problem domain or application area.

To form the language, we use a set of design patterns that occur in several existing configuration languages:

- **Block Scope** shapes your configurations into clear structures.
- **Method Chaining** shapes your configurations as readable sentences.
- **Keyword Arguments** turns a method into a list of properties
- **Seamless Constructor** hides the technicalities of object creations
- **Entity Alias** allows us to use application domain concepts
- **Operator Expressions** us to use application domain operations
- **Clean Method Calls** renders a human friendly syntax
- **Custom Return Objects** supports configurations as readable sentences.

Offer:

- Help you to understand the delicate trade offs when formulating a configuration language
- Analyze your existing language design with respect to our design principles
- Shape your code configurations into a configuration language
- Assist you with common patterns distilled from a large body of already existing languages.
- Finetune an existing configuration (language) implementation

Contact:

Dr.-Ing. Sebastian Günther
sebastian.gunther@vub.ac.be

Dr. Thomas Cleenewerck
tcleenewerck@gmail.com

Vrije Universiteit Brussel l Department of Computer Science l SOFT Research Group

Pleinlaan 2 l 1050 Brussels l Belgium

Photos courtesy of Martijn Nijenhuis, Noel C. Hankamer