E Simple Group Project

The goal of the project is to develop a product configuration tool (or simply a *configurator*) that can be run on different platforms.

E.1 Configurators

Products such as automobiles and computers can be customized to fulfill specific user needs. The user will assign values to a set of parameters. Then a product is built based on these values and delivered to the user. For an automobile a parameter may be the fuel type (Diesel/Gasoline) or the choice to include warmed seats (On/Off). For a computer a parameter may be the operating system (Vista/Win7/Win8/Linux/OS X) or the CPU type (AMD/Intel). These examples describe how a system can be built according to a predefined configuration of parameters.

In other situations, the parameters may be adjusted at runtime. HVAC systems control the heating, ventilation and air conditioning of indoor spaces (e.g. office buildings, universities, factories). Such systems may be configured at run-time by updating their functioning parameters. Such parameters may be the temperature settings for rooms or floors.

The parameters can be of various types depending on which system properties they describe. There can be Boolean parameters (e.g. On/Off, True/False), integer parameters which take their value from a range (e.g. 1..10, -3..+3), there can be parameters with a specific set of values (e.g. the material for wheel rims [Steel, Aluminum alloy, Chrome]).

While some parameters are mandatory (e.g. a car needs an engine) others may be optional (e.g. electric windows).

However, not all features are compatible with one-another. Choosing an OS X (Mac) operating system allows only for Intel CPUs. This means that some parameters are depending on others. In some cases selecting a value for a parameter might limit the range of values for other parameters; in other cases it might hide other parameters altogether.

In the most general case, the configurator tool will abide by a set of dependency constraints that limit the choices of the user. For example, for an automobile configurator that has among others the three parameters: *Rims*, *RimsColor* and *MotorPower*, the might be a constraint like the following:

the color of the rims can be matte black
if and only if the rims are made of aluminum
and the engine power is 120HP or 150HP.

In order to specify such constraints, the developers need an expression language which includes parameters, parameter values, relational and logical operators.

For further inspiration, we suggest investigating the available online configurator sites.

E.2 Requirements

The project objective is to design and implement a domain-specific language for configurators. The DSL is meant to be subject independent (so the domain here is "configurators" and not "automotive" or "ERP systems"). It should in principle be able to specify configurators for any domain (e.g. automotive, medical, operating systems, HVAC). Once a configurator instance is specified, it must be executed on a suitable platform (e.g. web application, Android application, Java application).

The process has the following steps:

1. Define a metamodel for the DSL. The metamodel specifies the components and structure of your models. It must allow working with several types of parameters and writing configuration constraints using an expression language. *The result is an EMF Ecore diagram of your metamodel.*
2. Specify the static semantics of the DSL. Static semantics defines restrictions on the structure of valid models that are hard or impossible to express with the metamodel. *The result is a set of constraints over the metamodel.*
3. Define a textual grammar for the DSL. The grammar must be concise and user friendly to allow even non-programmers to define configurator models with ease. *The result is a grammar.*
4. Make your model executable on two platforms. You can implement a *code generator* that turns your model into HTML+JavaScript or Java code which is then built into an application. You can also save your model in a common data format (e.g. XML, JSON) and implement an *interpreter* that takes the model as input and builds the user interface on the fly. The choice of platform is left to each group. A web application may be suitable for automobile configuration, while an Android app may be more suitable for runtime configuration of HVAC systems. *The result is* TWO *applications (each on a different platform) that can run configurator models written in your DSL.*
5. The applications must display the parameters in a user interface and allow the users to select values; must enforce the configuration constraint; must save the chosen configuration data. *The result is the configuration data.*
6. You must also test your implementations. *The result is a set of test plans or test cases along with a compelling argument for correctness.*

Notes:

1. Each group should target at least *two platforms.*
2. When choosing the platforms take into consideration future implications such as: will it take a long time to learn using the target language;
3. Consider how you will allow storing and accessing the configuration results (this is secondary for us, what the system does with a configuration, but a clear interface should be provided: save it in a file, send to a database, email it, allow querying it via an API, etc).
4. Before starting to work on a code generator, hand-code a configurator instance on your target platform. Once it is finished you will have a clear view of what the code generator must output and which are the variable parts. Then turn the hand-written code into a

template.

E.3 Hand-in

Hand-ins (in a single PDF file, in this order):

1. One page on key design decisions taken in your project. Do not exceed this. At least 11pt font size, 2cm margins.
2. The metamodel (submit several diagrams that emphasize different important aspects of the metamodel)
3. The concrete syntax (grammar) of the DSL and a couple of example models written using your syntax;
4. The static semantics (the constraints of your meta-model);
5. (If building a code generator) a print out of an interesting template;
6. (If building an interpreter) print out the key part of the interpreter; IF additional format than ecore or your textual syntax is used, please also provide an example in that format.
7. Test plans/test cases and a compelling argument for correctness. The argument should itself not exceed half a page, same conditions as in point 1.
8. No other file formats are accepted (no zip archives, no word documents)