Lecture on Generative Programming

for Software Design & Architectures

prepared by Michal Antkiewicz

1 Overview

The topics on Generative Programming brought up in this lecture:

- What is Generative Programming?

The definition of GP and elaboration on keywords: software system families, requirements specification, automatically manufactured, implementation components, configuration knowledge

- GP Process

The typical steps necessary to use GP in practice.

- Technology Projections

Technologies used to express concepts in Problem Space, to implement generators in Configuration Knowledge and to implement solution in Solution Space.

- Object Technology

Described in context of GP.

- Component Technology

Described in context of GP.

- Feature Modeling

The description of Feature Diagrams notation.

- Exercise

Feature Modeling of family of counters.

- Specialization

The process of removing variation points in context of Feature Model.

- MDA

MDA paradigm in GP. Can also be seen as a specific technology projection.

2 What is Generative Programming?

> " Generative Programming (GP) is a software engineering paradigm based on modeling of ***software system families ...***"

Instead of building one particular system from scratch, GP focuses attention on family of similar systems.

> "... such that, given a particular ***requirements specification,*** ..."

The user is particularly interested in system features (functional and non-functional). We would like a customer to order software systems is the same way as ordering items from catalog (for example: cars – manual or automatic transmission, gasoline or electric engine etc.).

> "... a highly customized and optimized intermediate or end-product can be ***automatically manufactured*** on demand, ..."

We would like to construct ordered system as automatically as possible. We can obtain intermediate product, which still requires some manual development or ready to use end-product.

> "... from elementary, reusable ***implementation components*** ..."

We would like to have implementation components in our generative library that would provide as much interoperability as possible and be as little redundant as possible.

> "... by means of ***configuration knowledge***. "

We would like to capture the knowledge which is usually lost during typical, single system,

development process. When manually composing components, detailed design decisions are usually not documented and therefore lost.

Very important is to mention that GP can be applied at every level. Can be used to construct the whole system or just small generative container library. The good example is GMCL (Generative Matrix Computation Library). The domain of matrix computations is well established over 30 years of development. The GMCL library can construct over eight thousand different matrix types and provide efficient algorithms for using them.

3 The Generative Programming Process

Unlike single system development process, generative programming process consists of two, running in parallel processes: development for reuse and development with reuse.
Development for reuse is a process of constructing a generative domain model.
Development with reuse is a process of constructing on demand the particular member of the software system family.

Applying GP is not profitable in every case. If a software company realizes one custom order which is not an usual project, there is no need to use GP, because overall costs would be to high. But in case, when a company specializes in one particular type of software (for example: WEB portals, portfolio management, mobile phones software), investment in creation of generative domain will be profitable.

In practice, when customers add new requirements, ordered system is extended manually. But, then these extensions are incorporated into generative domain to allow automatic build in the future.

4 Development For Reuse

Development for reuse is a process of creation of a generative domain, which includes:

- system family scoping – determining family boundaries, deciding which features should be included, which not, analyzing project stakeholders, potential markets, technology forecasts, etc.
- feature modeling – determining common features and variabilities (variation points) – what do family members have in common, where do they differ. Feature modeling provides the basis for deriving the categories of implementation components, the means for specifying family members and the configuration knowledge.
- design and implementation – designing common architecture, means of specifying family members, capturing configuration knowledge in a generator, implementation of components and generative library.

5 Technology Projections

In the problem space, we have domain specific concepts and features. We can use a variety of techniques to describe specifications (input to generator): textual and graphical languages, programming languages, interactive wizards and GUI's.

Our configuration knowledge (how to assembly a system that satisfies specification) can be captured using templates and frames, transformation systems, metaprogramming languages and extensible programming languages.

Our components can be implemented using many technologies such as generic components, component models and aspect oriented approaches.

This is a recursive process. One's solution space may be someone's else problem space.

6 Object Technology

Object technology was to solve all problems with software development. Unfortunately, industrial experience pointed out that although classes are reusable, they are to small to rapidly speed up development process. Another solution was a framework, which is highly customizable, speeds up the development process, but unfortunately frameworks from different vendors do not fit well together and lots of effort has to be put into framework integration, so that sometimes it's better to write some code manually.

In GP object technology can also be used. We can reuse software by creating highly customizable generative libraries, and capture design knowledge (for example patterns) in generators.

7 Component Technology

Component Technology, although very useful, has many disadvantages. First of all, small components have to be manually composed. Large components do not always fit well together.

The problem of “fat components” - even if we want to use only a part of component's functionality, we have to include the whole component to our system.

As needed components are assembled on demand, millions of concrete components with all possible combinations of properties do not need to be stored. Only parts from which they are constructed have to be stored. Adding only one elementary implementation component, may potentially double the number of concrete components you can build.

By specifying the required properties of the needed component rather than choosing a concrete component from library, you rise the abstraction level of your code! If better algorithms and data structures satisfying the required properties become available, a new version of a library will be able to generate a better component to suit your needs. And you do not even have to change the client code.
If you have chosen a concrete component, a library cannot automatically replace it with better one, because it simply cannot know why you have selected this component.

8 Feature Modeling

The goal of the feature modeling is to discover commonalities and variation points. Features are describing the system, but they should not describe concrete mechanisms. For example, for a database driven system, a good feature is “transactional”, but the bad feature is “MySQL”. It's the generator's goal to choose appropriate database engine, depending on abstract feature.

Mandatory, optional features,
Exclusive-Or, Inclusive-Or groups,
Open Feature.

9 Exercise

10 Specialization

Specialization is a process of choosing a concrete member from system family. In other words, it is a process of removing variation points from the model. Removing all of them is called full specialization.

During specialization, we examine every feature node in every level, starting from concept node.
In the example, the first variation point we encounter is optional feature *f3*. When chosen, it becomes mandatory feature, when not, the whole branch of the tree is removed. On the next level there are two variation points left. Under feature *f2* we can choose *f4* or *f5* or both *f4, f5*. Under feature *f3* we can choose either *f6* or *f7*. We say, that a member is fully specialized, when there are no more variation points.

11 MDA

We can view writing a program also as modeling.
PIM and PSM are relative concepts. “Someone's PIM can be someone's else PSM”

The modeling is done using some domain specific modeling languages.

12 References

[Cza02] Czarnecki K, Eisenecker U.: “Generative Programming, Methods, Tools and Applications”, Addison-Wesley, 2002.

[Omg] http://www.omg.org/mda/