UECS2344 Software Design: Lecture 12

Further Topics

Software Product Lines (From Software Engineering – Sommerville)

When a company has to support a number of similar but not identical software products or systems, one of the most effective ways is to create a software product line. This is another form of reuse. For example, a printer manufacturer has to develop printer control software, where there is a specific version of the product for each type of printer. These software product versions have many common elements, so it makes sense to create a core product (the product line) and adapt this for each printer type.

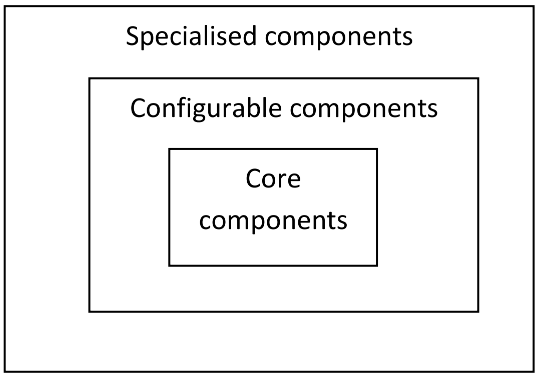
A software product line is a set of applications with a common architecture and common components, with each application specialised to reflect specific customer requirements. The core system is designed so that it can be configured and adapted to suit the needs of different customers or requirements. This may involve the configuration of some components, implementing additional components, and modifying some of the components to reflect new requirements.

Benefits:

- high proportion of the application code is reused in each system
- testing is simplified because tests for large parts of the application may also be reused
- overall development time is reduced
- software engineers learn about the application domain through the software product line and so become specialists who can work quickly to develop new applications

Designing a software product line involves identifying common functionality in product instances and developing a base application which contains:

- core components that provide infrastructure support. These are not usually modified when developing a new product instance of the software product line.
- configurable components that may be modified and configured to specialise them to an new product instance. Sometimes it is possible to reconfigure these components without changing their code by using a built-in component configuration language.
- specialised components which may be replaced when a new product instance of the product line is created.



Model-Driven Engineering (From Software Engineering – Sommerville)

Model-driven engineering (MDE) is an approach to software development where models rather than programs are the main outputs of the development process. The programs that execute on a hardware/software platform are then generated automatically from the models. This raises the level of abstraction in software engineering so that software engineers no longer have to be concerned with programming language details or the specifics of execution platforms.

Model-driven engineering was developed from the idea of model-driven architecture (MDA) proposed by the Object Management Group (OMG) as a new software development paradigm. MDA focuses on the design and implementation stages of software development whereas MDE is concerned with all aspects of the software engineering process.

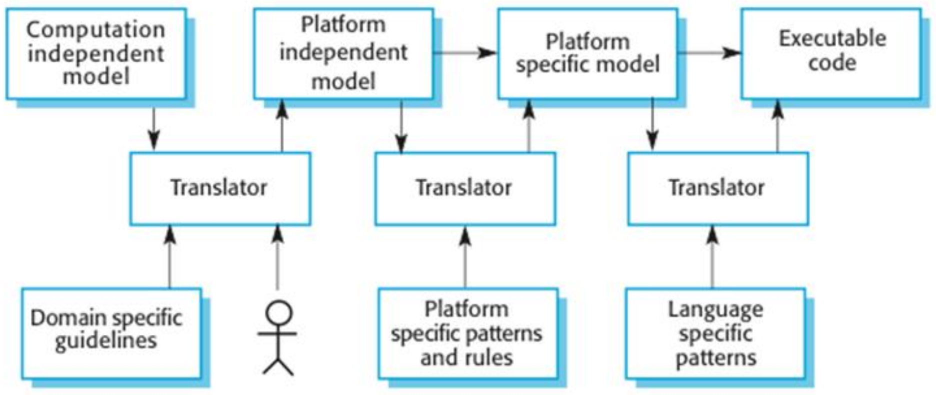
Model-Driven Architecture

Model-driven architecture is a model-focused approach to software design and implementation that uses a subset of UML models to describe a system. Here, models at different levels of abstraction are created. From a high-level, platform independent model, it is possible to automatically generate a working program.

The MDA method recommends that 3 types of abstract system model should be produced:

1. A computation independent model (CIM) that models the important domain abstractions used in the system. CIMs are sometimes called domain models.
2. A platform independent model (PIM) that models the operations of the system without reference to its implementation.
3. Platform specific models (PSM) which are transformations of the platform independent model with a separate PSM for each application platform.

Transformations between these models may be defined and applied automatically by software tools. Tool support for translation from PIMs to PSMs already exists for common platforms although automatic translation from CIM to PIM is still at the research stage.


Computation independent model
Platform independent model
Platform specific model
Executable code
Translator
Translator
Translator
Domain specific guidelines
Platform specific patterns and rules
Language specific patterns

The advantage of MDA is that it allows software engineers to think about systems at a high level of abstraction without concern for the details of implementation. This reduces the likelihood of errors, speeds up the design and implementation process, and allows for the creation of reusable, platform-independent application models. By using powerful tools, system implementations can be generated for different platforms from the same model. Therefore, to adapt the system to some new platform technology, it is only necessary to write a model translator for that platform. When this is available, all platform-independent models can be rapidly re-hosted on the new platform.