The Tyranny of the Dominant Model Decomposition

Maja D'Hondt and Theo D'Hondt
{mjdhondt|tjdhondt}@vub.ac.be
Vrije Universiteit Brussel, Belgium

September 19, 2002

1 Position Statement

The *Model-Driven Architecture* (MDA) approach advocates model refinement for transforming a platform-independent model of an application to a platform-specific model. As this workshop also advertises, generative techniques are a likely approach for transforming an abstract model to a more concrete realisation of that model. We expect, however, that the model refinement proposed by MDA and offered by generative techniques will result in each component being refined separately which will ultimately not suffice for obtaining the full integration of all the components involved. As *Aspect-Oriented Software Development* (AOSD) points out, a decomposition into components can not always be maintained throughout software development because there are components that do not fit the dominant decomposition of the representation medium used at a certain phase, and hence crosscut it.

Let us explain this position statement in the following sections. First we briefly explain some ideas of MDA and generative techniques relevant to our statement. The next section concerns AOSD and why it is indispensable for MDA. Finally we illustrate our point using business rules, a topic of current interest, which is an example of a not-so-traditional crosscutting aspect but nevertheless undeniably relevant for MDA.

2 Model-Driven Architecture and Generative Programming

When constructing an MDA-based application, a platform-independent model should be created first. But before transforming this model to a model for a specific platform such as CORBA or J2EE, one should determine what MDA refers to as the *pervasive services* of the application. Pervasive services exist in all applications, independent of their context (e.g. internet, enterprise, embedded, ...), and typically include directory services, event handling, persistence, transactions, and security. Since different platforms provide for the same pervasive services in a different way, MDA proposes a common model at the same level as the platform-independent model that does not restrict the

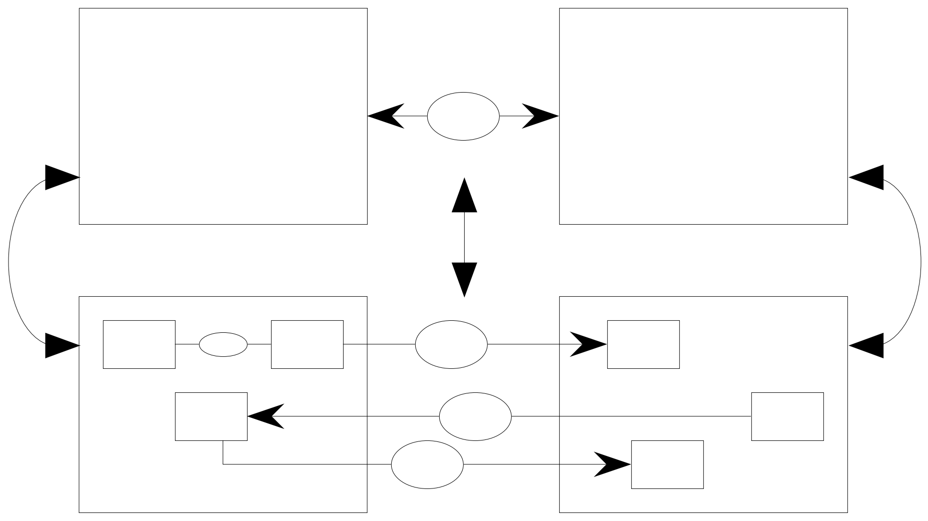

Figure 1: Refining models and model interactions (from [4]).

services to characteristics of a particular platform. Only when the features and architecture of a pervasive service are resolved, will platform-specific definitions be generated. This clearly shows that MDA envisions separate generators for each of the components at the platform-independent level, i.e. the core application and the pervasive services.

Moreover, [4] states that model refinement does not preserve the encapsulation of components in the models, which is illustrated in Fig. 1 (taken from the same document). This figure shows that a model typically contains components and relations between them. If such a component is refined, a complex network of objects appears. Likewise, a relation can be refined to reveal detailed interaction protocols between two components. As is shown in Fig. 1, the encapsulation of the abstract representation of the components is broken in the more concrete model, because the detailed interaction relates objects from the one component directly to objects from the other component.

3 Aspect-Oriented Software Development

On the AOSD web site [1] we read: "*Aspect-oriented software development is a new technology for separation of concerns (SOC) in software development. The techniques of AOSD make it possible to modularize crosscutting aspects of a system. Like objects, aspects may arise at any stage of the software lifecycle, including requirements specification, design, implementation, etc. Common examples of crosscutting aspects are design or architectural constraints, systemic properties or behaviors (e.g., logging and error recovery), and features.*".

The pervasive services defined at the platform-independent level in the MDA approach will ultimately crosscut the core application model at the implementation level if no aspect-oriented im-

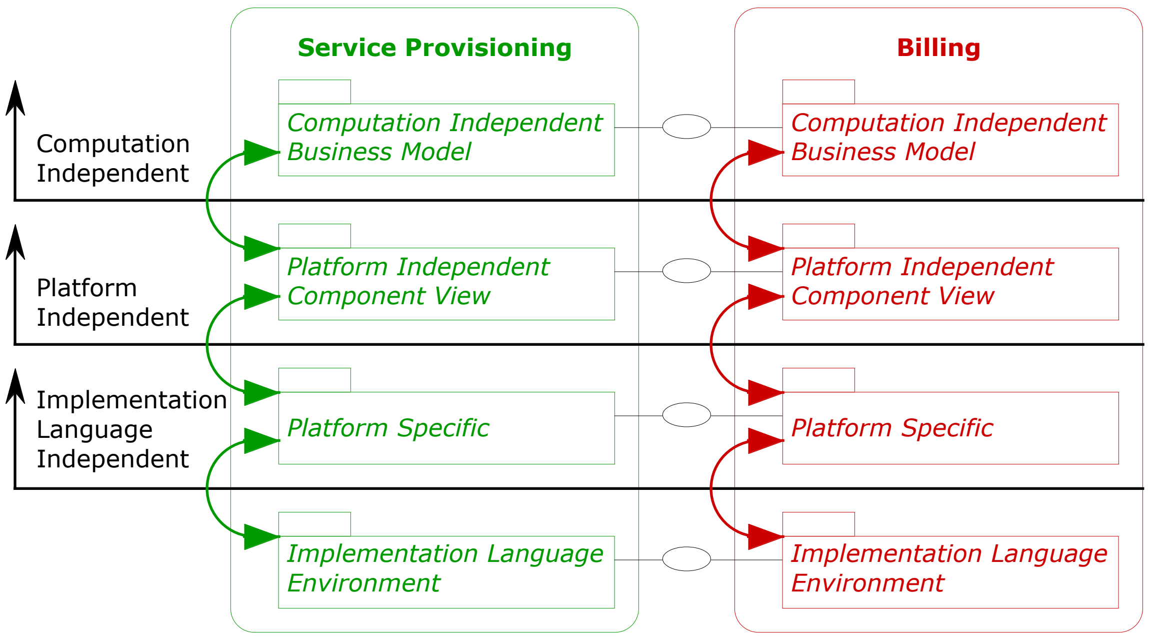
Service Provisioning
Billing
Computation Independent
Platform Independent
Implementation Language Independent
Computation Independent Business Model
Platform Independent Component View
Platform Specific
Implementation Language Environment
Computation Independent Business Model
Platform Independent Component View
Platform Specific
Implementation Language Environment

Figure 2: Consistent model separations and relationships in MDA (from [4]).

plementation approach is used. While abstracting away the crosscutting relation may work up to a certain level of modeling, the actual weaving of the components will have to be performed sooner or later. Similarly, crosscutting aspects may arise within the models. As explained earlier, model refinement breaks the encapsulation of components because detailed component interaction reveals objects from different components relating each other directly. Additionally, the encapsulation of the objects themselves can also be broken by these crosscutting aspects.

Even if models and components provide hooks for dealing with the crosscutting aspects, one still has to consider that these days applications evolve rapidly and that some development caters to entire software product lines. These forces result in a multitude of unanticipated variabilities in the software (product line), which AOSD techniques can take care of as well.

4 Business Rules

Although our interests and work are focused on decoupling business rules from the core application in order to facilitate evolving and updating them, we were prompted to mention them in this position paper because of an example in the MDA document [4]. Figure 2 is again based on a figure in this same document, and shows that MDA deems it necessary to separate models for Service Provisioning and Billing consistently. Billing policies are business decisions and are typically ideally represented as business rules. In our main research track, our hypothesis is that business rules are crosscutting aspects, as is also argued in a master's thesis from our lab, which in addition to this shows how AspectJ can be used to deal with decoupling them from the core application [2].

We particularly like using business rules as a case for our position statement here because – although they are crosscutting aspects – they are independent of the implementation language, the platform, and even the computation. This means that business rules already manifest themselves in the most abstract level of models in MDA. This is not the case of the more traditional aspects such as synchronisation and event handling: they are more solution-oriented and one could argue that weaving them with the other models could be postponed to a more concrete level. The business rules aspect, however, already exhibits crosscutting behaviour in the computation-independent business model and one is forced to express crosscutting relationships at this level. Moreover, business rules require a sufficiently expressive medium and a declarative style is more than indicated. Experience shows that this approach spans the full depth of the software development process [3].

References

[1] Aspect-oriented software development. `http://aosd.net/`.

[2] María Agustina Cibrán. Using aspect-oriented programming for connecting and configuring decoupled business rules in object-oriented applications. Technical report, Vrije Universiteit Brussel, Belgium, 2002.

[3] Theo D'Hondt, Kris De Volder, Kim Mens, and Roel Wuyts. Co-evolution of object-oriented software design and implementation. In *Software Architecture and Component Technology Symposium*. Kluwer Academic Publishers, 2000.

[4] The Object Management Group. *Model-Driven Architecture*, 2001. `http://www.omg.org/mda`.