About templates, models and objects

Blog by Peter Lammersma

January, 25 2019

As an experienced software solutions creator (sounds prettier than ‘software developer’, doesn’t it?), I spend most of my time in Uniface’s Integrated Development Environment (IDE). The latest version 10 mentions *template*, *model* and *object* on nearly every editor. I think it’s important to know what they mean and to understand the difference between them. Spoiler alert: it has something to do with the level of abstraction.

Development objects, specialization and generalization

Let’s start with objects. I have been using Uniface to develop and deploy applications for many years and I know the ins and outs of Uniface, but I have never seen something that is called an object. In fact, an object is a generalized term for things you and I work with every day.

Anytime you read the word ‘object’, you can replace it with anything you know in Uniface, like field, entity, menu, property, component, etc. These specializations are implementations of objects and are categorized as development objects. Development objects are created and maintained in the IDE.

Some development objects have parents. For example, a field can never exist without an entity, and it’s impossible to define a menu without a library.

Development objects can also be nested. For instance, a component contains at least one entity, and an entity contains fields. To look at it the other way around, the fields have the entity as their parent and the entity has the component as its parent.

Development objects that don’t have a parent, like components and application shells, are called *main development objects*. Main development objects have a dedicated editor in the IDE.

Abstraction

In the Uniface world, it’s all about the level of abstraction. Templates and modeled objects help you create your development objects. Templates are used to create both objects and modeled objects. The biggest difference between a template and a modeled object is inheritance. This is shown in Figure 1.

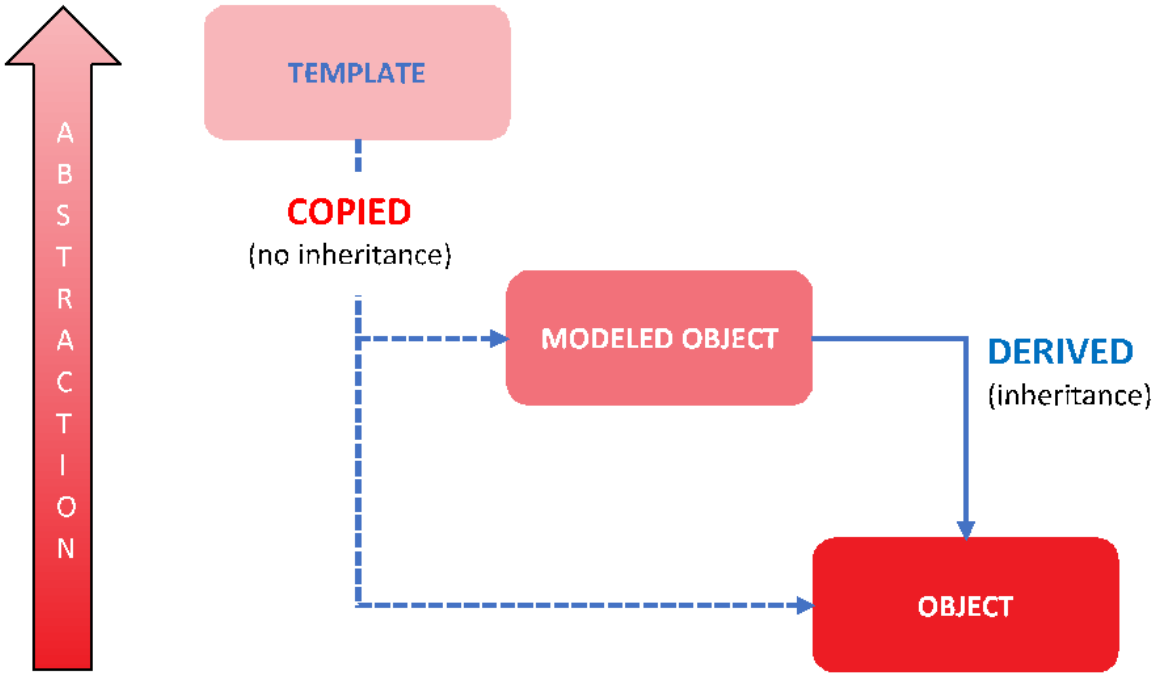

Figure 1 Relationship between template, modeled object, and object

The template is the most abstract level, while an object is concrete.

Anything created from a template has no inheritance from that template. A change in the template does not affect the (modeled) object at all. The (modeled) object is a separate copy from the template.

Uniface delivers a set of templates you can work with, but you can also create your own. In the new Uniface 10 development environment, the first thing a developer does is create a repository that will be based on templates. The first time Uniface is started, the developer must specify and load a set of templates. This can be the default Uniface set or a tailormade set of templates. But once loaded in the repository, they are static: they are not development objects.

Modeled objects are a kind of template, but they are created and maintained in the current development repository. This means they add dynamics to the templates.

A modification made in a modeled object is inherited by an object based on that modeled object. This is the inheritance that we, Uniface developers, have known and used for many years.

Template or model

For us Uniface developers, it's common practice to define a modeled representation of the structured data with entities and fields. As soon as these modeled entities and fields are 'painted' on a component, they become derived entities and fields. We know and trust that all properties and scripts are inherited from the model. That's what makes Uniface so efficient. And that is why you and I use Uniface.

Let's picture the template, model and object as layers and forget about the specialization of the objects for a minute.

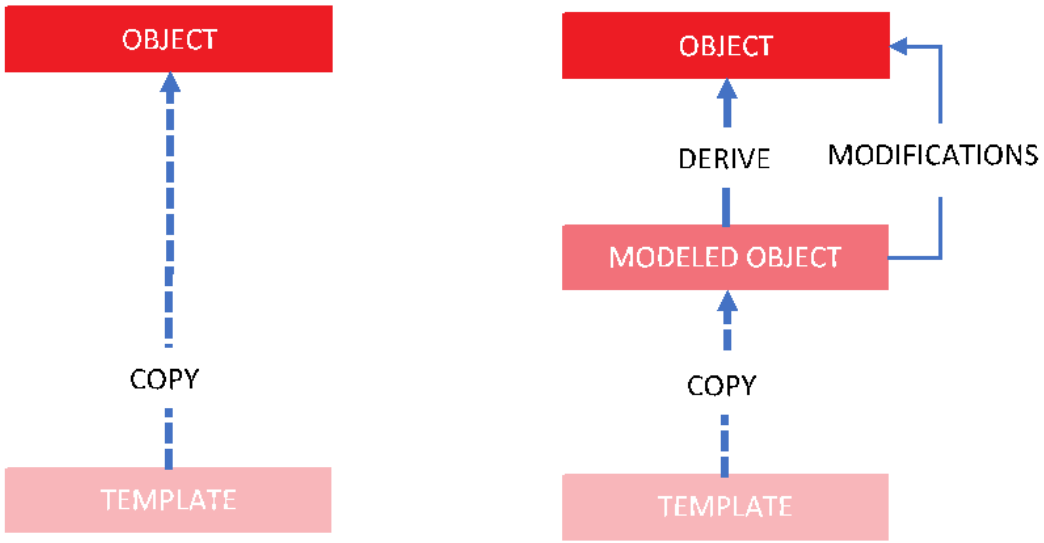

Figure 2 Template, model and object as layers

As shown in the figure, every object is a copy of a template. The model layer adds inheritance, but this is optional.

Every part of your development object is in fact based on templates. Every component, entity, field, etc. is a copy of a template. If you want to save development time or enhance maintainability, you can add inheritance to properties or coding by adding a modeled object layer for these objects.

From modeled to derived objects

Here's a small example to illustrate what I've just been discussing. An application has a few components and a couple of database tables with fields. From a business perspective, the data is the most important element; the application is there to serve the data. To ensure the consistency of the data, it's normal for us, as Uniface developers, to define modeled entities. These modeled entities have fields, keys and relationships; all of these are modeled too. This model allows the developer to use the data on components without the need to know anything about the implementation of the database.

This is a major strength of Uniface. Again, as soon as a *modeled entity* is used on a component it is a *derived entity* and used fields are derived fields.

In case of the modeled entities and fields, every Uniface developer uses templates and models daily. Uniface 10 has enhanced the use of models through the whole environment. It's up to the developer to use this concept. Templates and modeled objects are present at nearly every stage of the development process, and they make the lives of developers easier and more efficient.

Using modeled objects as much as possible is a logical move. When you want to take a model-driven approach to Uniface application development, using models is a must!