II

The Building Blocks of a Model-Driven Design

To keep a software implementation crisp and in lockstep with a model, in spite of messy realities, you must apply the best practices of modeling and design. This book is not an introduction to object-oriented design, nor does it propose radical design fundamentals. Domain-driven design shifts the emphasis of certain conventional ideas.

Certain kinds of decisions keep the model and implementation aligned with each other, each reinforcing the other's effectiveness. This alignment requires attention to the details of individual elements. Careful crafting at this small scale gives developers a steady platform from which to apply the modeling approaches of Parts III and IV.

The design style in this book largely follows the principle of "responsibility-driven design," put forward in Wirfs-Brock et al. 1990 and updated in Wirfs-Brock 2003. It also draws heavily (especially in Part III) on the ideas of "design by contract" described in Meyer 1988. It is consistent with the general background of other widely held best practices of object-oriented design, which are described in such books as Larman 1998.

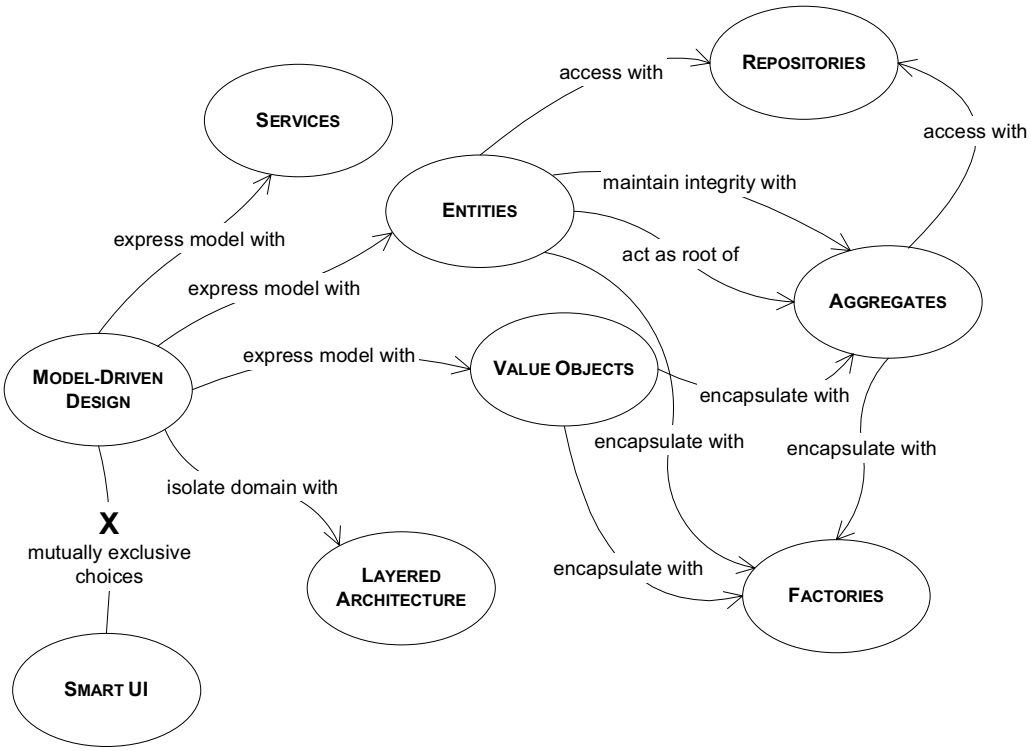
As a project hits bumps, large or small, developers may find themselves in situations that make those principles seem inapplicable. To make the domain-driven design process resilient, developers need to understand *how* the well-known fundamentals support MODEL-DRIVEN DESIGN, so they can compromise without derailing.

The material in the following three chapters is organized as a "pattern language" (see Appendix A), which will show how subtle model distinctions and design decisions affect the domain-driven design process.

The diagram on the top of the next page is a *navigation map*. It shows the patterns that will be presented in this section and a few of the ways they relate to each other.

Sharing these standard patterns brings order to the design and makes it easier for team members to understand each other's work. Using standard patterns also adds to the UBIQUITOUS LANGUAGE, which all team members can use to discuss model and design decisions.

Developing a good domain model is an art. But the practical design and implementation of a model's individual elements can be relatively systematic. Isolating the domain design from the mass of other concerns



A navigation map of the language of MODEL-DRIVEN DESIGN

in the software system will greatly clarify the design's connection to the model. Defining model elements according to certain distinctions sharpens their meanings. Following proven patterns for individual elements helps produce a model that is practical to implement.

Elaborate models can cut through complexity only if care is taken with the fundamentals, resulting in detailed elements that the team can confidently combine.