

Application Modeling

2026-09-14
SimpleModeling

Part 8 introduced Domain Modeling: organizing facts, concepts, and rules about the subject world as a Domain Model. Part 9 uses that semantic foundation to turn stakeholders' needs and goals into system behavior through Application Modeling.

An Application Model becomes a thin orchestrator that connects domain functionality according to the user's purpose. Domain Models manage important data operations and business state; the Application Model defines how processing is combined and ordered.

1 Positioning Domain Models and Application Models

A SimpleModeling Model comprises an Object Model, a Knowledge Model, and a Literate Model. Domain Models and Application Models are applied Models: they select the necessary Model elements from these constituents and organize them for a purpose.

Aspect	Domain Model	Application Model
Concern	What holds in the subject world	What users want to achieve
Starting point	Facts and observations	Needs, goals, and Use Cases
Main role	Define meaning, rules, and state	Combine domain functionality

Domain Modeling inductively organizes concepts and rules from observations of facts. Application Modeling uses that vocabulary, rules, and constraints to define behavior that fulfills needs and goals.

2 The application becomes a thin orchestrator

The Application Model receives input, invokes the required domain Services and Operations, and connects results and Events to subsequent processing or responses to users.

Business decisions, important data operations, invariant checks, and the management of business state and its transitions belong to the domain. StateMachines are also defined in the Domain Model; applications use them through Services and Events.

This division lets applications concentrate on coordination for the user's purpose while domain meaning and rules remain a shared foundation.

2.1 Why define the Models separately?

A Domain Model accumulates understanding of the subject world and has a relatively long lifecycle. It evolves with discoveries of facts, deeper understanding, and changes in institutions or business practices.

An Application Model changes with stakeholder needs and usage environments. Defining them separately lets us share understanding of the subject world while continually making behavior concrete for changing needs.

3 One application orchestrates multiple domains

One important role of the Application Model is to orchestrate multiple domains. To fulfill the user's purpose, it combines domain functionality and coordinates invocation order, the transfer of results and Events, and failure handling.

For example, when orders, inventory, and payments are managed in separate domains, the application assembles the overall procedure. Each domain manages its own meaning, rules, and state; the application connects their functionality according to the purpose.

4 Applications for different needs use a shared domain

Different stakeholders may have different needs concerning the same domain. Application Models for those needs use a common Domain Model.

For example, a customer-facing ordering application and an order-management application for business staff have different goals and Use Cases. Both can share the meaning and rules of the order domain while defining the behavior each requires.

Shared domain knowledge includes Entities, Values, relationships, rules, Events, and business state. Components and execution infrastructure may also be shared when needed during implementation.

Each application accommodates changes in its own needs. When updating a shared domain, review the impact on every application that uses it.

5 From Use Case Scenarios to Realization Models

A Use Case describes an actor's goal and the Scenarios that achieve it. Application Modeling maps the actions in a Scenario to UI boundaries, internal participants, processing, and results.

We construct a Use Case Realization Model from a Literate Model's Use Case Scenario and review their correspondence. Describing the approved executable Model in CML makes the path from requirements to implementation traceable.

5.1 Making "Confirm order" concrete

Consider a Scenario in which a user reviews an order, requests confirmation, and receives the result.

- The UI presents the order and passes the confirmation request to the application.
- The application receives the request and invokes the order domain's confirmation Operation.
- The order domain checks business rules and invariants, updates order data and business state, and emits an order-confirmed Event.
- The application receives the result, connects it to any required subsequent processing, and responds to the UI.

Retain the mappings between Scenario actions and these operations and results. These mappings also provide evidence when reviewing normal flows and failure handling.

5.2 Reviewing correspondence through Collaborations and Interactions

A Collaboration expresses how the UI, application, domain Services, and other participants divide roles and responsibilities to achieve a purpose. An Interaction expresses a concrete sequence of invocations, results, and Events as an execution example.

Collaborations organize the participant structure; Interactions let us review execution examples of a Scenario. When several domains are involved, they clarify the correspondence between application coordination and the decisions and operations belonging to each domain.

6 Adjusting Operation characteristics through properties

Application Models and Domain Models also require different Operation characteristics. Set these as properties of Components, Services, and Operations, then adjust them through defaults and individual declarations. Explicit characteristics help us select suitable execution forms and optimize realization.

A Component can have application use, domain use, or both. In a Component with a single use, that use becomes the default for its Services. Individual Services may specify their own use, and Operations inherit their Service's characteristics.

Services for application use default to stateless; Services for domain use default to stateful. Service and Operation settings can be changed for individual requirements.

A stateless Operation obtains the state needed for correct execution from arguments or external persistence and does not depend on memory-resident state retained across calls. An Operation that updates persisted domain data can have this characteristic. The stateful distinction concerns dependence on memory-resident state across calls.

6.1 An example: realization as cloud functions

A stateless Operation is a candidate for realization as a cloud function. Operations inheriting the default characteristics of Components and Services for application use are such candidates.

During implementation, check execution-platform conditions such as processing time and resources, and realize the Operation as a cloud function or as an Operation on a conventional server. When usage and pricing conditions fit, cloud functions can substantially reduce costs. This is one example of using property-defined characteristics to optimize realization.

7 Generative AI manages Models; people approve them

Generative AI proposes mappings from Use Case Scenarios to Services, Operations, and Events and generates and manages Models. People review and approve whether goals are achieved, important business decisions and data operations belong to the appropriate domains, and boundaries and failure handling are clear.

textus-cbd-support, which supports Component-Based Development, visualizes Scenario-to-Model mappings, changes, evidence, and validation information. Conformance to execution examples described in Interactions also helps assess Model behavior.

We describe the approved executable Model in CML and connect it to automatic program generation by Cozy and the Textus runtime. Feedback from users informs subsequent needs and Model updates, allowing applications to evolve continually on a shared domain foundation.