Domain Modeling – CS 314 Spring 2016

James M. Bieman
Sudipto Ghosh
Geri Georg

Definitions

- Wikipedia:

  System of abstractions that describes selected aspects of a sphere of knowledge, influence, or activity (a domain). The model can then be used to solve problems related to that domain.

  Representation of meaningful real-world concepts pertinent to the domain that need to be modeled in software. Concepts include the data involved in the business and rules the business uses in relation to that data.

  Uses the vocabulary of the domain so that a representation of the model can be used to communicate with non-technical stakeholders.

- A precise, concise, understandable, & correct model of the application domain.
- A way to describe and model real world entities and the relationships between them, which collectively describe the problem domain space.[1]

Additional Sources:
[1](http://www.scaledagileframework.com/domain-modeling/)

3-1

Why Create and Use Domain Models?

- Provides a conceptual framework of the problem space.[1]
- Helps bridge the gap between understanding the problem domain and defining requirements for software that will be used to solve problems in the domain.[2]
- Uses vocabulary from the domain in its elements.
  - Therefore provides a common language for domain experts, business personnel, and development staff.
- Since it focuses on important concepts in the problem space and the relations between them, it can be used to show time-invariant business rules that are part of the domain.[1]

Additional Sources:
[1]csis.pace.edu/~marchese/CS389/L8/DomainModel-UML_short.pdf
[2](http://www.scaledagileframework.com/domain-modeling/)

3-2

How do Domain Models fit into Agile Development?

- CS314 has focused on User Stories so far.
  - User stories provide value to users/customers by providing a solution to something in the problem domain.
- User stories comprise the product backlog.
  - Usually several related user stories are gathered to solve a larger problem and make up a product release.
- Agile Epics are containers defined for major initiatives or functionality, which may crosscut multiple releases and multiple development entities.[1]
  - Like User stories, they may be defined strictly in terms of business value, or in terms of enablers, such as large efforts that will be used to enable future business value epics.
  - Like User stories, they have acceptance criteria that must be defined as they become more understood. They also have to be staffed and resources devoted to them until they can be implemented. At this point they are split and defined as many User stories that are added to the product backlogs for all the development groups that will work to implement them.
- Domain models can be used to understand the impact of Epics on the problem domain, and thus be used as rationale or as modifiers of proposed Epics. [1]

Additional Sources:
[1](http://www.scaledagileframework.com/domain-modeling/)

3-3

Epic Template

One Epic template: [1]

For <customers>,

who <do something>,

the <solution>

is a <something, the "how that we want to produce">

that <provides this value> .

Unlike <a competitor, current solution, non-existent solution>,

our solution <does something better, the "why we want to do this">.

Success criteria:

Things that are in-scope for this epic:

Things that are out-of-scope of this epic:

Non-functional requirements:

Additional Sources:
[1](http://www.scaledagileframework.com/domain-modeling/)

3-4

What goes into a Domain Model?

- Import concepts from the problem domain:
  - Concept name that can be a real-world item or a business item (for example, 'Sale' is a business concept, and 'Groceries' is a real-world concept)
  - The intent of the concept (for example, 'Sale represents the event of a purchase transaction and has a date and time')
  - A generalization of the concept (for example, grocery sales, clothing sales, home sales can all be considered as types of the 'Sale' concept)
- Relationships between the concepts:
  - For example, a Student takes a Course at CSU: Student and Course are concepts, and there is a relation between them.
  - There are many kinds of relationships that are possible between concepts.
- We often use the UML to draw domain models, specifically using UML class diagrams:
  - Concepts are represented as classes.
  - Relationships are represented as associations. Depending on the kind of relationship, we can use the different notations that we've used for associations – non-hierarchical, part-of (aggregation), is-a (inheritance), and use dependencies (transient connections).

3-5

Creating a Domain Model

Usually we start from natural language documents.

From Dr. Bieman's notes:

1. Identify concept "classes and objects".
2. Identify key attributes of these concepts.
3. Identify concept responsibilities and attributes.
4. Determine associations and generalizations between concepts.

3-6

Identifying Domain Concepts

Study problem statement & use cases looking for:

- Tangible objects or devices: valve.
- Roles: Supervisor.
- People: Worker.
- Places, locations: Home, Office.
- Other systems.

Domain concepts have:

- Retained information: the entity has state.
- Needed services: it will probably have some operations.
- Attributes are common to all instances of a concept.
- All instances of a concept will have common operations.

3-7

Analyzing Text to find Domain Model Elements

Consider this text about an ATM machine:[1]

The ATM verifies whether the customer's card number and PIN are correct.

…

Checking the balance displays the account balance.

Analyze each subject and object in each sentence:

If it represents a person performing an action then it is an actor. (*Customer*)

If it is a verb then it may be a method. Methods are not included in Domain Models. (*Verify, Check* [balance], *Display*)

If it is a simple value like color (string) or money (number) then it is probably an attribute. ([Card] *Number*, *PIN, Balance*)

Any other nouns may be classes. (*ATM*, *Account, Card*)

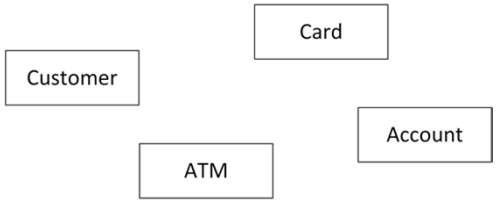
Verbs can also be classes – for example, 'deposit' can either be a noun or a verb depending on context. If it retains state, then it should be a class.

Additional Sources:

[1] csis.pace.edu/~marchese/CS389/L8/DomainModel-UML_short.pdf

3-8

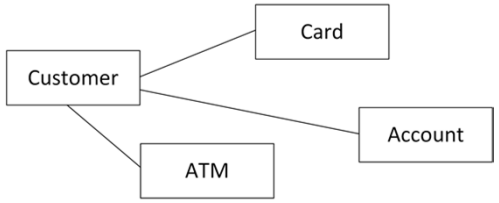
Add the actors and other possible classes to the domain model



Additional Sources:
[1]csis.pace.edu/~marchese/CS389/L8/DomainModel-UML_short.pdf

3-9

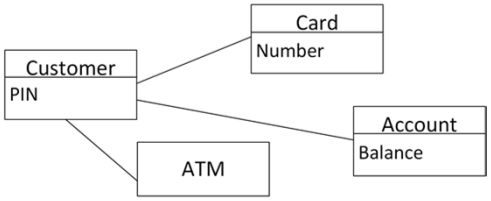
Add associations for relationships that need to be retained



Additional Sources:
[1]csis.pace.edu/~marchese/CS389/L8/DomainModel-UML_short.pdf

3-10

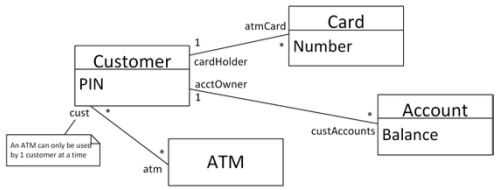
Add attributes for information that needs to be preserved



Additional Sources:
[1]csis.pace.edu/~marchese/CS389/L8/DomainModel-UML_short.pdf

3-11

Add labels and multiplicities to associations



Here we use 'role' labels on the associations. At each end of the association there is a class that plays a role in the relationship. Role names must be nouns.

3-12