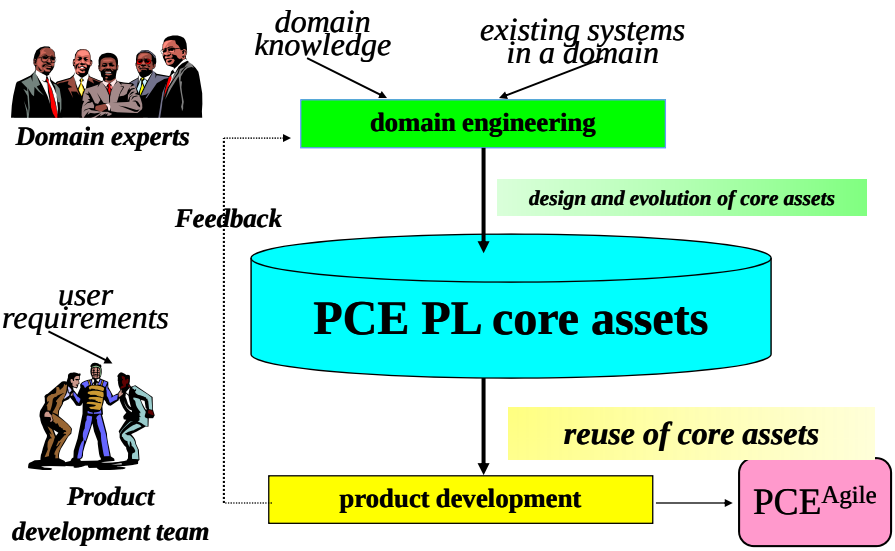
CS6201 Software Reuse

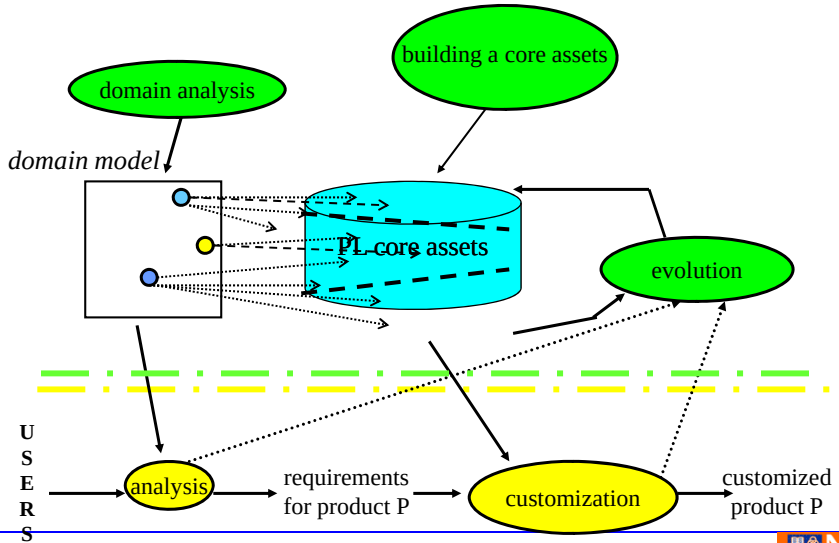
Lecture Notes Set#5: Domain Analysis

1. Domain engineering
2. Modeling a typical system in a domain
   - UML notations
3. Modeling domain variants
   - feature models
   - extending UML notations
4. Case study: Facility Reservation Systems

SPL core assets and their reuse



Domain engineering and system engineering (II)

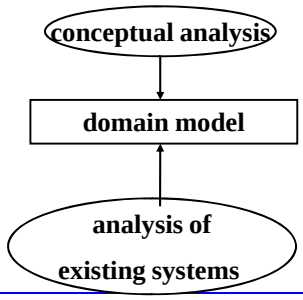


Domain analysis

- Modeling PL requirements
  - common (mandatory) and variant features
  - functional and non-functional properties
- Modeling feature dependencies
  - legal configurations of features in products
- Modeling products (e.g., UML)
  - feature semantics

Domain analysis process

- top-down:
  - expert knowledge
  - conceptual analysis of similarities and differences in a domain
- bottom-up: analysis of existing systems



How to analyze a domain?

- Model typical product
  - default product models
  - e.g., using UML diagrams
- Model variant features
  - e.g., using feature diagrams
- Describe how variants affect product models
  - e.g., using extended UML diagrams

Facility Reservation Systems (FRS) Product Line

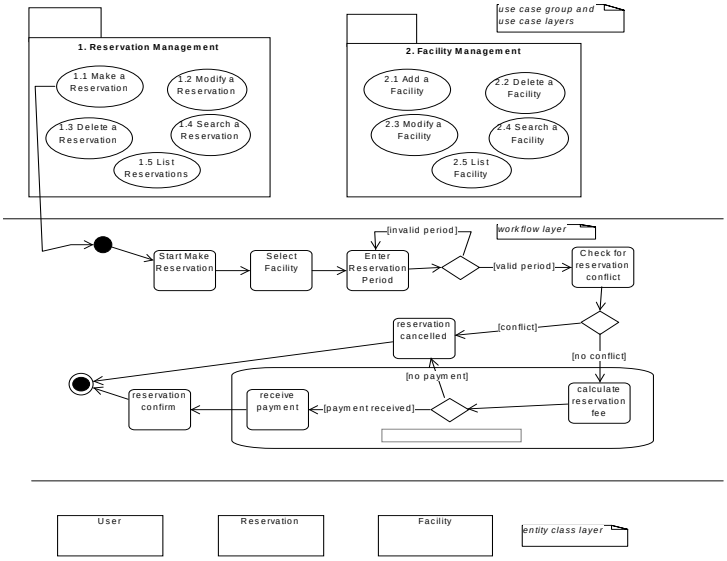
- FRSes for: universities, hotels, offices, …
- Requirements for FRS PL:
  - Common features:
    - Reservation Management
    - Facility Management
  - Variant features:
    - User Management
    - User rights definition
    - Payment

Common and variant FRS features

An FRS helps company staff manage (add, delete, modify) reservation of facilities. Facilities are rooms and equipment (such as PCes or OHPs).

- ± Different companies have different physical facilities and rules for their reservation.
- ± FRS may allow one to view reservations by facility and/or by date
- ± FRS may allow one to make reservations with or without a preferred facility in mind
- ± FRS may allow one manage users and facilities
- ± Payment may be required for reservation and cancellation in certain companies. Printing bills may be required

Default product models



FRS default product

Reservation Management:

- Make a Reservation
- Delete a Reservation
- Modify a Reservation
- Search/Retrieve Reservations

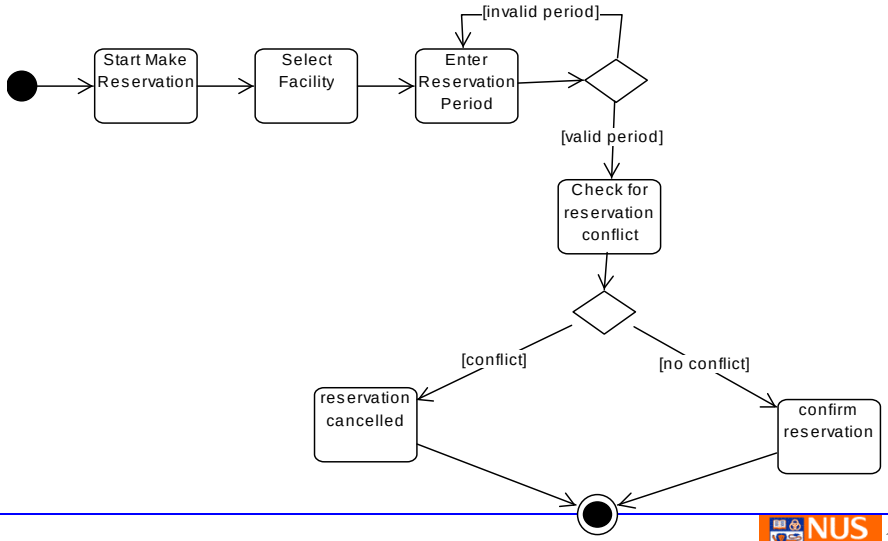
Facility Management:

- Add a Facility
- Delete a Facility
- Modify a Facility
- Search/Retrieve Facilities

Default use case *Make Reservation*

If USER has a preference
    USER searches through FACILITY
IF USER does not have a preference
    System searches through FACILITY
System selects FACILITY to reserve on behalf of user
USER supplies time period of reservations
If FACILITY is reserved at the specified time
    USER repeats the reservation use case
// *Special Requirements*:
System should respond to use input within RESPONSETIME
// The default values of the variables are listed below:
USER = user
FACILITY = facility
RESPONSETIME = 30secs

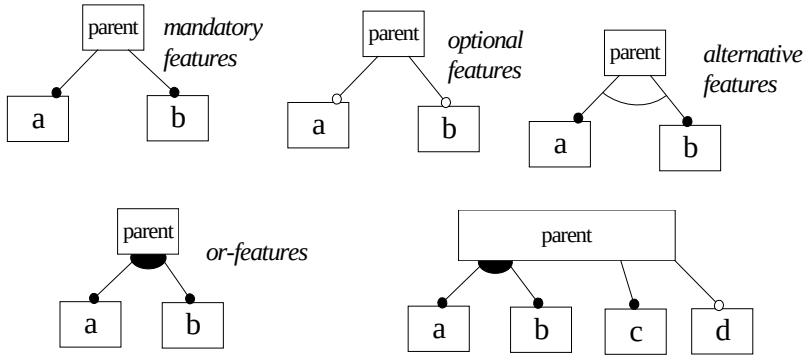
Default workflow *Make Reservation*



Feature diagrams

Feature diagrams

diagrams depict mandatory and variant features as a tree-like hierarchy



Examples of optional features

Cancellation_Fee–OPT // the system shall charge a certain amount whenever a reservation is cancelled within "n" days of the reservation

Conflict_Resolution–OPT // a conflict arises if a new reservation would result in a clash with other reservation(s); the system will perform some form of conflict resolution

Suggest_Other_Facility–OPT // when a conflict arises, the system shall suggest other similar facilities that are available for the time period required

Retrieve_Available_Facilities–OPT // given a particular time period, the system shall produce a list of facilities available during this time period.

OR-features

OR-features indicate that any number of many options

may be selected

FRS may allow one to retrieve reservations by date, by facility or by both date and facility

Retrieve_Reservations_by : Date-OR

Retrieve_Reservations_by : Facility-OR

FRS may allow one to make reservation with or without preferred facility in mind

Make_Reservation_with : No_Pref_Facility-OR

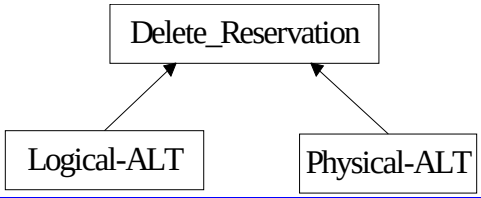
Make_Reservation_with : Pref_Facility-OR

Alternative features

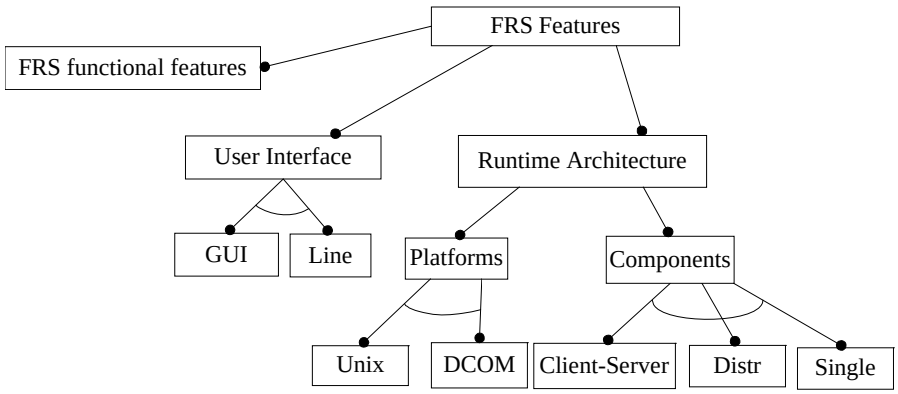
Delete_Reservation
- FRS will allow existing reservation to be deleted

Delete_Reservation : Logical-ALT
- FRS logically deletes reservations, meaning that reservations will just be marked as deleted

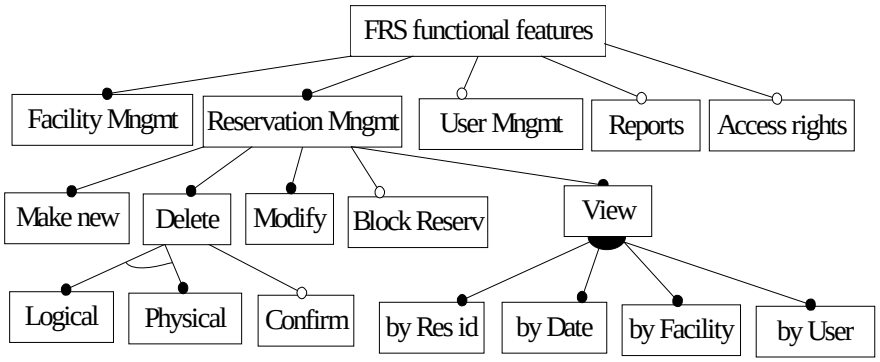
Delete_Reservation : Physical-ALT
- FRS physically deletes reservations



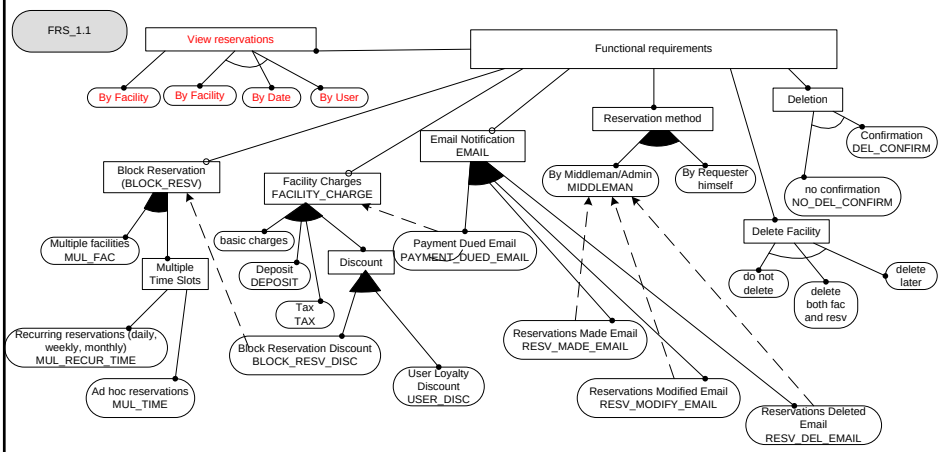
Features in the FRS domain



Functional features in FRS domain



Feature diagrams with dependencies



Feature dependencies: payment

- only certain configurations of features are legal

For example, FRS may have features blow only if it also has feature payment

- reservation cost, computation of payment
- discounts
- printing bills
- cancellation fee

We say that above features depend on payment

Specifying feature dependencies

R1 depends on R2: $R_1 \Rightarrow R_2$

if a product has feature R_1 then it must also have R_2

Bill_Construction–OPT $\Rightarrow$ Payment–OPT

Cancellation_Fee–OPT $\Rightarrow$ Payment–OPT

Suggest_Other_Facility–OPT $\Rightarrow$ Conflict_Resolution–OPT &
Retrieve_Available_Facilities–OPT

Specifying feature dependencies

In any given product:

$R_1 \Rightarrow R_2$ has R_1, then it must have R_2

$R_1 \Rightarrow$! R_2 has R_1, then it must not have R_2

$R_1 \Rightarrow$ < R_2 | R_3 | … | R_N > has R_1, then it must also have at least one of the features R_2, R_3, …, R_N

$R_1 \Rightarrow$ < R_2 & R_3 & … & R_N > has R_1, then it must also have all features R_2, R_3, …, R_N

[R1, R2, …, R_N] must either have all of the R_1, R_2, …, R_N or none of them

Modeling the impact of features on product models

Modeling how variants affect default product model

- use cases and scenarios
- information model (Entity-Relationship or class diagrams)
- activity diagrams
- State Transition diagrams
- Data Flow diagrams
- and many others

as these notations are meant for modeling one-of-the-kind system; we have to modify the notations to model variants

Use cases with variants

Use case for Making Reservation

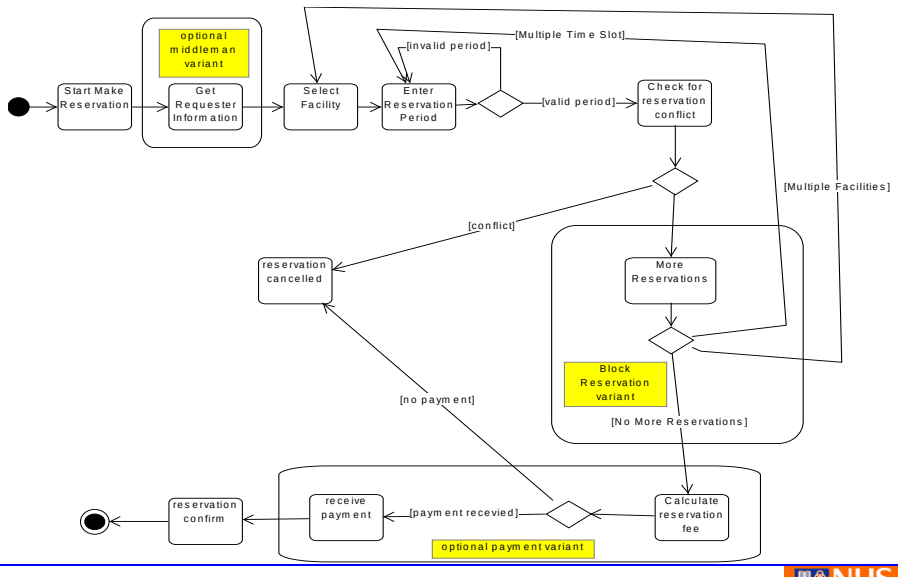
1. (No_Pref_Facility-OR): User has no preference for a facility:
   1.1. User supplies time period of the reservation.
   1.2. (View_Available_Facilities–OPT): System supplies a list of available facilities
   1.3. User indicates a particular facility (from this list).
   1.4. (Block_Reservation–OPT): User repeats step 1.3 (at least once) for each additional facility to be reserved.
2. (Pref_Facility–OR): User has preference for a facility:
   2.1. User specifies the facility to be reserved.
   2.2. System verifies that the facility specified actually exists.
   2.3. (Block_Reservation–OPT):...

Comments on use cases

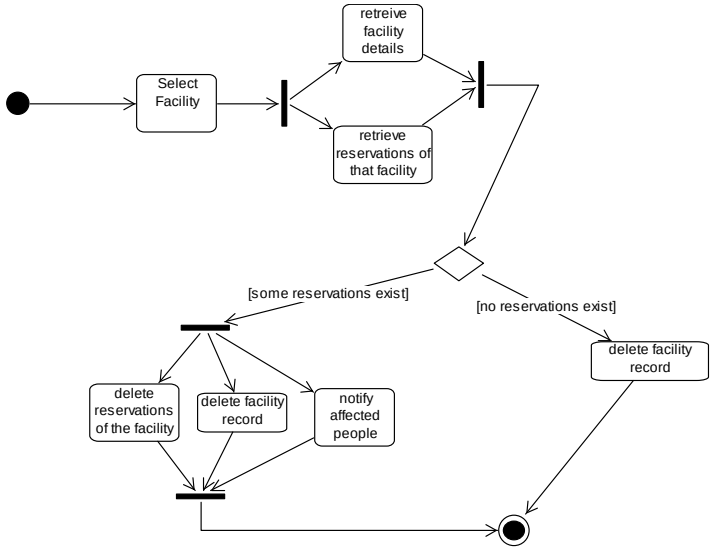
A use case is an outline of operations that produces some system behavior visible to the user. We cross-reference use cases with features in two ways. Firstly, we indicate a feature that gives raise to a given use case. Secondly, we label steps in use cases with feature names: a step labeled with a variant feature in brackets ("<" and ">") is executed only if the feature holds in a given FRS. Consider different use cases for making reservations. One possible use case is that the user does not have a preference for a facility to be reserved (see above). This use case corresponds to a variant requirement No_Pref-Facility-OR. If a gibe FRS has that feature, then the lower-level steps (i.e., steps 1.1 to 1.4) describe the relevant action.

A block reservation option allows a user to reserve many facilities for the same period of time.

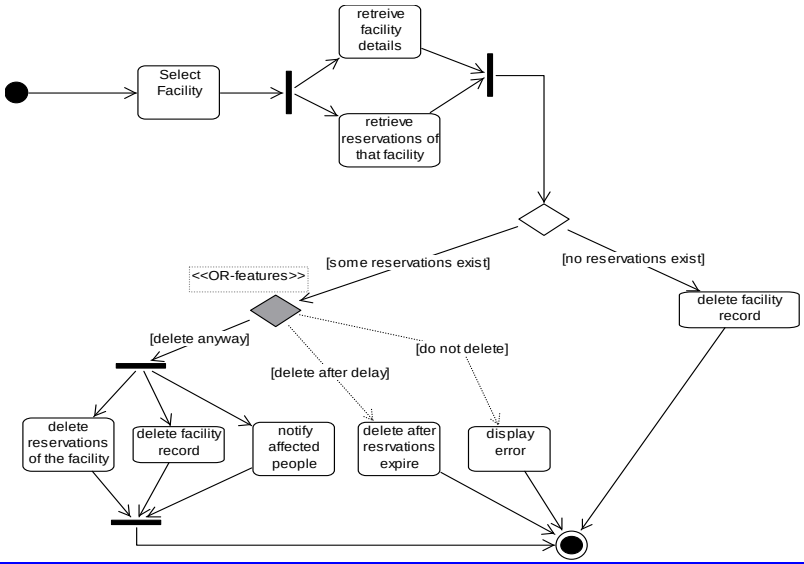
Make Reservation workflow with variants



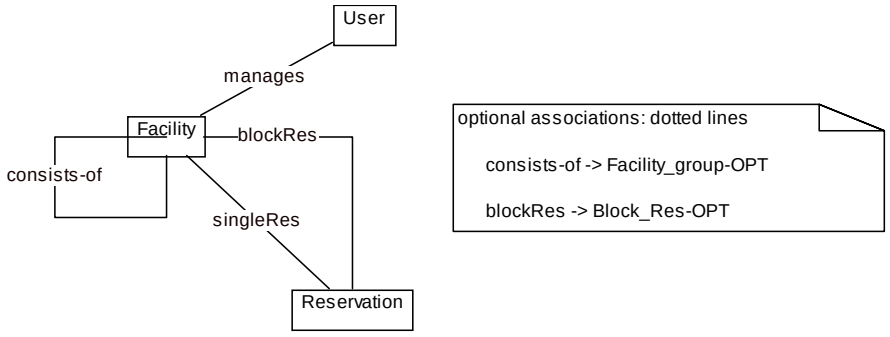
Delete Facility default workflow



Delete Facility workflow with variants



Class diagram

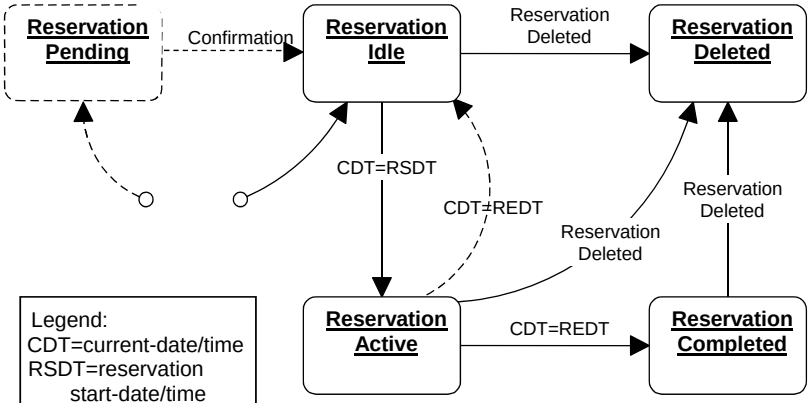


Comments

Class diagram

A class diagram describes the FRS domain in terms of the entities involved and the relationships among them. Dotted links represent optional relationships that correspond to variant features. Optional relationships are marked with names of corresponding variant features. Classifications of domain concepts (so-called taxonomies) play an important role in domain modeling. We can conveniently express taxonomies using class inheritance. We can construct a number of taxonomies for entities such as facilities, users and reservations. The diagram shows such a taxonomy for facilities. Different types of rooms (e.g. conference room and classroom) can be generalized into a single "room" type. Likewise, rooms, equipment, seats and beds can all be generalized as a "facility" that can be reserved.

State Transition Diagrams



Reservation Pending -> Confirm_by_User–OPT

Confirmation -> Confirm_by_User–OPT

CDT=REDT -> Recurring_Reservation–OPT

Comments

A state transition diagram describes the events and states in the FRS domain. It allows us to capture the dynamic behavior of a system. Variant features in a domain cause that some states and transition can be optional (i.e., they may describe the behavior of one FRS but not all of them). We extended the usual state transition notation to deal with the variability of states and events:

Dotted box denotes a state that only exists if a given FRS has the associated variant feature

Dotted arrow denotes a transition that only occurs if a given FRS has the associated variant feature

The above diagram shows how we modeled the change in the state of reservations over time. In some companies, a requester must confirm a reservation, say 2 days before the reservation date, (variant feature Confirm_by_User-OPT). Reservations that have not been confirmed yet are called pending reservations. State **Reservation Pending** corresponds to these reservations. The leftmost arrow on the left shows an optional state and a dotted arrow points to optional starting state.

Reservation Idle represents reservations (possibly confirmed) yet waiting for realization.

During the Reservation period, a reservation remains in the state **Reservation Active**.

Recurring reservation is a reservation of the same facility for a number of time periods (e.g., every Monday from 10 till 12). Capability of making recurring reservation is an optional feature (Recurring_Reservation-OPT).

Desirable qualities of a domain model

- Easy to understand
- Sufficiently expressive: domain model should capture various types of user requirements: functional feature, behavior, non-functional requirements, constraints
- Domain model should capture commonality and variants
- Modular domain model
- Support for multiple levels of abstraction
- Scaleable - the domain model should be able to capture a large volume of domain information
- Implementable - the domain model should provide direct guidelines for design of a PL architecture

Domain modeling: Lessons learnt

- A domain model should:
  - provide guidelines for the design of a PL architecture
  - describe a range of functions and variants implemented within a PL architecture for reuse
- You must be innovative to model domains
- Different domains may require different modeling techniques
- Existing notations can be augmented to model variants
- Different views of a model must be cross-referenced for traceability
- How to make a domain model understandable as the volume of information grows, with complex dependencies among variants?

Q & A

End of Set #5 Domain Analysis