MAJOR UNIT: ANALYSIS

In 1915, Rubin exhibited a drawing similar to the figure below to illustrate the concept of multi-stable images. What do you see? Two faces looking at each other? If you focus more closely on the white area, you can see a vase instead. Once you are able to perceive both shapes individually, it is easier to switch back and forth between the vase and the faces.

Requirement specifications, like multi-stable images, contain ambiguities caused by the inaccuracies inherent to natural language and by the assumptions of the specification authors. For example, a quantity specified without a unit is ambiguous (e.g., the "Feet or Miles?"), a time without time zone is ambiguous (e.g., scheduling a phone call between different countries).

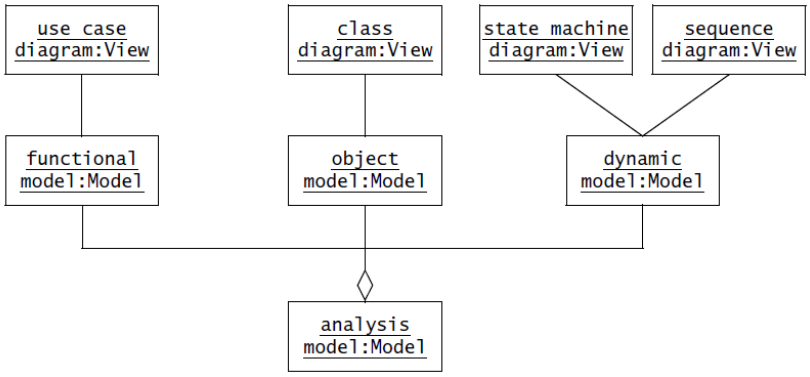
Formalization helps identify areas of ambiguity as well as inconsistencies and omissions in a requirements specification. Once developers identify problems with the specification, they address them by eliciting more information from the users and the client. Requirements elicitation and analysis are iterative and incremental activities that occur concurrently.

An Overview of Analysis

Analysis focuses on producing a model of the system, called the analysis model, which is correct, complete, consistent, and verifiable. Analysis is different from requirements elicitation in that developers focus on structuring and formalizing the requirements elicited from users. This formalization leads to new insights and the discovery of errors in the requirements.

The **analysis model** is composed of three individual models:

- the **functional model**, represented by use cases and scenarios,
- the **analysis object model**, represented by class and object diagrams, and
- the **dynamic model**, represented by state machine and sequence diagrams.



Major Analysis Concepts

- Analysis Object Models and Dynamic Models
- Entity, Boundary, and Control Objects
- Generalization and Specialization

Analysis Object Models and Dynamic Models

The analysis model represents the system under development from the user's point of view.

- The **analysis object model** is a part of the analysis model and focuses on the individual concepts that are manipulated by the system, their properties and their relationships. The analysis object model, depicted with UML class diagrams, includes classes, attributes, and operations. The analysis object model is a visual dictionary of the main concepts visible to the user.

- The **dynamic model** focuses on the behavior of the system. The dynamic model is depicted with sequence diagrams and with state machines. Sequence diagrams represent the interactions among a set of objects during a single use case. State machines represent the behavior of a single object (or a group of very tightly coupled objects). The dynamic model serves to assign responsibilities to individual classes and, in the process, to identify new classes, associations, and attributes to be added to the analysis object model.

2BWatch Scenario

Setting the time on 2Bwatch requires the actor 2BWatchOwner to first press both buttons simultaneously, after which 2Bwatch enters the set time mode. In the set time mode, 2Bwatch blinks the number being changed (e.g., the hours, minutes, seconds, day, month, or year). Initially, when the 2BWatchOwner enters the set time mode, the hours blink. If the actor presses the first button, the next number blinks (e.g, if the hours are blinking and the actor presses the first button, the hours stop blinking and the minutes start blinking). If the actor presses the second button, the blinking number is incremented by one unit. If the blinking number reaches the end of its range, it is reset to the beginning of its range (e.g., assume the minutes are blinking and its current value is 59, its new value is set to 0 if the actor presses the second button). The actor exits the set time mode by pressing both buttons simultaneously.

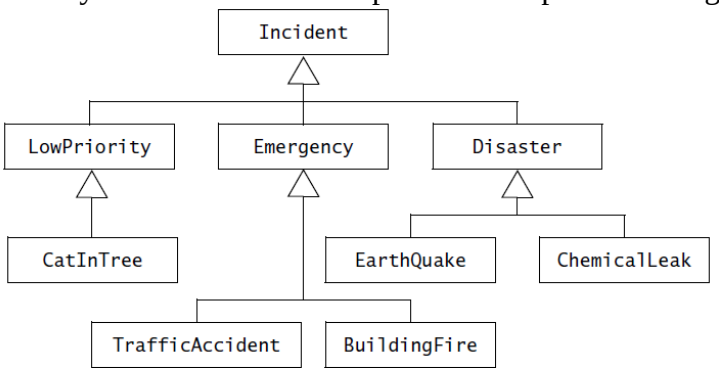
Entity, Boundary, and Control Objects

The analysis object model consists of entity, boundary, and control objects [Jacobson et al., 1999].

- **Entity objects** represent the persistent information tracked by the system.
- **Boundary objects** represent the interactions between the actors and the system.
- **Control objects** are in charge of realizing use cases.
- In the 2Bwatch example, Year, Month, and Day are entity objects; Button and LCDDisplay are boundary objects; ChangeDateControl is a control object that represents the activity of changing the date by pressing combinations of buttons.

Generalization & Specialization

- **Generalization** is the modeling activity that identifies abstract concepts from lower-level ones.
- **Specialization** is the activity that identifies more specific concepts from a high-level one.



Analysis Activities: From Use Cases to Objects

Here, we shall describe some activities that transform the use cases and scenarios produced during requirements elicitation into an analysis model. Some analysis activities include:

- Identifying Entity Objects
- Identifying Boundary Objects
- Identifying Control Objects

- Identifying Associations
- Identifying Aggregates
- Identifying Attributes

Much of this is dependent on natural language processing. Natural language analysis [Abbott, 1983] is an intuitive set of heuristics for identifying objects, attributes, and associations from a requirements specification. Abbott's heuristics maps parts of speech (e.g., nouns, having verbs, being verbs, adjectives) to model components (e.g., objects, operations, inheritance relationships, classes). However personal practical experience is also used in conjunction with the NLP.

Exercise

Use case name ReportEmergency

Entry condition:

- The FieldOfficer activates the "Report Emergency" function of her terminal.

Flow of events

- FRIEND responds by presenting a form to the officer. The form includes an emergency type menu (general emergency, fire, transportation), a location, incident description, resource request, and hazardous material fields.
- The FieldOfficer completes the form by specifying minimally the emergency type and description fields. The FieldOfficer may also describe possible responses to the emergency situation and request specific resources. Once the form is completed, the FieldOfficer submits the form by pressing the "Send Report" button, at which point, the Dispatcher is notified.
- The Dispatcher reviews the information submitted by the FieldOfficer and creates an Incident in the database by invoking the OpenIncident use case. All the information contained in the FieldOfficer's form is automatically included in the incident. The Dispatcher selects a response by allocating resources to the incident (with the AllocateResources use case) and acknowledges the emergency report by sending a FRIENDgram to the FieldOfficer.

Exit condition

- The FieldOfficer receives the acknowledgment and the selected response.

Abbott's heuristics for mapping parts of speech to model components [Abbott, 1983]

Part of speech	Model component	Examples
Proper noun	Instance	Alice
Common noun	Class	Field officer
Doing verb	Operation	Creates, submits, selects
Being verb	Inheritance	Is a kind of, is one of either
Having verb	Aggregation	Has, consists of, includes
Modal verb	Constraints	Must be
Adjective	Attribute	Incident description

Identifying Entity Objects

Following Heuristics may be used for identifying entity objects along with the ones given by Abbott:

- Terms that developers or users need to clarify in order to understand the use case
- Recurring nouns in the use cases
- Real-world entities that the system needs to track
- Real-world activities that the system needs to track

- Data sources or sinks.

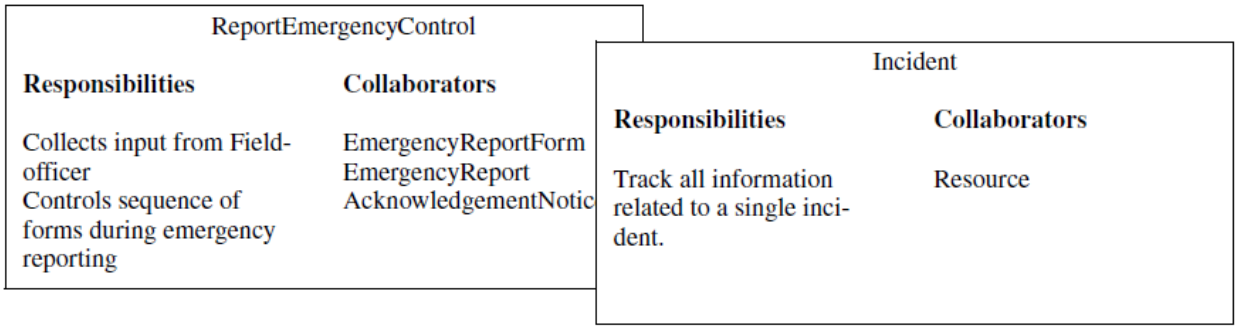
Identifying Boundary Objects
Boundary objects represent the system interface with the actors. In each use case, each actor interacts with at least one boundary object. The boundary object collects the information from the actor and translates it into a form that can be used by both entity and control objects.

Following Heuristics may be used for identifying boundary objects along with the ones given by Abbott:
- Identify user interface controls that the user needs to initiate the use case.
- Identify forms the users needs to enter data into the system.
- Identify notices and messages the system uses to respond to the user.
- When multiple actors are involved in a use case, identify actor terminals to refer to the user interface under consideration.
- Do not model the visual aspects of the interface with boundary objects.
- *Always* use the end user's terms for describing interfaces; do not use terms from the solution or implementation domains.

Identifying Control Objects
Following Heuristics may be used for identifying control objects along with the ones given by Abbott:

- Identify one control object per use case.
- Identify one control object per actor in the use case.
- The life span of a control object should cover the extent of the use case or the extent of a user session. If it is difficult to identify the beginning and the end of a control object activation, the corresponding use case probably does not have well-defined entry and exit conditions.

Modeling Interactions among Objects with CRC Cards
An alternative for identifying interactions among objects are **CRC cards**. CRC cards (CRC stands for class, responsibilities, and collaborators) were initially introduced as a tool for teaching object-oriented concepts to novices and to experienced developers unfamiliar with object-orientation. Each class is represented with an index card (called the CRC card). The name of the class is depicted on the top, its responsibilities in the left column, and the names of the classes it needs to accomplish its responsibilities are depicted in the right column.



Documenting Analysis

Requirement Analysis Document is used for documenting the "analysis object model".

Requirements Analysis Document

1. Introduction
2. Current system
3. Proposed system
   - 3.1 Overview
   - 3.2 Functional requirements
   - 3.3 Nonfunctional requirements
   - 3.4 System models
     - 3.4.1 Scenarios
     - 3.4.2 Use case model
     - 3.4.3 Object model
       - 3.4.3.1 Data dictionary
       - 3.4.3.2 Class diagrams
     - 3.4.4 Dynamic models
     - 3.4.5 User interface—navigational paths and screen mock-ups
4. Glossary

Solution to Exercise

Entity Objects

- **Dispatcher** Police officer who manages Incidents. A Dispatcher opens, documents, and closes Incidents in response to Emergency Reports and other communication with FieldOfficers. Dispatchers are identified by badge numbers.
- **EmergencyReport** Initial report about an Incident from a FieldOfficer to a Dispatcher. An EmergencyReport usually triggers the creation of an Incident by the Dispatcher. An EmergencyReport is composed of an emergency level, a type (fire, road accident, other), a location, and a description.
- **FieldOfficer** Police or fire officer on duty. A FieldOfficer can be allocated to, at most, one Incident at a time. FieldOfficers are identified by badge numbers.
- **Incident** Situation requiring attention from a FieldOfficer. An Incident may be reported in the system by a FieldOfficer or anybody else external to the system. An Incident is composed of a description, a response, a status (open, closed, documented), a location, and a number of FieldOfficers.

Boundary Objects

- **AcknowledgmentNotice** Notice used for displaying the Dispatcher's acknowledgment to the FieldOfficer.
- **DispatcherStation** Computer used by the Dispatcher.
- **ReportEmergencyButton** Button used by a FieldOfficer to initiate the ReportEmergency use case.
- **EmergencyReportForm** Form used for the input of the ReportEmergency. This form is presented to the FieldOfficer on the FieldOfficerStation when the "Report Emergency" function is selected. The EmergencyReportForm contains fields for specifying all attributes of an emergency report and a button (or other control) for submitting the completed form.
- **FieldOfficerStation** Mobile computer used by the FieldOfficer.
- **IncidentForm** Form used for the creation of Incidents. This form is presented to the Dispatcher on the DispatcherStation when the EmergencyReport is received. The Dispatcher also uses this form to allocate resources and to acknowledge the FieldOfficer's report.

Control Objects

- **ReportEmergencyControl** Manages the ReportEmergency reporting function on the FieldOfficerStation. This object is created when the FieldOfficer selects the "Report Emergency" button. It then creates an EmergencyReportForm and presents it to the FieldOfficer. After submitting the form, this object then collects the information from the form, creates an EmergencyReport, and forwards it to the Dispatcher. The control object then waits for an acknowledgment to come back from the DispatcherStation. When the acknowledgment is received, the ReportEmergencyControl object creates an AcknowledgmentNotice and displays it to the FieldOfficer.
- **ManageEmergencyControl** Manages the ReportEmergency reporting function on the DispatcherStation. This object is created when an EmergencyReport is received. It then creates an IncidentForm and displays it to the Dispatcher. Once the Dispatcher has created an Incident, allocated Resources, and submitted an acknowledgment, ManageEmergencyControl forwards the acknowledgment to the FieldOfficerStation.

Exercise: For the following scenario, identify the entity, boundary object and control object.

Name **PSLTournament**

Flow of events

1. The **LeagueOwner** requests the creation of a **tournament**.
2. The system checks if the LeagueOwner has exceeded the **number of tournaments** in the **league** or in the **arena**. If not, the system presents the LeagueOwner with a form.
3. The LeagueOwner specifies a **name**, **application start and end dates** during which Players can apply to the tournament, **start and end dates** for conducting the tournament, and a **maximum number of Players**.
4. The system asks the LeagueOwner whether an exclusive sponsorship should be sought and, if yes, presents a **list of Advertisers** who expressed the desire to be **exclusive sponsors**.
5. If the LeagueOwner decides to seek an exclusive sponsor, he selects a subset of the **names** of the **proposed sponsors**.
6. The system notifies the selected sponsors about the upcoming tournament and the **flat fee** for exclusive sponsorships.
7. The system communicates their **answers** to the LeagueOwner.
8. If there are interested sponsors, the LeagueOwner selects one of them.
9. The system records the **name** of the exclusive sponsor and charges the flat fee for sponsorships to the **Advertiser**'**s account**. From now on, all **advertisement banners** associated with the tournament are provided by the exclusive sponsor only.
10. If no sponsors were selected (either because no Advertisers were interested or the LeagueOwner did not select any), the advertisement banners are selected at random and charged to each Advertiser's account on a per unit basis.
11. Once the sponsorship issues is closed, the system prompts the LeagueOwner with a **list of groups of Players, Spectators, and Advertisers** that could be interested in the new tournament.
12. The LeagueOwner selects which groups to notify.
13. The system creates a home page in the arena for the tournament. This page is used as an entry point to the tournament (e.g., to provide interested Players with a form to apply for the tournament, and to interest Spectators into watching **matches**).
14. At the application start date, the system notifies each interested user by sending them a link to the main tournament page. The Players can then apply for the tournament with the ApplyForTournament use case until the application end date.