ICT - 4231
OBJECT ORIENTED SOFTWARE ENGINEERING

LECTURE - 5

CHAPTER – 5 : ANALYSIS

Moinul Islam, IIT, JU

CONTENTS

INTRODUCTION

“ *I am Foo with a name, if I could only remember it.* ”

-----A programmer of very little brain

INTRODUCTION

- Analysis results in a model of the system that aims to be correct, complete, consistent, and unambiguous.
- Developers formalize the requirements specification produced during requirements elicitation and examine in more detail boundary conditions and exceptional cases. Developers validate, correct and clarify the requirements specification if any errors or ambiguities are found.
- The client and the user are usually involved in this activity when the requirements specification must be changed and when additional information must be gathered.
- In object-oriented analysis, developers build a model describing the application domain.

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.
- As the analysis model may not be understandable to the users and the client, developers need to update the requirements specification to reflect insights gained during analysis, then review the changes with the client and the users.
- In the end, the requirements, however large, should be understandable by the client and the users.

AN OVERVIEW OF ANALYSIS

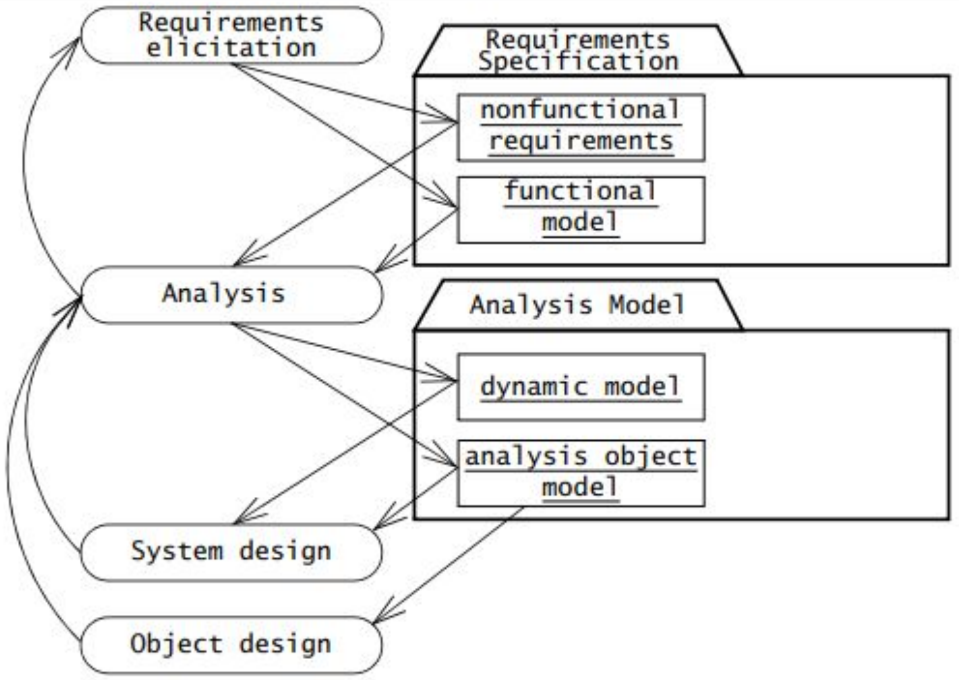


Figure 5-2 Products of requirements elicitation and analysis (UML activity diagram).

AN OVERVIEW OF ANALYSIS

- There is a natural tendency for users and developers to postpone difficult decisions until later in the project.
- A decision may be difficult because of lack of domain knowledge, lack of technological knowledge, or simply because of disagreements among users and developers.
- Postponing decisions enables the project to move on smoothly and avoids confrontation with reality or peers.
- Unfortunately, difficult decisions eventually must be made, often at higher cost when intrinsic problems are discovered during testing, or worse, during user evaluation.
- Translating a requirements specification into a formal or semiformal model forces developers to identify and resolve difficult issues early in the development.
- 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

AN OVERVIEW OF ANALYSIS

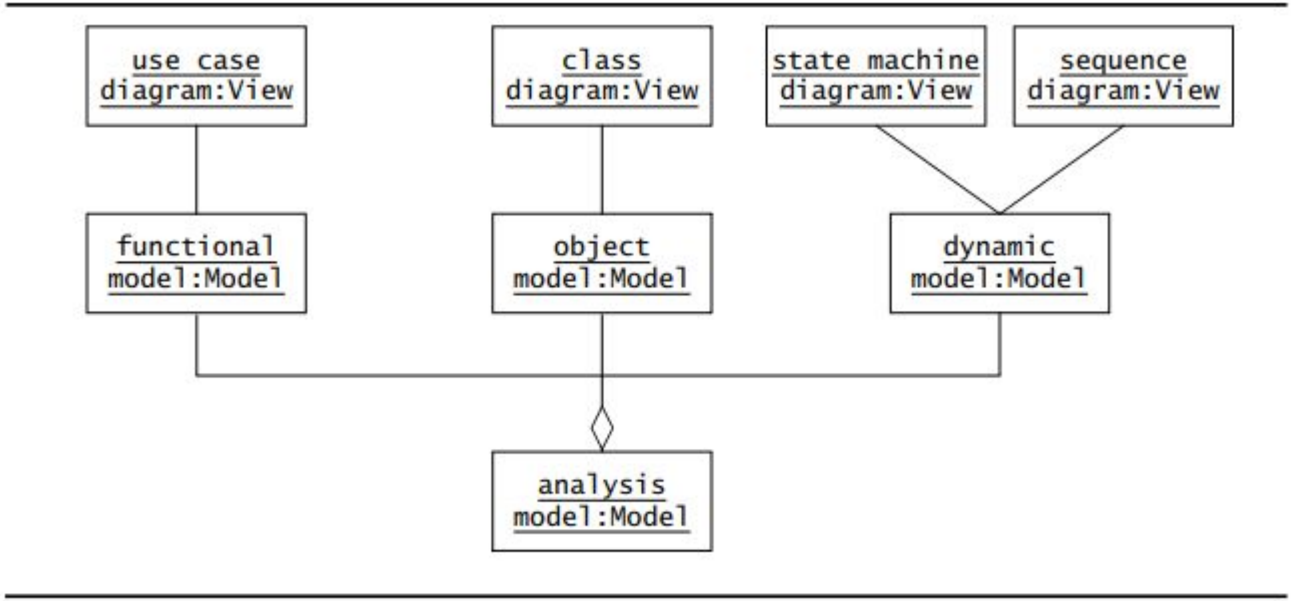


Figure 5-3 The analysis model is composed of the functional model, the object model, and the dynamic model. In UML, the functional model is represented with use case diagrams, the object model with class diagrams, and the dynamic model with state machine and sequence diagrams.

ANALYSIS CONCEPTS

Analysis concepts include the following topics:

- 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.

ANALYSIS OBJECT MODELS AND DYNAMIC MODELS

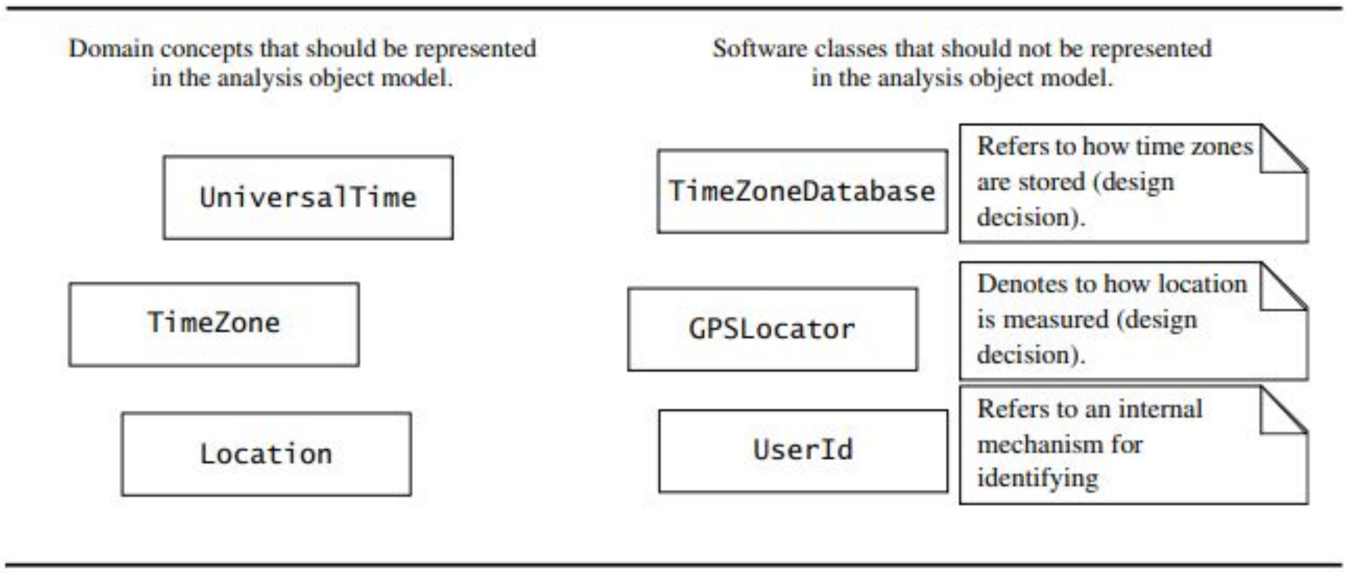

Domain concepts that should be represented in the analysis object model.
Software classes that should not be represented in the analysis object model.
UniversalTime
TimeZone
Location
TimeZoneDatabase
Refers to how time zones are stored (design decision).
GPSLocator
Denotes to how location is measured (design decision).
UserId
Refers to an internal mechanism for identifying

Figure 5-4 Examples and counterexamples of classes in the analysis object model of `SatWatch`.

ENTITY, BOUNDARY, AND CONTROL OBJECTS

- **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.

Modeling systems with entity, boundary, and control objects provides clear distinctions between concepts like time and its display. This approach results in smaller, more specialized objects, making models resilient to change. By separating interface (boundary objects) from core functionality (entity and control objects), most of the model remains unchanged during interface modifications.

ENTITY, BOUNDARY, AND CONTROL OBJECTS

UML employs stereotypes to differentiate object types, allowing developers to attach meta-information to modeling elements. For instance, in Figure 5-5, the «control» stereotype is applied to the ChangeDateControl object. Additionally, naming conventions aid clarity, suggesting that control objects may have "Control" appended to their names, boundary objects may denote interface features (e.g., Form, Button, Display, or Boundary), and entity objects typically lack suffixes. This convention enhances clarity and ensures class types are evident even without UML stereotypes, such as when reviewing source code.

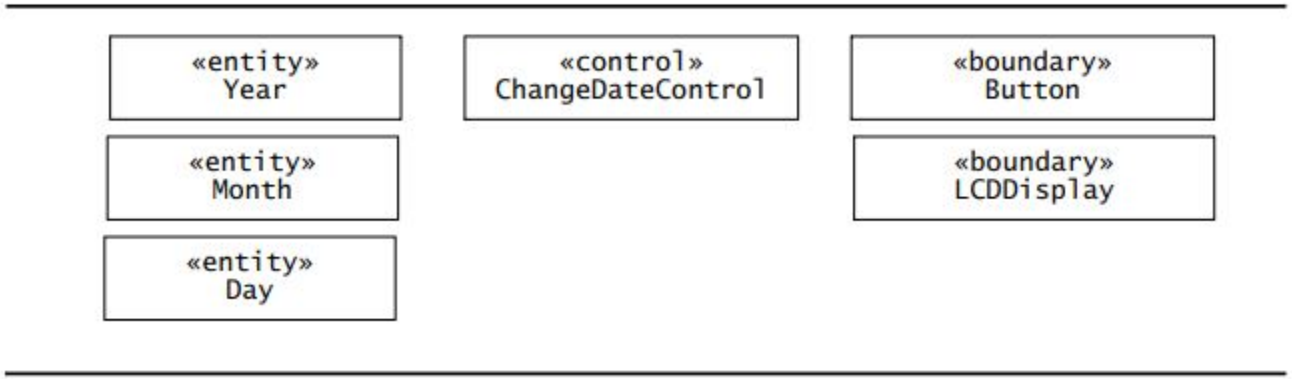


Figure 5-5 Analysis classes for the 2Bwatch example.

GENERALIZATION AND SPECIALIZATION

Generalization is the modeling activity that identifies abstract concepts from lower-level ones. For example, assume we are reverse-engineering an emergency management system and discover screens for managing traffic accidents and fires. Noticing common features among these three concepts, we create an abstract concept called Emergency to describe the common (and general) features of traffic accidents and fires.

Specialization is the activity that identifies more specific concepts from a high-level one. For example, assume that we are building an emergency management system from scratch and that we are discussing its functionality with the client. The client first introduces us with the concept of an incident, then describes three types of Incidents: Disasters, which require the collaboration of several agencies, Emergencies, which require immediate handling but can be handled by a single agency, and LowPriorityIncidents, that do not need to be handled if resources are required for other, higher-priority Incidents.

In both cases, generalization and specialization result in the specification of inheritance relationships between concepts. In some instances, modelers call inheritance relationships generalization-specialization relationships. In this book, we use the term “inheritance” to denote the relationship and the terms “generalization” and “specialization” to denote the activities that find inheritance relationships.

GENERALIZATION AND SPECIALIZATION

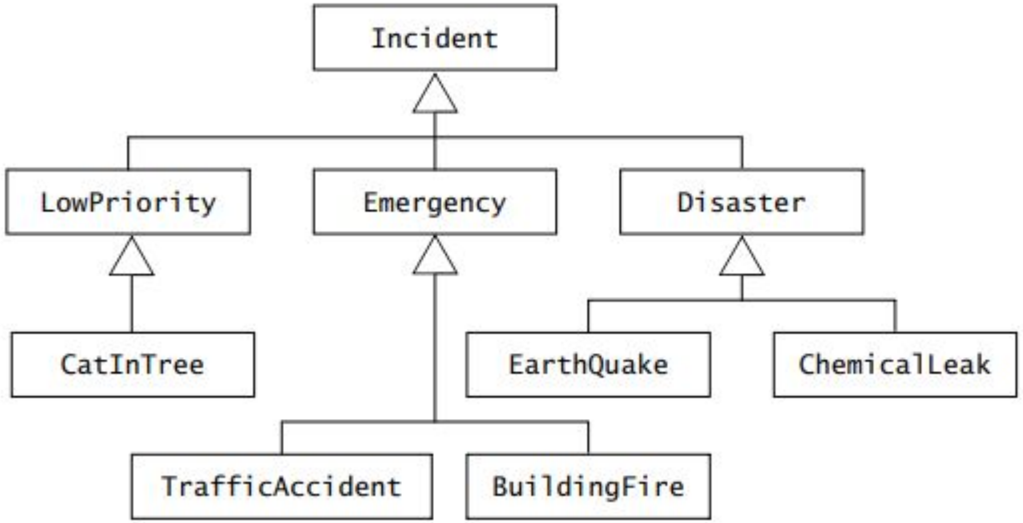


Figure 5-6 An example of a generalization hierarchy (UML class diagram). The top of the hierarchy represents the most general concept, whereas the bottom nodes represent the most specialized concepts.

ANALYSIS ACTIVITIES

Analysis activities include:

- Identifying Entity Objects
- Identifying Boundary Objects
- Identifying Control Objects
- Mapping Use Cases to Objects with Sequence Diagrams
- Modeling Interactions among Objects with CRC Cards
- Identifying Associations
- Identifying Aggregates
- Identifying Attributes
- Modeling State-Dependent Behavior of Individual Objects
- Modeling Inheritance Relationships
- Reviewing the Analysis Model

IDENTIFYING ENTITY OBJECTS

Participating objects form the basis of the analysis model. Requirements Elicitation, participating objects are found by examining each use case and identifying candidate objects. 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). Table 5-1 provides examples of such mappings by examining the ReportEmergency use case (Figure 5-7)

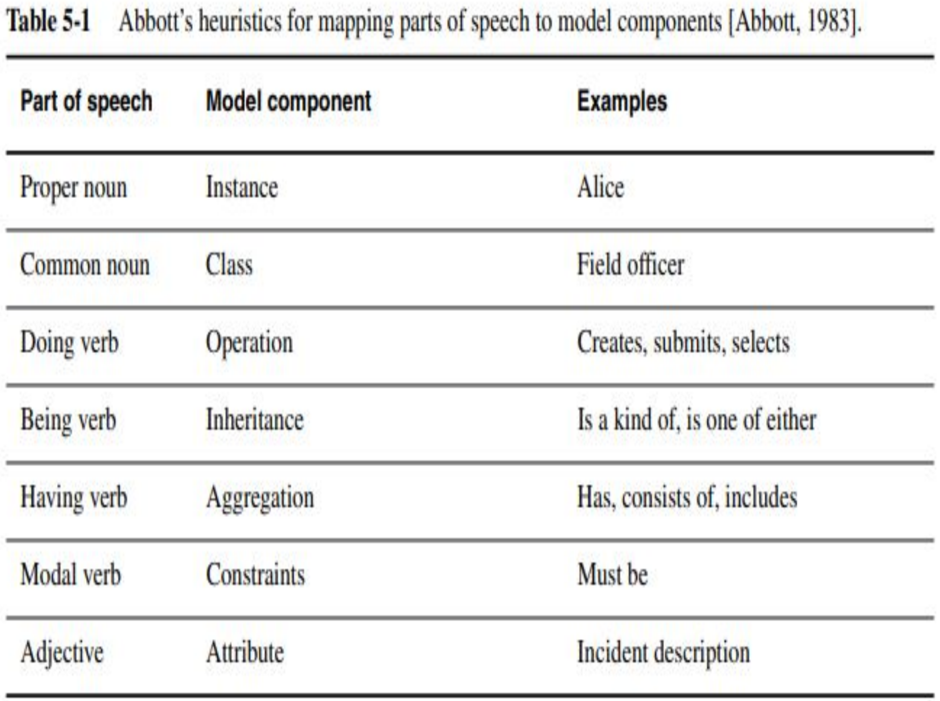


Table 5-1 Abbott's heuristics for mapping parts of speech to model components [Abbott, 1983].

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

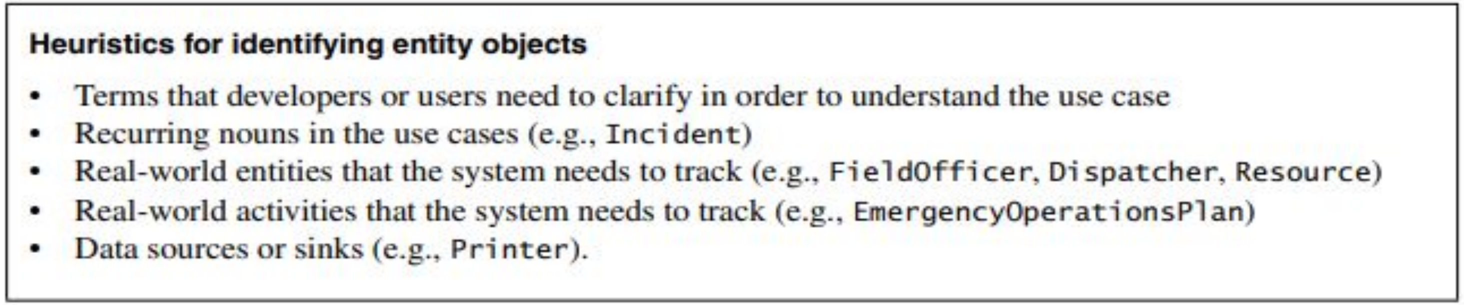
Natural language analysis has the advantage of focusing on the users' terms. However, it suffers from several limitations.

- First, the quality of the object model depends highly on the style of writing of the analyst (e.g., consistency of terms used, verbification of nouns). Natural language is an imprecise tool, and an object model derived literally from text risks being imprecise. Developers can address this limitation by rephrasing and clarifying the requirements specification as they identify and standardize objects and terms.
- A second limitation of natural language analysis is that there are many more nouns than relevant classes. Many nouns correspond to attributes or synonyms for other nouns. Sorting through all the nouns for a large requirements specification is a time-consuming activity.

In general, Abbott's heuristics work well for generating a list of initial candidate objects from short descriptions, such as the flow of events of a scenario or a use case. The following heuristics can be used in conjunction with Abbott's heuristics:

IDENTIFYING ENTITY OBJECTS

The following heuristics can be used in conjunction with Abbott's heuristics:



Heuristics for identifying entity objects

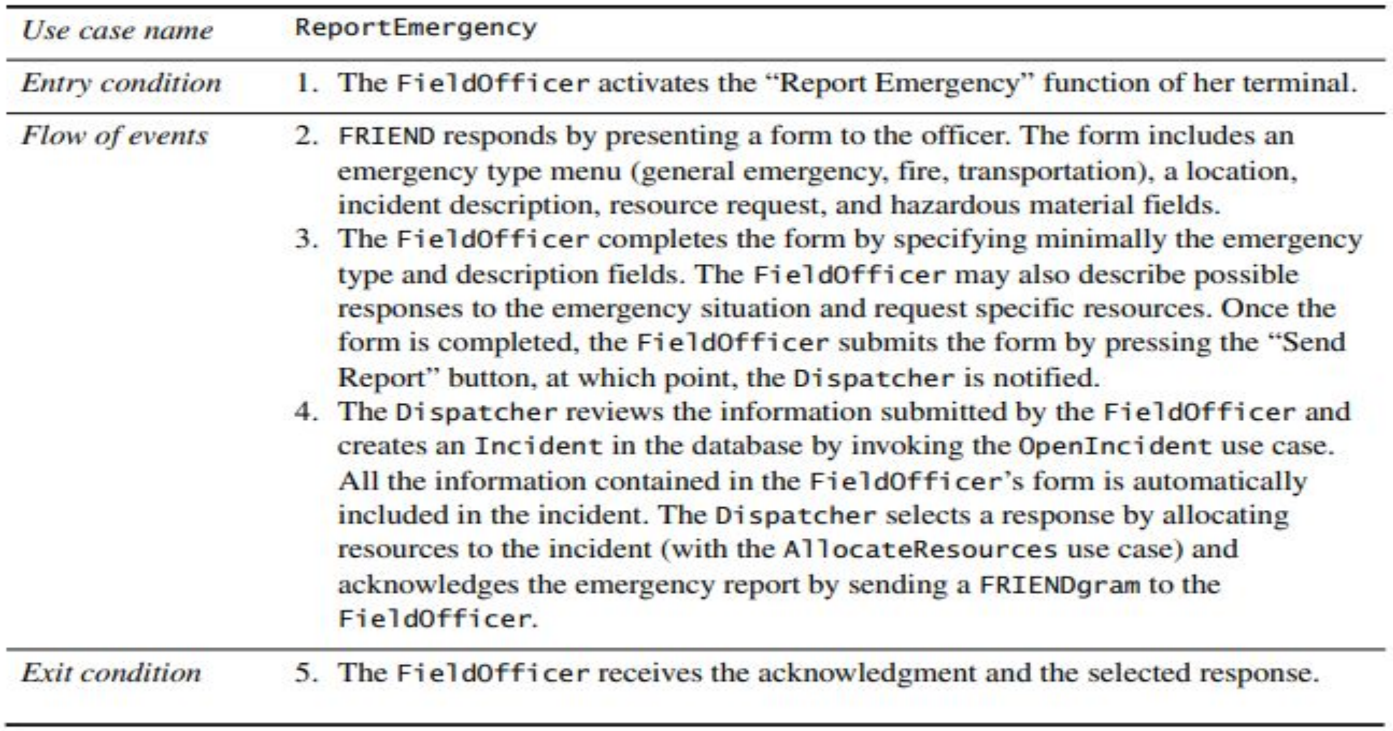
- Terms that developers or users need to clarify in order to understand the use case
- Recurring nouns in the use cases (e.g., `Incident`)
- Real-world entities that the system needs to track (e.g., `FieldOfficer`, `Dispatcher`, `Resource`)
- Real-world activities that the system needs to track (e.g., `EmergencyOperationsPlan`)
- Data sources or sinks (e.g., `Printer`).

IDENTIFYING ENTITY OBJECTS

Developers should name and briefly describe objects, their attributes, and responsibilities as they are identified. Unique naming promotes standard terminology. For entity objects, starting with end user names is recommended. Brief descriptions aid in clarifying concepts and avoiding misunderstandings. Detailed documentation is unnecessary initially, as the analysis model is still evolving. Attributes and responsibilities should be documented if not obvious, with tentative names and descriptions sufficing. Refinement can occur through iterations, with detailed descriptions necessary once the analysis model stabilizes.

For example, after a first examination of the ReportEmergency use case (Figure 5-7), we use application domain knowledge and interviews with the users to identify the objects Dispatcher, EmergencyReport, FieldOfficer, and Incident. Step 4 of the use case refers to the emergency report as the "information submitted by the FieldOfficer." After review with the client, we discover that this information is usually referred to as the "emergency report" and decide to name the corresponding object EmergencyReport.

IDENTIFYING ENTITY OBJECTS



Use case name	ReportEmergency
Entry condition	1. The FieldOfficer activates the "Report Emergency" function of her terminal.
Flow of events	2. 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. 3. 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. 4. 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	5. The FieldOfficer receives the acknowledgment and the selected response.

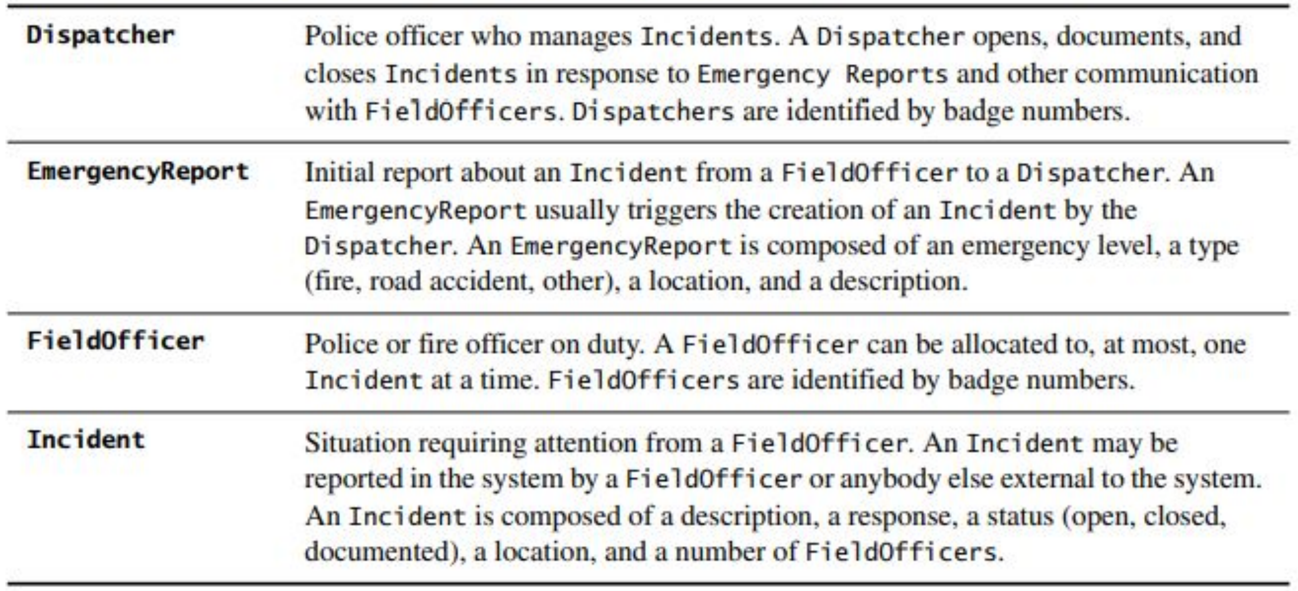
Figure 5-7 An example of use case, ReportEmergency (one-column format).

Activate

IDENTIFYING ENTITY OBJECTS

The definition of entity objects leads to the initial analysis model described in Table 5-2

Table 5-2 Entity objects for the `ReportEmergency` use case.

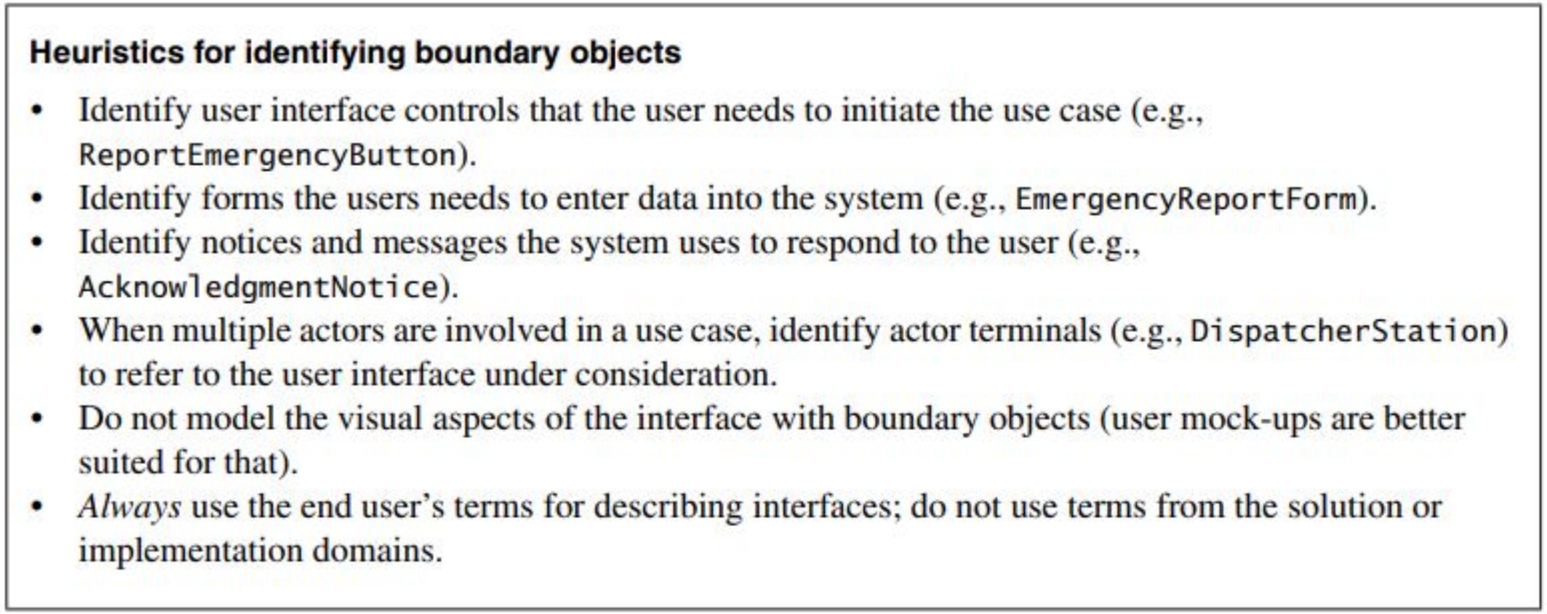


`Dispatcher`	Police officer who manages `Incidents`. A `Dispatcher` opens, documents, and closes `Incidents` in response to `Emergency Reports` and other communication with `FieldOfficers`. `Dispatchers` are identified by badge numbers.
`EmergencyReport`	Initial report about an `Incident` from a `FieldOfficer` 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`.

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.
- Boundary objects model the user interface at a coarse level. They do not describe in detail the visual aspects of the user interface.
- For example, boundary objects such as “menu item” or “scroll bar” are too detailed. First, developers can discuss user interface details more easily with sketches and mock-ups. Second, the design of the user interface continues to evolve as a consequence of usability tests, even after the functional specification of the system becomes stable.
- Updating the analysis model for every user interface change is time consuming and does not yield any substantial benefit

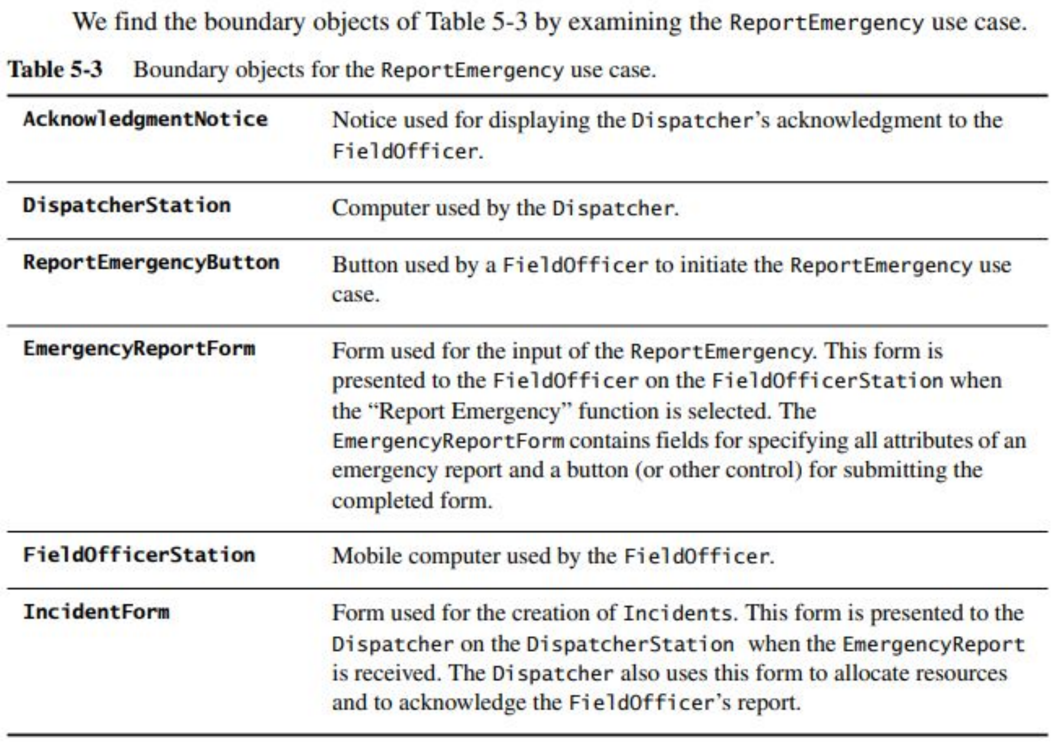
IDENTIFYING BOUNDARY OBJECTS



Heuristics for identifying boundary objects

- Identify user interface controls that the user needs to initiate the use case (e.g., `ReportEmergencyButton`).
- Identify forms the users needs to enter data into the system (e.g., `EmergencyReportForm`).
- Identify notices and messages the system uses to respond to the user (e.g., `AcknowledgmentNotice`).
- When multiple actors are involved in a use case, identify actor terminals (e.g., `DispatcherStation`) to refer to the user interface under consideration.
- Do not model the visual aspects of the interface with boundary objects (user mock-ups are better suited for that).
- *Always* use the end user's terms for describing interfaces; do not use terms from the solution or implementation domains.

IDENTIFYING BOUNDARY OBJECTS



We find the boundary objects of Table 5-3 by examining the ReportEmergency use case.

Table 5-3 Boundary objects for the ReportEmergency use case.

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.

THANK YOU