Design:
Decomposition and Decoupling

Decomposition & decoupling

- Program: built from a *collection of parts*
  - What are these parts?
  - Is it necessary to have it this way?
  - Is there *the right way* to identify these parts?
- *Decomposition*: identifying the parts that built up a program
- *Decoupling*: eliminating dependencies among these parts

Decomposition: Why?

- Alternative: make the whole program as a single block

- Advantages of decomposition:
  - Division of labour
  - Reuse
  - Modular analysis and verification
  - Localized change

Decomposition: What are the parts?

- Well… they are descriptions
  - Of some collections of data and its transformations
  - Of some algorithms
  - …
  - They will be translated into a programming language (again a description)
    - The computer follows this description, when executing the program.

So… what are the parts?

- Groups of similar or related concepts
- Together, they give structure to a program

  →The architecture

How to get these parts?

- Top-down approach
  - If a part is already available, you've got it
  - Else, split it into subparts, develop them, combine them together
- "Similar level of abstraction" approach
  - Start with some parts
  - Analyse, describe, refine each part before starting to develop them

Decoupling

- The parts need to work together
  - There are some dependencies among them
  - Some parts call functions of other parts,
    - hence, they need to know about these other parts
    - hence, they ***depend*** on these other parts
- Decoupling: minimizing these dependencies
- Dependency: introduced by the relation “uses”

“Uses”

- A uses B if the meaning of A depends on the meaning of B
- For executable code, “meaning” means “behavior”

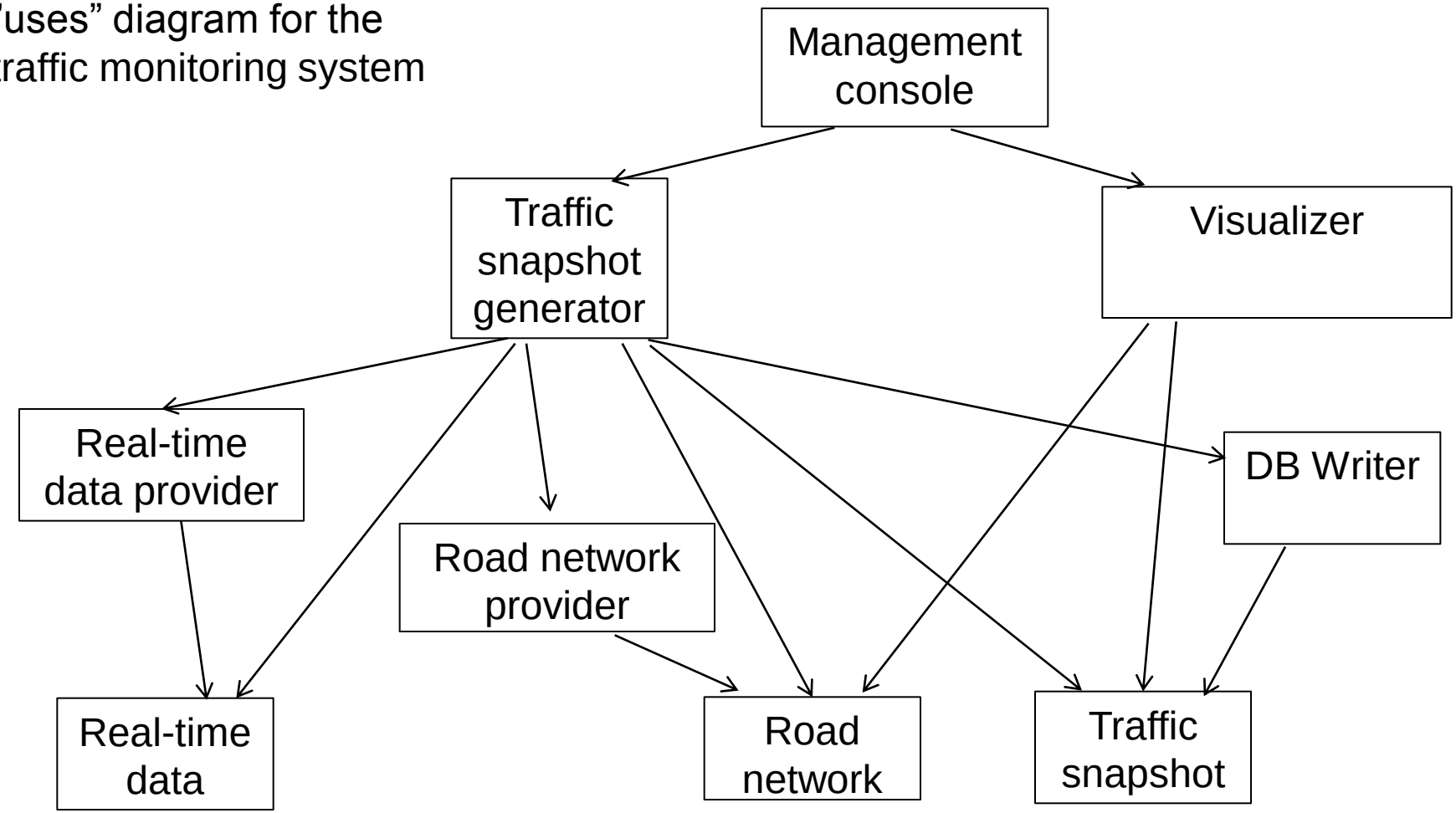
  → A uses B if the behavior of A depends on the behavior of B

Example: Traffic monitoring system

- A real-life system
  - → interprets traffic data
  - → issues regularly snapshots with the traffic situations
  - – Data - read from a database
    - Contains real-time info sent by sensors
  - – A simplified road network is used
  - – Computed traffic data - sent to some other database
    - to be used in other application
    - to be visualized in a browser
  - – Abnormal traffic situations are announced to another traffic monitoring system
  - – Statistics need to be extracted

My parts...

”uses” diagram for the traffic monitoring system



Description

- Traffic snapshot generator:
  - Maps real-time data onto road segments
  - Integrates readings from various sources
  - Issues snapshots of the traffic situation
- Real-time data
  - Collections of measurements raised in a specific time frame
  - Can come from static (vehicle counters) or moving (GPS sensors installed on vehicles) sources.
- DB Reader
  - Connects to the database with real-time data
  - Downloads readings according to some criteria (usually temporal)

Description (2)

- Road network
  - Contains information about a subset of the road infrastructure
  - A set of road segments (edges), and
  - A set of crossroads (nodes)
  - Geometry is the most important information about a road segment
- Road network provider
  - Reads/loads/downloads a road network
- DB Writer
  - Writes traffic information into some database
- Visualizer
  - Represents graphically the snapshot of the traffic situation
  - Shows a road map with the road segments colored according to the traffic on them (red for traffic jams, yellow for delays, green for normal traffic)

“Uses” diagrams

- Reasoning
  - Is part A correct?
    - Need to analyse all parts on which A depends
      - And al parts on which these parts depend, etc.
  - If I modify B, what changes?
    - All parts that use B
    - And all parts that use them, etc.

    → Impact analysis

“Uses” diagrams

- Reuse
  - Identify collections of parts so that no part in the collection uses parts outside the collection
  - E.g, Road network, Traffic snapshot, Real-time data

“Uses” diagrams

- Implementation order
  - Assign parts to different implementation teams
  - Make sure that no team has to wait for another team to finish
  - Start with parts that use no other parts

  → try to avoid cyclical “uses”

...Inconvenience

- “uses” is transitive
  - If A depends on B and B depends on C, then A depends on C
- Well then,
  - Can we define a dependency that “stops after one step”?

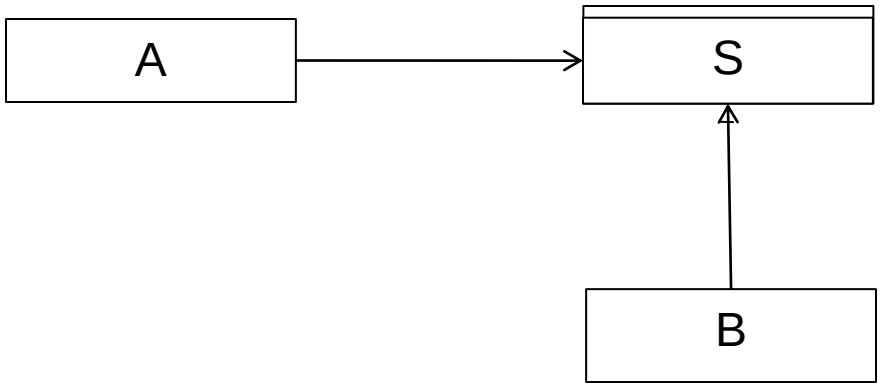
Idea

- All parts on which A depends should be complete (should not depend on any other parts)
  - Their description completely describe their behavior
- Such a description is called *specification*
- *…but* a specification cannot be executed

→ we need at least one implementation part, behaving as the specification says

Extended “uses” diagram

- New type of blocks: specifications
- New relation: “meets”, or “fulfills”
  - part B meets specification S if it behaves according to S



Advantages

- Weakened assumptions
  - if A uses B, it is improbable that all aspect of B are important to A
  - with specifications, we focus only on those aspects, S, of B on which A really depends
  - *usually specifications are much simpler than implementations*

Advantages

- Evaluating change
  - What if I change B?
    - Analyse only if the changed B *still fulfills S*
- Communication among developers
  - People developing A and B only need to agree on S.
- Multiple implementation
  - There may be sets of Bs
  - The actual B can be decided by a configuration

In my traffic system

- I can use specifications for
  - The real-time data provider
  - The real-time data
  - The road network
  - …

What are, in fact, specifications?

- They describe *contracts* that their implementations must fulfill
- The contracts describe *behaviors*
  - Services that must be offered
- Hence collections of functions

  → interfaces

Example

```
interface IRealtimeDataProvider
{
    void setConnectionInformation(string s);
    ITrafficData getRTData(DateTime tBegin, DateTime tEnd);
    ITrafficData getLastRTData(DateTime tBegin);
}
```

C# Lecture

Delegates and Events

Delegates

- from MSDN:
  - a delegate is *similar to a function pointer in C or C++*
  - *encapsulates a reference to a method*
  - a *delegate declaration defines a* [reference] ***type*** *that encapsulates a method with a particular set of arguments and return type*

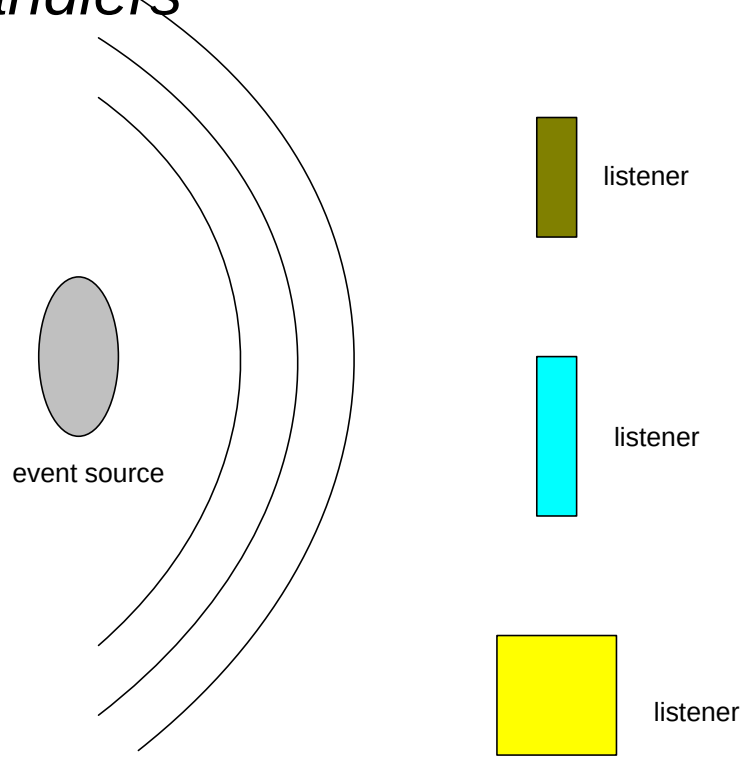
```
[<access>] delegate <return_type> <name>(<param_list>)
```

  - delegates *can be composed using the "**+**" operator*
  - an instance of a delegate is created with *new*

```
public delegate void SendString (string s);
. . .
SendString mySendStringDelegate = new SendString(DisplayMessage);
. . .
private void DisplayMessage(string s)
{ . . . }
```

Events

- Sources – event generators
- Listeners – event consumers
  - must provide *event handlers*



Events

- from MSDN:

  An event in C# is a way for a class to provide notifications to clients of that class when some interesting thing happens to an object.

- an object transmits a notification, to whatever is interested, that something has happened / changed
- *events are declared using delegates.*

Events

```
public class MyClassWithEvent {
    public event SendString NewMessage;
    public void MyFunction() {
        bool ok = true; . . .
        if (!ok) OnNewMessage(“Not OK!”);
    }
    private void OnNewMessage(string msg) {
        if (NewMessage != null)
            NewMessage(msg);
    }
}
public class AnotherClass
{
        . . .
        MyClassWithEvent myClass = new MyClassWithEvent ();
        myClass.NewMessage += new SendString(DisplayMessage);
}
```

Event mechanism

- The event consumers must:
  - ***register*** its event handling function to the event source (also called *wiring*)
- Example - adding a click handler to a button:

```
myBrowseButton.Click +=
       new System.EventHandler(browseButton_Click);

void browseButton_Click(object sender, EventArgs e)
{
   // . . .
}
```

Event handlers

- recommended:

```
public delegate void MyEventHandlingDelegate
        (object sender, TArgs e);
```

- where `TArgs` is a type derived from EventArgs.

Events for UI-components

- mouse events (Click, MouseUp, MouseDown, …)
- key events (KeyPress, KeyDown, …)
- selection events (SelectedIndexChanged, …)
- check events (CheckedChanged)
- form-specific events (Load, Resize, …)

Homework

- Reason on part “Road network” in the diagram on slide 10
  - is it self-contained, or some other parts need to be defined?
  - can you define a specification for it?
    - what would be the main services in this specification?
    - *write it as a C# interface*

      → I will ask you to write one or more method declarations on the blackboard.