Lecture 3

Software Design Space

Today's lecture

- General theory of design spaces
  - Structure of design spaces
  - Solve design problems with design spaces
- Application to software design
  - The design space of software architectural elements
  - The varieties of the elements
  - The properties of the elements

What is design space?

- The relationship between the process inputs (material attributes and process parameters) and the critical quality attributes can be described as the design space

“Input” Variables

- Input Variables:
  - The “cause”
  - Independent variable (It is a variable that stands alone and isn't changed by the other variables you are trying to measure. Independent variable causes a change in dependent variable)
  - Factor
- Output Variables
  - The “effect”
  - Dependent variable (variable that depends on other factors. Dependent variable responds to the independent variable)
  - Responses

Types of Variables

Independent

The one thing you change. Limit to only one in an experiment.

Example:
The liquid used to water each plant.

Dependent

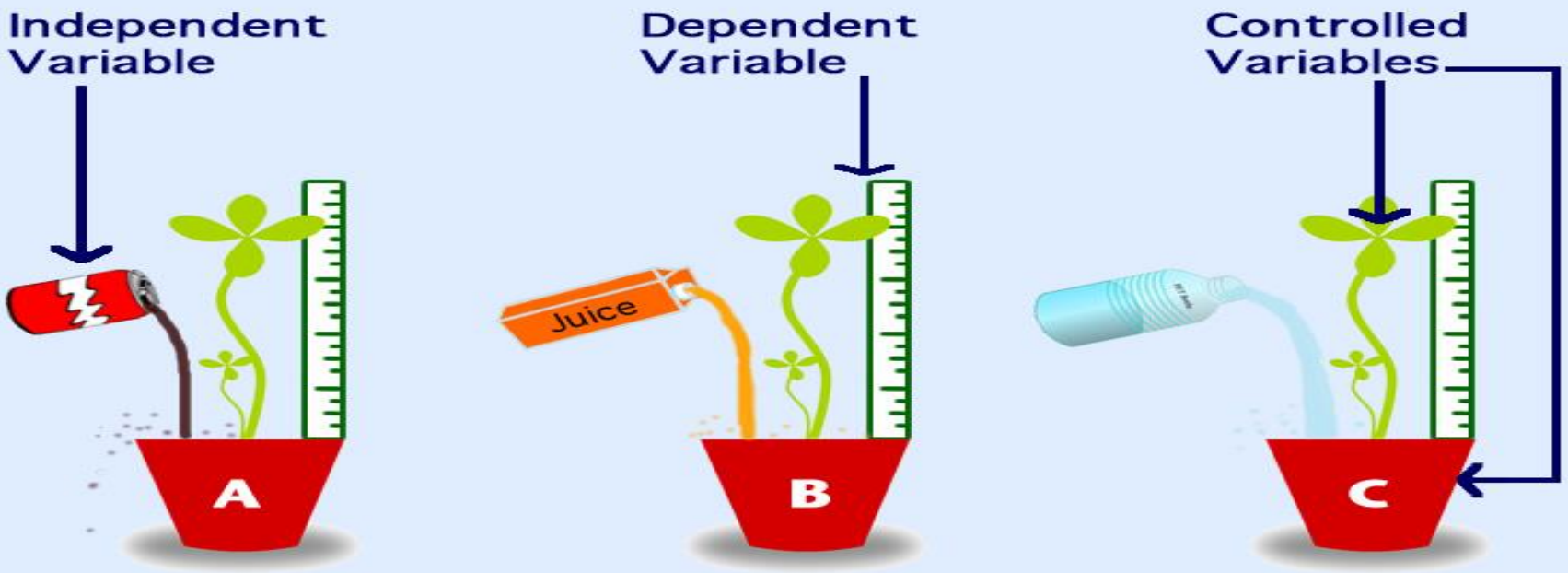
The change that happens because of the independent variable.

Example:
The height or health of the plant.

Controlled

Everything you want to remain constant and unchanging.

Example:
Type of plant used, pot size, amount of liquid, soil type, etc.



Assurance of Quality

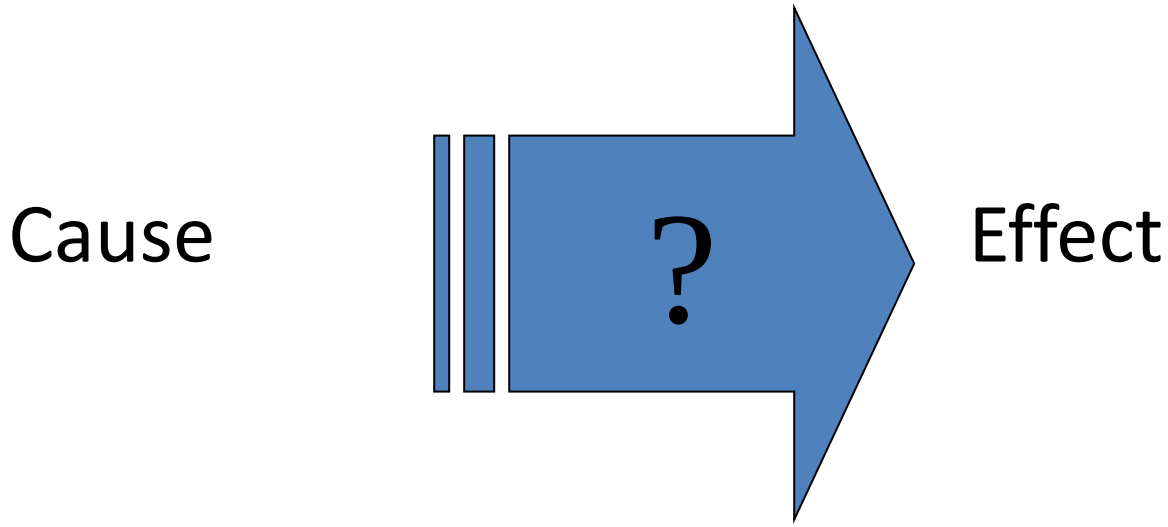
- Assurance is a high probability of meeting:
  - Safety
  - Strength
  - Quality
  - Identity
  - Purity
- For all measured quality characteristics

Why understanding design space is important?

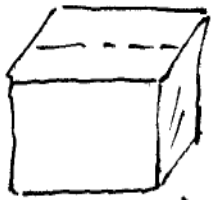
Design is to ‘find the right component of a structure’ .

(Alexander, C. *Notes on the Synthesis of Form*, Harvard University Press, Cambridge, Mass., 1964.)

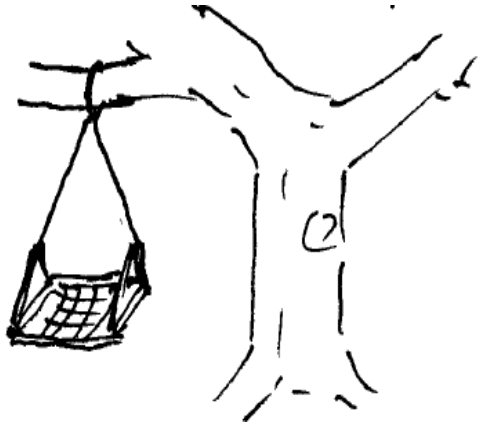
Basic Science



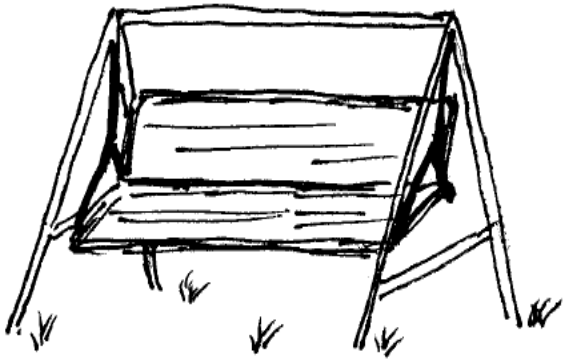
Example of Design Space: Chairs



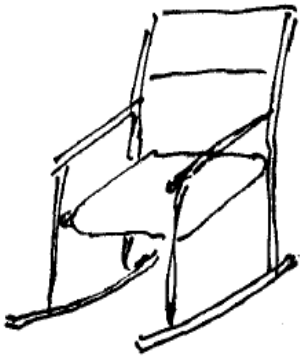
A: Box



B: Suspended Chair 1



C: Suspended Chair 2

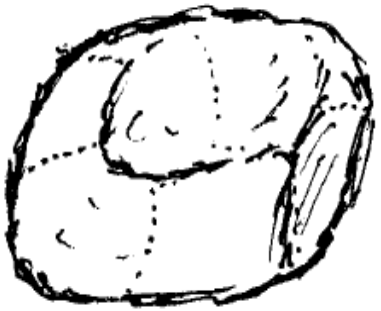


D: Rocking Chair

E: Scandinavian Chair

F: Office Chair

G: Wheel Chair



H: 'Bean-Bag' Chair

Example: Observable properties of chairs

Structure	Chair							
	A	B	C	D	E	F	G	H
Has a seat	+	+	+	+	–	+	+	+
Has a back support	–	–	+	+	+	+	+	–
Has legs	–	–	+	–	–	+	–	–
Has wheels	–	–	–	–	–	+	+	–
Has a vertical axis	–	–	–	–	–	+	–	–
Is lightweight	+	+	–	+	+	+	+	+
Has a hanger	–	+	+	–	–	–	–	–
Has a brake	–	–	–	–	–	–	+	–

Example: Functional properties of chairs

Function	Chair							
	A	**B**	**C**	**D**	**E**	**F**	**G**	**H**
Seats: prevents a downward movement of the body	+	+	+	+	+	+	+	+
Supports back: supports an upright posture	–	–	+	+	+	+	+	–
Revolves: revolves around a vertical axis	+	+	–	–	–	+	+	+
Movable: can be easily moved	+	–	–	–	–	+	+	+
Constrains back: constrains backward movement of back	–	–	–	–	+	+	+	–
Easy to manufacture: has a simple design with standard components	+	+	–	–	–	+	–	+
Aesthetic	–	–	+	+	+	–	–	–

General structure of design space

- A set of objects of the design domain
- A set of observable properties of the objects
  - More concrete features, such as the structures of the objects
  - Features that designers can choose from
- A set of functional properties of the objects
  - The features that a designer want to achieve
  - More abstract than observable properties
- Empirical design knowledge of the specific domain
  - Codified by the links between observable properties to functional properties through the objects
  - Association of objects to their observable and functional properties

Yoshikawa (1981)

Crucial factors

- *Entity set*:
  - whether a good sample of real objects was chosen determines whether the knowledge codified in the design space is complete or not.
- *Functional properties* and *observable properties*:
  - whether a good set of functional properties and observable properties were chosen determines whether properties of design concerns are included in the design space.
- *Association of properties to entities*:
  - the correctness of the domain knowledge represented in a design space is determined by the correctness of the association of properties to the objects.

Solving design problems with design space

- Types of design problems
  - *Synthesis* problem
    - When the functional properties are given, to find an artefact structure that satisfies the functionalities, i.e. to satisfy a set of functional properties
    - A set of functional properties $\Rightarrow$ a set of observable properties
  - *Analysis* problem
    - When a description of an object is given, to find out the functional properties of the object.
    - A set of observable properties $\Rightarrow$ a set of functional properties

Solving synthesis design problems: Example

- The specification of our design task:
  - to design a chair that is
    - (a) movable and
    - (b) constrains the back.
- Using the knowledge of chairs depicted in slide 6 about the functional properties of chairs, we can find that
  - the ***office chair*** (F) and the ***wheelchair*** (G) satisfy the functional requirements (a) and (b).
- The common observable structure properties of objects (F) and (G) (as in slide 5) are:
  - **Has a seat**
  - **Has a back support**
  - **Has wheels**
  - **Is lightweight**

} ***Solution to the design problem***

A slightly more complicated example

- The specification of our design task:
  - to design a chair that is
    (a) movable and
    (b) constrains the back.
    (c) aesthetic
- Using the knowledge of chairs depicted in slide 6
  - There is no chair in the set of objects that satisfies the specification
  - Nearly good candidates:
    (1) *movable* and *constrains back*: wheelchair (G) and office chair (F);
    (2) *constrains back* and *aesthetic*: Scandinavian chair (E);
    (3) *Movable* and *aesthetic*: No object.
- The common observable structure properties of objects
  - Has a back support;
  - Is lightweight.
- Need to invent new objects with new observable properties
  - Invention can be based on any nearly good candidates with new properties

Solving analysis problems: Example

- Description of design:
  - a chair that has legs and has a hanger.
- Using the empirical knowledge codified in Slide 5, we can find that
  - Suspended Chair 2 (C) is the only one that has this structure.
- Using the empirical knowledge codified in Slide 6, we can find that it has the following properties:
  - Seats,
  - Supports back,
  - Does not revolve,
  - Not movable,
  - Does not constraints back,
  - Not easy to manufacture,
  - Aesthetic.

Result of analysis of the design

Such analysis is only empirical. Further analysis of the design must be made upon the details of the design.

Software architectural elements

- Components
  - A component is a unit of software that performs some function at run-time.
  - *Examples*:
    - Procedures, Classes, Processes, Clients and servers, Databases
- Connectors
  - A connector is a mechanism that mediates communication, coordination, or cooperation among components. It enables the interactions among components.
  - *Examples*:
    - Procedure calls, method invocation, communication protocols, data streams, transaction streams.

Matching components and connectors

- The access to a component must be through a matching connector

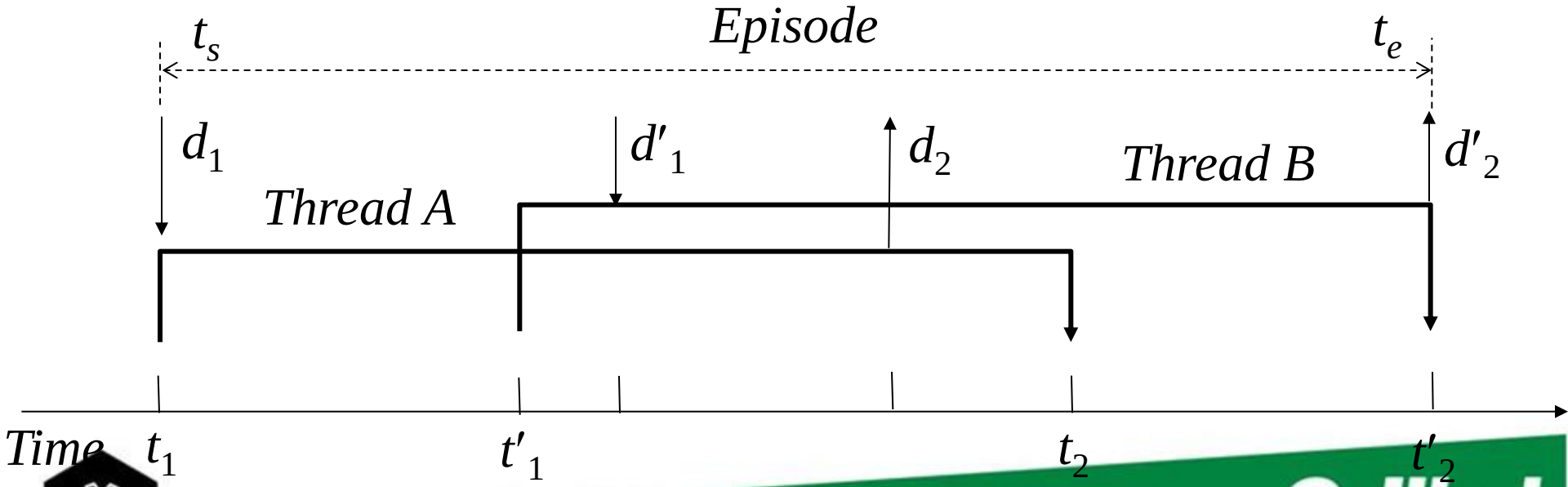
Component	Matching Connector(s)
Procedure /function	Procedure/function call
Module	Importation (access to variables, calls to procedures)
Object/class	Method call/attribute access
Thread	Send/receive messages, access shared data
Process	Send/receive messages
Distributed process	Send/receive messages,
Remote procedure	Remote procedural call
Filter	Pipe
Global data	Access data
File	Read/write/open/close operations
Database	Database queries and updates

Understanding SW architectural elements

- To understand a software architectural element, we must know both of its behavioural properties and structural properties
  - Behavioural properties (dynamic properties)
    - The properties of the software architectural element exhibited during the execution and related to the dynamic behaviour
  - Structural properties (static properties)
    - The structure of the element and irrelevant to the execution

Definition of behavioural properties

- The behaviour features of a software architectural element are concerned with the temporal relationships between the run-time events of the elements.
  - Whether or not does the acceptance of data always happen at the same time when control is accepted?
  - Whether or not a second thread of control can be accepted?



Static properties

- Static features that do not relate to the run-time events
- Determined by the syntax and semantics of code of the element
- Mostly about the structure of the element and its structural relationships to other elements; (Thus also called structural properties)
- For example:
  - Where do the data come from?
  - Where do the data go to?
  - Do the connections to other elements have directions?

Procedures/Functions

- Structure
  - Procedure declaration:

Procedure *name* ({*para-kind parameter*: *type*;}*) :
[*return_type*]

Begin

local variable declarations;

statement;

End;

  - Procedure Call:

name({*expr*});

Behavioural properties of procedure/function

- Executable
- Accept control when the procedure is called
- Transmit control when the procedure terminates
- Accept data from parameters when the procedure is called (optional)
- Transmit data when the procedure terminates (optional)
- Does not create new thread of control
- Retain state (optional, when it is static)
- Single/Multiple entry

Structural properties of procedure/function

- Single port:
  - There is only one place that the control and data are accepted;
- Directed port:
  - The port has a specific direction (i.e. control and data go into the procedure body)
- Binding at link time:
  - Which procedure is executed by a procedure call statement is determined at link time;
- Lifespan as the program:
  - A procedure exists as far as the program is executing.
- The functionality is transformation
  - The main functionality of a procedure is to transform and process data;
- The data scope is within the same virtual address space:
  - The parameters must be in the same computer.
- The control scope is within the same virtual control address space:
  - The caller must be running on the same computer.

Module

- Structure

Module interface *name;*
Uses *module_names*;
Variable declarations;
Procedure declarations; (*heads only*)
end
Module Implementations *name;*
local variable declarations;
local procedure declarations;
declaration of exported procedures;
end
Access to module's variable: *m.var*
Calls to module's procedure: *m.proc*({*expr*})

Lists the other modules that define some variables and procedures that this module will use.

Variables accessible from other modules

Procedures can be called by code in other modules

Information for the other modules to use

Variables and procedures only used in this module and not accessible from other modules

Information about how the module is implemented

Variable name

Module name

Procedure name

Behavioural properties of module

- Executable
- Accept control when a procedure is called
- Transmit control when the procedure executed terminates
- Accept data from parameters when a procedure is called (optional)
- Transmit data when the procedure executed terminates (optional)
- Does not create new thread of control
- Retain state (if it has global or local variables);
- Multiple entry (for example, when it has recursive procedures)

Structural properties of module

- Multiple port:
  - There are many places that the control and data are accepted (through different procedures);
- Directed port:
  - The ports have specific directions (i.e. control and data go into the procedure bodies)
- Binding at link time:
  - Which procedure is executed by a procedure call statement is determined at link time;
- Lifespan as the program:
  - A modules and its variables and procedures exist as long as the program is executing.
- The functionality is transformation and storage
  - The main functionality of a module is to transform and process data and also to store data in the variables;
- The data scope is within the same virtual address space:
  - The parameters must be in the same computer.
- The control scope is within the same virtual control address space:
  - The caller must be running on the same computer.

Class

- Structure
  - Declaration of class

    Class name **inherits** class_names;
      uses class_names;
      declaration of attributes;
      declaration of methods;
      declaration of constructors;
      declaration of destroyers;

    End;
  - Creation of objects:

    new *class_name*(*parameters*)

    new *object_varaible*(*parameters*)
  - Call object method:

    object.method(*parameters*)
  - Access to object's attributes:

    object.attribute

In the form of variable declarations

Visibility Var ID: Type

In the form of procedure/function declaration

A procedure called when a new object is created to initialise the object;

A procedure called when an object is released/destroyed;

A class declaration can also be divided into two parts of interface and implementation.

Behavioural properties of class

- Executable
- Accept control when a method is called
- Transmit control when the method executed terminates
- Accept data from parameters when a method is called (optional)
- Transmit data when the method executed terminates (optional)
- Does not create new thread of control
- Retain state (through attributes)
- Multiple entry (new objects can be created at runtime)

Structural properties of class

- Multiple ports:
  - There are many places that the control and data are accepted (through the methods);
- Directed port:
  - The ports have specific directions (i.e. control and data go into the object through method calls)
- Binding at run time:
  - Which procedure is executed by a method call statement is determined at run time due to inheritance and polymorphism;
- Lifespan is independent to the program but not beyond the program:
  - An object can be created and destroyed at run time
  - When the program terminates, its objects are all destroyed.
- The functionality is transformation and storage
  - The main functionality of a procedure is to transform and process data and also to store information;
- The data scope is within the same virtual address space:
  - The parameters must be in the same computer.
- The control scope is within the same virtual control address space:
  - The caller must be running on the same computer.

Thread

Data shared between the threads

- Structure
  - Declaration:

Unit *name*;
variable declarations;
thread $name_1$;
begin $statement_1$ **end**;
…
thread $name_k$;
begin $statement_k$ **end**;
thread main;
begin statement **end**;
end

  - Creation of new thread:
    fork *thread_name*;
  - Destroy thread:
    terminate;
  - Communications between threads:
    access shared variables;
    send messages: **send** (*thread-name*, *expr*)
    receive messages: **receive**(*thread_name*, expr)

In object-oriented programming languages, thread is often defined as a class with fork, terminate, send, receive, etc. as the methods, and the states of a thread as the attributes. Communication using send/receive can also be replaced by method calls.

Message contents

Example of threads in Java

```
class SimpleThread extends Thread {
    public SimpleThread(String str) {
        super(str);
    }
    public void run() {
        for (int i = 0; i < 10; i++) {
            System.out.println(i + " " + getName());
            try { sleep((int)(Math.random() * 1000));
            } catch (InterruptedException) {}
        }
        System.out.println("DONE! " + getName());
    }
}
class TwoThreadsTest {
    public static void main (String[] args) {
        new SimpleThread("Jamaica").start();
        new SimpleThread("Fiji").start();
    }
}
```

```
0 Jamaica
0 Fiji
1 Fiji
1 Jamaica
2 Jamaica
2 Fiji
3 Fiji
3 Jamaica
4 Jamaica
4 Fiji
5 Jamaica
5 Fiji
6 Fiji
6 Jamaica
7 Jamaica
7 Fiji
8 Fiji
9 Fiji
8 Jamaica
DONE! Fiji
9 Jamaica
DONE! Jamaica
```

Behavioural properties thread

- Executable;
- Accept control when a thread is created;
- Transmit control during execution, e.g. when create a new thread;
- Accept data from parameters when a thread is created and during execution;
- Transmit data during execution;
- Retain state; (through shared data)
- Multiple entry; (new threads can be created at runtime)

Structural properties of thread

- Single in port of control:
  - There is only one place that the control is accepted;
- Multiple out ports of control;
  - There may be many ports to transmit control (through fork statements);
- Multiple data ports:
  - There are many places that data are accepted (through receive statements) and transmitted (through send statements);
- Directed port:
  - The ports have specific directions (i.e. control and data go into the thread through receive statements, and go out the thread through send statement, etc.)
- Binding at run time:
  - Which thread is executed by a ford statement is determined at link time time
- Lifespan is independent to the program but not beyond the program:
  - A thread can be created and destroyed at run time
  - When the program terminates, its threads are all terminated.
- The functionality is transformation and storage
  - The main functionality of a thread is to transform and process data;
- The data scope is within the same virtual address space:
  - The shared data between threads must be in the same computer.
- The control scope is within the same virtual control address space:
  - The caller must be running on the same computer.

Process

In object-oriented programming languages, process is defined as a class in a similar way as threads.

- Structure
  - Declaration:

Unit *name*;
  process $name_1$;
    begin $statement_1$ **end**;
    …
  process $name_k$;
    begin $statement_k$ **end**;
  begin *statement* **end**;
end

  - Creation of new process:
    fork *process_name*;
  - Destroy process:
    terminate;
  - Communications between processes:
    send messages: **send** (*process-name*, *expr*)
    receive messages: ***receive***(*process_name*, *expr*)

The main differences between process and thread is that there is no shared data between processes.

Processes can be distributed to a number of computers;

A group of threads that share a common data structure must be on the same computer.

There are also other inter- process communication facilities of higher abstraction levels.

Behavioural properties of process

- Executable;
- Accept control when a process is created;
- Transmit control during execution, e.g. when create a new process;
- Accept data during execution;
- Transmit data during execution;
- Does not retain state;
- Multiple entry; (new processes can be created at runtime)

Structural properties of process

- Single in port of control:
  - There is only one place that the control is accepted;
- Multiple out ports of control;
  - There may be many ports to transmit control (through fork statements);
- Multiple data ports:
  - There are many places that data are accepted (through receive statements) and transmitted (through send statements);
- Directed port:
  - The ports have specific directions (i.e. control and data go into the process through receive statements, and go out the process through send statement, etc.)
- Binding at run time:
  - Which process is executed by a ford statement is determined at link time time
- Lifespan is independent to the program but not beyond the program:
  - A process can be created and destroyed at run time
  - When the program terminates, its processes are all terminated.
- The functionality is transformation and storage
  - The main functionality of a process is to transform and process data;
- The data scope is independent of data address space:
- The control scope can be across hardware address space (in a network).

Global shared data

- Structure
  - Declaration
    - Data type: **Type** *T*= *Type_Expr*;
    - Variable: **var** *myVar*: *TExpr*;
  - Operation:
    - Access

      -- Assignment: *myVar***:=** *expr;*

      -- Reference: *var* := *myVar* + 1;

Behavioural properties of global shared data

- Not-executable;
- Does not retains state between two executions of the program;
- Changes state during execution;
- Not accessible by multiple programs/processes simultaneously

Structural properties of global shared data

- Undirected port:
  - The data can be assigned to the variable as well as read from the variable
- Binding at link time:
  - Which variable is accessed by an ID is determined at link time time
- Lifespan is the same as the program :
  - When the program terminates, the data in the variable disappears.
- The functionality is storage
- The data scope is the virtual data address space

File

- Structure
  - Declaration
    - File type: **Type** *TFile*= **file of** *Type_Expr*;
    - File variable: **var** *myFile*: *FileType*;
  - Operation:
    - Assignment: **Assign(***fileVar*, *filename*);
    - Open/create: **Reset/rewrite/open(***fileVar*);
    - Read/write: **Read/write(***fileVa***r**, *expr*);
    - Close: **Close(***fileVar*);
    - Other operations:
      - **Seek(***fileVar*, num);
      - **Append(***fileVar*, expr);
      - **EOF(***fileVar*); etc.

Behavioural properties of file

- Not-executable;
- Retains state between two executions of the program;
- Change state during execution;
- Not accessible simultaneously by multiple programs for writing.

Structural properties of file

- Directed port:
  - For each file, where it can be either read or write depends on how it was opened.
  - The direction of the port is determined at run time.
- Binding at run time:
  - Which file is accessed by file variable is determined at run time depends on the value assign to the file variable
- Lifespan is persistent, i.e. the beyond the life of the program
  - When the program terminates, the file still exists.
- The functionality is storage
- The data scope is the virtual data address space

Database

- Structure
  - Definition of the format of data tables:
    - table_name,
    - primary_key(s),
    - {fieldName: type;}*
  - Operations:
    - Queries:
      - returns value according to the contents in the database
      - Does not change the contents
    - Update: change the contents in the database
  - Transactions:
    - A sequence of operations that must be treated as a whole

```
CREATE TABLE Students (
StdNo regRange NOT NULL,
Name VARCHAR(30),
Field progRange,
PRIMARY KEY (StdNo)
)
```

```
CREATE ASSERTION Rule1
CHECK (NOT EXISTS (SELECT
   StdNo
FROM Students
GROUP BY StdNo
HAVING SUM (creditUnits) >
   24))
```

Behavioural properties of database

- Not-executable;
- Retains state between two executions of the program;
- Change state during execution;
- Accessible simultaneously by multiple programs/processes/threads;

Structural properties of database

- Undirected port:
  - The data can be added to the database and retrieved from the database.
- Binding at link time:
  - Which database is accessed is determined at link time
- Lifespan is persistent, i.e. beyond the life of the program:
  - When the program terminates, the database still exists.
- The functionality is storage and transform
- The data scope is the beyond the virtual data address space as it can be shared through network.

Summary

- General theory of design space
  - Structure of design space
    - A set of objects, Observable and Functional properties
    - Link the properties by association of properties to the objects
  - Solving design problems with empirical knowledge of a domain codified in a design space
    - Synthesis problem: find a design that has a set of properties
    - Analysis problem; to find out the properties of a design
- The properties of the basic building blocks of software systems
  - Behavioural properties:
    - the properties about the run time behaviours;
  - Structural properties:
    - the static properties about its structure

References and further readings

[1] Zhu, H., Software Design Methodology. Chapter 8.

[2] Yoshikawa, H., General design theory and a CAD system, in Sata, T. and Warman, E. editors, *Man-Machine Communication in CAD/CAM, Proceedings of the IFIP WG5.2-5.3 Working Conference*, 1980, Tokyo, Japan, pp35–57, North-Holland, 1981.

[3] Kazman, R., Clements, P., Abowd, G. and Bass, L., Classifying architectural elements as a foundation for mechanism matching, *Proc. of COMPSAC'97*, Washington DC, August, 1997.

[4] Melta, N. R., Medvidovic, N. and Phadke, S., Towards a taxonomy of software connectors, in *Proc. of ICSE'2000*, Limerick, Ireland, 2000, pp178–187.