# Business & Computer Science

‘It is not from the benevolence of the butcher, the brewer, or the baker that we expect our dinner, but from their regard to their own interest.’

*The Wealth of Nations, Books 1-3*
Adam Smith

Dario Loi, Student, Department of Computer Science
Applied Computer Science & Artificial Intelligence
`loi.1940849@studenti.uniroma1.it`

# Contents

# 1 Course Introduction

## 1.1 Course Aims

Despite what the name might make you think, the course is *still* a practical one, it aims to introduce how computers work in a real world scenario, such as a company.

**The Purpose of a Computer** Through the lens of this course, the purpose of a software is to improve the company's efficency and throughput, everything is evaluated through a framework of costs and profits, hence, even something as simple as buying a pen must be evaluated at multiple levels, from the cost of the pen itself, to the cost of the time spent by the employee to go to the store and buy it.

## 1.2 Course Structure

The course is usually organized in two parts, alternating on a weekly basis:

| *Day* | *Purpose* |
|---|---|
| Wednsday | Lecture |
| Thursday | Seminars |

Table 1.1: Course Structure

Naturally, the seminars are *included* in the course's materials and are hence expected

The exam is also split into multiple sections:

| Part | Contents |
|---|---|
| Written | 30 to 50 closed questions |
| Oral | Optional, to raise marks |

Table 1.2: Exam Structure

The general information about the course is available at the official course page.

## 1.3 Information Systems

As studied in the first unit of *Data Management & Analysis*, an information system is a set of components that allows an organization to collect, process, store and distribute information, it is *not* necessarily something software-based, it can even be a file cabinet!

**Definition 1.3.1.** *An Information System is something that* Manages *the flow of information in an organization.*

**Information** Naturally, information is something *abstract* and *immaterial*, it is naturally difficult to manage, but it is possible to construct a set of rules, conventions and tools that allow us to represent it in a way that is *manageable*, this process is the *Flow*, and hence, as defined above, these systems are called *Information Systems*.

**Computers** When a computer is involved, the Information System is naturally labeled a Computer Information System, or *CIS* for short.

This chapter aims to give us the following capabilities

- Define what an Information System is
- Describe the history of Information Systems
- Describe the basic argument behind the article *Does IT matter?* by Nicholas Carr.

An information system is made up of *Three* main components:

- People: In charge of decision making and organization
- Technology: Hardware and Software that supports the business
- Processes: Collecting and storing information

**In General** This is a pretty broad definition, that allows a wide variety of software to be seen through the lens of an Information System, from network analyzers, to national healthcare systems, to online education platforms.

**Acronyms** Depending on the application, an Information System can be called by different acronyms, such as:

- **MRP**: *Manufacturing Resource Planning*
- **CIM**: *Computer Integrated Manufacturing*
- **SAP**: *Systems, Applications and Products*

### 1.3.1 Anthony's Triangle

Anthony's triangle is a diagram that categorizes the three purposes of an Information System:

1. Strategic (*Executive Information System*): for senior management decisions
2. Tactical (*Management Information System*): For middle management decisions
3. Operational (*Transaction Processing Systems*): For daily transactions of business

### 1.3.2 Information System Components

Several Components work together to add value to an organization, they are

1. Hardware: The physical components of the system
2. Software: The programs that run on the hardware, they can be:
    - Operating Systems: The programs that manage the hardware
    - Application Software: The programs that are used by the users
3. Data: The information that is stored in the system
4. People: The users of the systems, both producers and consumers of infomation
5. Processes: The procedures that are used to collect, process and store information[1]

This is somewhat redundant to what we stated beforehand in definition 1.3.1, but it is important to restate it to emphasize the fact that the system is composed of both *People* and *Technology.*

### 1.3.3 Processes

One of the most important components of an Information System is the *Process*, it is the goal of the Computer Information System to optimize the processes of an organization, bringing about an increase in efficency.

[1] Organizing something in *processes* that is, a series of well-defined steps, brings about a series of benefits in productivity.

**Processes Formally** Since processes are an advantageous way to organize work, it is important to spend some time to also give a formal definition of their nature.

**Definition 1.3.2.** *A process is a series of well-defined steps that are used to achieve a specific goal, it can be defined as a set of* activities

## 1.4 Does IT Matter?

Nicholas Carr, in *Harvard Business Review*, argues that Information Technology is not an *Investment*, but rather a commodity, so something that must be managed to optimize the company's profits by reducing its operational costs.

**IT as a Marketing Tool** It is also interesting to note how a company is percieved as better when it uses IT, and when this IT is of high quality, therefore IT can be seen as a *Marketing Tool* that can be used to attract customers.

## 1.5 Mainframes

A Mainframe is a class of computer that is usuall used as the heart of an Information System, where everything is *centralized*, as opposed to a distributed systems, users, which in this architecture are defined *dumb*, are not allowed to access the system directly, but rather act as consumers of its services as allowed by the system's administrators and operating system.

**Definition 1.5.1.** *A* Mainframe *is an architecture in which a central computer with very high processing power is connected to a multitude of* terminals *through a* star *topology, where the* central computer *is the* hub *of the network.*

Even though *Mainframes* are not as popular as they used to be, they are still used in many field where they are unmatched in terms of performance.

**IBM** The current IBM's Operating System is `z/OS`, which is a *Monolithic* Operating System, it is being opened to different programming languages, such as `Java`, beforehand, however, it was only available in `COBOL`.

**Batch Processing** Mainframes are based on the *Batch Processing* paradigm, where a set of jobs are submitted to be executed in a *batch*, that is, a set of jobs that are executed in a single run, this is in contrast to *Real Time Processing*, where jobs are executed as soon as they are submitted.

This, provided that the company has a sufficient amount of jobs to be executed, allows for a continuous flow of work, and hence, a higher level of productivity.

# 2 Process Modeling

## 2.1 Modeling Business

In order to properly formalize the concepts that are going to be introduced through the rest of this course, we will introduce *mathematical models* that allow us to describe the ebbs and flows of human economy.

**Processes** These models are called *Processes*, there are certain common classes of projects, such as:

1. Service
2. Support
3. Management and Control
4. Physical
5. Information
6. Business

## 2.2 Process Descriptors

These processes can then be described by some diagrams:

- Hierarchical
- State Diagrams (Automata)
- DFD – Data Flow Diagram
- WIDE – Workflow on an Intelligent and Distributed database Environment
- Action Workflow
- Petri Nets

All of these give an intuitive way to describe the processes that are going on in a company, allowing for better understanding of the company's operations.

### 2.2.1 Hierarchical Process Model

Everything in a company can be described as a set of *Hierarchies*, that is, a tree of *Processes* that are organized in a *Top-Down* fashion, where the *Top* process is known as *Macroprocess*, the hierarchy goes as follows:

1. Macroprocess – Sales
2. Process – Sales Management
3. Phase – Order Processing
4. Activity – Shipment
5. Operation – Pricing, Packaging, etc…

Naturally, these hierarchy together form a *Forest.*

### 2.2.2 Data Flow Diagrams

Data Flow diagrams are what we also call *Flowcharts*, they are a graphical representation of the flow of the data through the company‘s Information Systems, they are not used often due to them being subject to *spaghettification*, that is, the diagrams can easily become too complex to be understood.

These flowcharts are usually composed of a set of components that are used to represent the different parts of the system, they are:

- Processes – Circles
- Data Collections – Rectangles
- Interface – Bordered Rectangles
- Data Flows – Directed Arrows

**Definition 2.2.1** (Data Flow)**.** *A* Data Flow *represents any kind of flow in a system, the first component* must *be a process,the second can be either a process, a data collection or an interface, moreover, they can be either*

1. *Elementary*
2. *Structured*

**Data Dictionary** Usually, to help with readability, we provide a *Data Dictionary* and a textual description of each process, to help the user understand what the process does.

### 2.2.3 WIDE

WIDE Relies on three main components:

1. Process Model – Describes the activities in the system
2. Information Model – Describes the data being processed
3. Organization Model – Describes the structure and agents that are involved

**Anti-Spaghetti Techniques** The WIDE model introduces some more complex components with respect to the DFD model, such as forks and joins, to better describe the flow of the data through the system with fewer connections between the components.

**Remark 2.2.1.** *An analyst's role is to translate the business model in a way that is understandable in layman's terms, that is, when you are describing a process through a graph, strive to be clear, the important thing is that* People Are Going to Read It.

In conclusion, the WIDE model is just a more *expressive* version of the DFD model, which allows for more concise flowcharts.

### 2.2.4 Petri Nets

Petri Nets are a formal model that is used to describe the flow of data through an Information Systems

**Definition 2.2.2** (Petri Nets)**.** *A petri net is a 3-Tuple* $:= (P, T, A)$ *that forms a* Bipartite Graph*:*

$$G(V, E) := \begin{cases} V & := (P \cup T) \\ A & := E \subseteq (P \times T) \cup (T \times P) \end{cases}$$

*Where:*

- $P$ *is a set of places*
- $T$ *is a set of transitions*
- $A$ *is a set of arcs (Edges) that connect places and transitions in a* Directed *fashion*

**Definition 2.2.3** (Marking)**.** *A* Marking *of a petri net is a function*

$$M : P \to \mathbb{N}_0.$$

*Mapping a place to a non-negative integer, that is, the number of tokens in that place.*

*At each step of the evolution of the Petri Net, the marking function is updated to reflect the new state, for example,* $M_0$ *is the initial marking and* $M_f$ *is the final marking.*

Essentially, a Petri Net is a Finite State Automata in which a set of markings indicate the states that are 'Firable', once a transition happens, the markings are updated, and the process continues, so we are effectively running multiple Finite State Machines in parallel, each with the same topology.

**Terminology** Each transition $t \in T$ has an input set $^\circ t \subseteq P$ and an output set $t^\circ \subseteq P$ also called input and output places, the same notation also applies to places.

**Petri Net Evolution** An enabled transition can *Fire*, deleting a token in each input place and creating a token in each output place:

$$\begin{aligned} &M_0 = (2, 1, 0, 0, 1) \\ &M_0 \to M_1 \\ &M_1 = (1, 0, 1, 1, 1) \end{aligned}$$

A *Firable Sequence* is a sequence of transition $\sigma = \langle t_1, t_2, \ldots, t_n \rangle$ where

$$M_0 \to^{t_1} M_1 \to^{t_2} M_2 \ldots \to^{t_n} M_n.$$

For which we can use the closure notation $M_0 \to^{\sigma} M_n$ to denote the evolution of the markings.

**Further Definitions** Now, given a net

$$P = (P, T, A, M_0).$$

We have that:[1]

- A *Potentially Firable Transition* $t \in T$ is such that
$$\exists \tau \in T^{\star} s.t\ \tau t \text{ Is Firable}: \ {M_0}^{\tau} \to^{t}.$$
- A *Potentially Firable Sequence* $\sigma \in T^{\star}$ is such that there exists a prefix sequence $\tau \in T^{\star}$ such that $\tau\sigma$ is firable ( $M_0 \to^{\tau\sigma} M_n$ )

[1] $T^{\star}$ is the set of all sequences of transitions, so $\tau t$ is a sequence of transitions that starts with $\tau$ and ends with $t$, hence we are using notation coming from regular expressions ( Concatenation, Kleene star, etc. . . )

- A *Reachable Marking $M$* is such that
$$\exists \sigma \; s.t \; M_0 \rightarrow^{\sigma} M.$$
- $R(M_0)$ is the set of all reachable markings.
- $P_r$ is the set of reachable places s.t
$$P_r = \{p \in P | \exists M \in R(M_0), M(p) > 0\} .$$

**Petri Nets and Automata** We have that Petri nets are a generalization of automata under particular conditions:

**Definition 2.2.4** (Petri Nets and Automatas)**.** *A Finite state machine is a petri net where, for each transition $t$ both the input and the output places have cardinality* 1*:*
$$\forall t \in T, |^{\circ}t| = |t^{\circ}| = 1.$$

## 2.2.5 Workflow Nets

Workflow nets are a generalization of Petri Nets, in which we have two additional conditions:

1. A source s.t $|^{\circ}t| = 0$ for all $t \in T$[2]
2. A sink s.t $|t^{\circ}| = 0$ for all $t \in T$

Hence for each node the node is *reachable* from the source and *can reach* the sink.

**Workflow Nets as Models** Workflow nets are more representative of real world systems, since they can model *producers* and *consumers* in a system, which are widespread in real world systems.

[2] AI generated math, please check (will fix once the slides are up).

# 3 Petri Nets – Cont.

## 3.1 Introduction

Petri nets are, as aforementionedly introduced, a *Graphical* and *Mathematical* modelling tool, that is, they are formalizable through math and representable pictorially.

**Graphical Representation** The graphical representation is a bipartite graph, where we have two kind of nodes:

- Places
- Transitions

The transitions are stylized as *black rectangles*, in order to distinguish them more easily, but they are easily interchangeable with the usual labeled circles that are used to represent states in a finite state machine.

Tokens are pictorially represented with dots, however, this is not that scalable, so we can also use mathematical notations and label a node with a number $n \in \mathbb{N}$ to represent $n$ tokens in the place, be careful with this representation since it reduces the expressiveness of the model, especially towards non-mathematics oriented people.

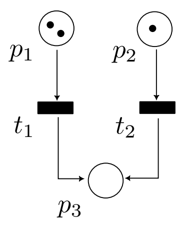

Figure 3.1: A Petri Net with 3 places and 2 transitions, we can see three tokens in $p_1$ and one token in $p_2$.

**Matemathical Representation** Since petri nets are just a graph, they can be rigorously formalized, as in definitions 2.2.2 and 2.2.3, so we omit the formalities here to avoid redundancy.

**Transition Enabling** We can further complicate a Petri Net by introducing the concept of transition *enabling*: a transition is enabled if it has enough tokens in its input places to fire, and it is disabled otherwise, naturally, the firing of the transition removes a token from each input place and adds a token to each output place.

The system steps in time *discretely*, that is each event happens simultaneously in a single step in time.

## 3.2 Petri Nets and Threads

In the previous chapter, we drew a parallel between Petri Nets and Finite State Machines, stating that their behaviour is *similar* to that of a set of FSM running in parallel, in fact, this parallel is not *entirely* made up, since we can notice that some common issues of parallelism such as conflicts are present in Petri Nets.

**Definition 3.2.1** (Petri Net Run). *A run of a petri net is a finite or infinite sequence of markings and transitions:*

$$M_0 \rightarrow^{t_0} M_1 \rightarrow^{t_1} \dots \rightarrow^{t_{n-1}} M_n \rightarrow^{t_n} \dots .$$

*such that $M_0$ is the initial marking, each transition $t_i$ is enabled in the marking $M_i$ and the marking $M_{i+1}$ is the marking that results from firing $t_i$.*

**Execution Properties** Now we can define some other concepts relative to the run of a petri net:

- Sequential Execution – Happens when transitions are on a *chain*, that is, there is a path from one transition to the other, imposing a *precedence constraint*.

- Synchronization – When multiple places are connected to a single transition, we have a *synchronization* between the places, that is, the transition will only fire when all the places have a token.
- Merging – When multiple transitions are connected to a single place, we have a *merging* of the transitions, that is, the place will only have a token when all the transitions have fired, reducing the number of total tokens.
- Forking – When a single transition is connected to multiple places, we have a *forking* of the transition, that is, the transition will fire multiple times, increasing the number of total tokens.
- Concurrency – When multiple transitions are connected to multiple places, we have *concurrency*, that is, the transitions can fire independently of each other, and the places can have tokens independently of each other.

### 3.2.1 Non-Determinism

Given the previously explored execution model of petri nets, one more thing must be said: their *evolution* (that is, the sequence of markings that they go through from their initial marking), is Non-Deterministic, hence, any enabled transition can fire at any time, and the marking can change at any time, this is a very important property of petri nets, since it allows us to model *concurrency* in a very natural way, but, as said before, it also opens the door to *conflicts*.

**Definition 3.2.2** (Conflicts)**.** *A conflict is a pair of transitions $t_i, t_j$ such that:*

1. *$t_i$ and $t_j$ are both enabled in the same marking $M$.*
2. *$t_i$ and $t_j$ are connected by a path in the graph.*
3. *The firing of $t_i$ leads to the disabling of $t_j$ and vice-versa.*

These conflicts are not too dissimilar to what was explored in the Operating Systems course with the *deadlock* concept[1], they can be resolved by introducing *priority* to the transitions, that is, we can restructure the mathematical definition of a Petri Net to allow for a hierarchy of transitions, or we can simply resolve the conflict by changing the network topology.

[1] Specifically the dining philosophers problem.

## 3.3 Reachability Analysis

A marking is said to be *reachable* from another marking if there exists a sequence of transitions $\langle t_1, t_2, \ldots, t_n \rangle$ that when fired, lead from the first marking to the second.

**Definition 3.3.1** (Reachability Set)**.** *The* reachability set *of a Petri Net is the set of all the markings that are reachable from the initial marking, one can think of this as a form of* closure*.*

Given the topology of a Petri Net (The set of its places and transitions), we can then uniquely determine the state the network is in by its markings, each of these being a unique element of the reachability set.

Since our petri nets describe a real-world system, it might be helpful to know if, from our current state, we are likely to reach a particular outcome, the process that determines this is called *reachability analysis*, in which we try to solve a membership problem given a marking and the reachability set of the Petri Net.

### 3.3.1 Reachability graphs

Reachability graphs are a way to represent the evolution of a Petri Net, they are a directed graph, where each node is a marking of the Petri Net, and each edge represents a transition that can be fired from the current marking to the next marking.

**Reachability Graph Construction** Reachability graphs are constructed by starting from the initial marking, and then, for each marking, we generate all the possible markings that can be reached from the current marking by firing a transition.

By properly identifying the frontier nodes, the generation pf the reachability graph is performable in a finite amount of steps, even if the Petri Net is unbounded, this is polynomial time, and therefore the reachability graph can be generated in a finite amount of time.

In general, there are three types of frontier nodes:

1. Terminal – no possible transition
2. Duplicate – already generated
3. *Infinitely Reproducible* – can be generated infinitely many times

**Infinite Reproducibility** A marking $M'' \geq M'$ is *Infinitely Reproducible if*:

$$M'' \geq M' \tag{3.1}$$
$$m''_i \geq m'_i \quad \forall i \in (1, 2, \ldots, n) \tag{3.2}$$

If the condition holds there exists a sequence that leads from $M'$ to $M''$, and that is surely firable infinitely many times, hence, the marking is infinitely reproducible, in that case, the Petri Net is said to be *unbounded*.

For an unbounded Petri Net, we can indicate an arbitrarily large amount of tokens through the symbol $\omega$

## 3.4 Petri Nets Extensions

Petri nets can be *extended* in order to model more complex systems, for example, we can have:

- Arc Multiplicity
- Inhibitor Arcs
- Priority Levels
- Enabling Functions (Guard Conditions)

**Remark 3.4.1.** *The last three extensions* destroy *the infinitely reproducibility property of the Petri Net.*

### 3.4.1 Arc Multiplicity

An *arc cardinality* may be associated with input and output arcs, whereby the enabling and firing rules change:

- Each input must contain at least as many tokens as the input arc's cardinality
- When the transition fires, the number of tokens removed from each input place is equal to the input arc's cardinality, as is the number of token added to each output place.

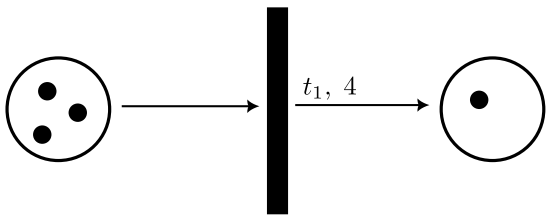

Figure 3.2: A Petri net with arc multiplicity, note that $t_1$ has *no way* to fire since it requires 4 tokens in $p_1$ but it only ever has 3.

### 3.4.2 Inhibitor Arcs

Inhibitor arcs are arcs represented with a circle head, the transition can fire only if the inhibitor place does *not* contain any tokens.

They are used to implement constraints on the firing of a transition, in the restaurant example, we could think of a transition that represents the *delivery* of a dish, and we could have an inhibitor arc from the *delivery* transition to the *payment* transition, so that the dish can only be delivered once the payment has been made.

**Inhibitor Arcs and Multithreading** Once again, we also notice a stunning similarity with multithreading, where we can think of the inhibitor as some sort of synchronization primitive, such as a *mutex*.or a *semaphore*.

**Inhibitors and Transitions** An inhibitor arc placed on a transition means that the transition can only fire if the inhibitor place does *not* contain any tokens, however, the cardinality of the output arc is not affected by the presence of the inhibitor arc, nor is the inhibitor affected by the cardinality.

### 3.4.3 Priority Levels

A Priority level can be attached to each Petri Net Transition, the standard execution rules are then modified to allow the highest priority to fire first in case of conflict, when two transitions have the same priority, that subset of transitions fires simultaneously.

### 3.4.4 Enabling Functions

An enabling function (or guard) is a boolean expression, composed with Petri Nets primitives, the enabling rules are modified so that, besides the standard conditions, the transition can only fire if the guard evaluates to *true*.

Hence we can formally define an enabling functions as:

$$\begin{aligned} f(tk) &:= (P, T, A) \cup (+, \cdot, \leq, \geq, \neq, \\ &\quad \wedge, \vee, \neg, \text{true}, \text{false}) \rightarrow \{\text{true}, \text{false}\} \end{aligned}$$

But, in practice, we often adopt a verbal description of the boolean predicate that the enabling function implements[2]

[2] Remember, *someone is gonna read this*, and it's *not* going to be a mathematician.

### 3.4.5 Colored Petri Nets

We can extend petri nets with colors, to increase expressiveness:

- We say that $C$ is a set of colors of cardinality $|C|$ and $x$ is a color of $C$.
- Place $p$ can contain tokens of any color $x \in C$
- Transition $t$ can fire tokens of any color $x \in C$

### 3.4.6 Stochastic Petri Nets

Petri nets can be extended by clearly associating *time* with the firing of transitions, resulting in *Timed Petri Nets*.

A special case of timed Petri Nets is stochastic petri nets (SPN), where the firing times are considered to be random variables.

This can be useful to model the random natures of sales, that are based on a number of difficuly-to-predict factors, such as the customer's tastes, current season, advertisements, world events, etc...