



IT2Rail



This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No: 636078



IT2Rail

Interoperability Framework

Evolution of software development technologies

Paolo Umiliacchi – CNC



This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No: 636078

Scope

- Scope of this presentation is to show a quick summary of software development technologies and techniques from the beginning of the computing age up to now.
- The objective is to clarify how the methodologies and technologies adopted within IT2Rail are the logical consequence of such evolution, so overcoming some misunderstandings and unjustified fears associated with them.
- This document complements document: ITR-T1.6-T-CNC-005-02 INTEROPERABILITY CONCEPT IN IT2RAIL, which addresses the same objective from the functional point of view.



1 - Machine language

- In the beginning it was the Binary Code.
- All digital computers work internally using binary coded data, sequences of 0's and 1's; this was applying both to code (instructions) and data to work on
- First programmers had to make software using directly the language of the computer: this was extremely difficult and time consuming, literally a crazy job
- The programmer had to work down at the level of the computer, also knowing its internal structure
- Using an Hexadecimal coding was only a very small help
- Typically, they were working on 8 bits chunks (bytes)
- There were no tools. The display was a row of lamps and the input was done using switches. Hardware could support a step-by-step execution of the program code
- The machine language was different for each processor

Example

Binary:
10100000
00000000
10110001
10111010
11110000
00011100
11001101
00000100

Hex (with memory addresses):

0398- A0 00
039A- B1 BA
039C- F0 1C
039E- CD 06 04
03A1- F0 04
03A3- C8
03A4- 4C 9A 03



2 - Assembly language

- Next step was to write software using text-based mnemonic keywords, which a programmer could give a meaning and use more easily
- The result was a new language, named Assembly
- The keywords were one-to-one matching the binary instructions of the computer and needed to be translated by a software tool named Assembler
- Programming was occurring using a keyboard and a monitor (black & white, or, more usually, black and green)
- Assemblers were offering some additional help (e.g. in memory allocation, possibility to include comments). More tools were developed, like Debuggers and Disassemblers.
- They were anyway tight to the processor and therefore different for each processor.

Example

(with line numbers and comments):

```
0451          LDY #$00
0460 LOOP    LDA (L1L),Y
0470          BEQ STOPLINE    ; ZERO = LINE FINISHED
0480          CMP BASIC+6     ; SAME AS 1ST SAMPLE CHAR?
0490          BEQ SAME        ; YES? CHECK WHOLE STRING
0500          INY              ; NO? CONTINUE SEARCH
0510          JMP LOOP
-----
```



3 - Procedural language

- A major step was to abandon the adherence to the processor internal language and define a completely artificial language with its own syntax
- A much more complex tool, a Compiler, was needed then to translate the high level programming language to the machine binary code (sometimes with an intermediate step in Assembly format)
- The generated code was often larger than if generated manually. An optimising step was introduced in Compilers.
- The programmer was using a keyboard and a monitor. A mouse and colour monitor were slowly appearing
- Many such languages were created, like COBOL, FORTRAN, C, PASCAL. In some cases (e.g. BASIC) translation was occurring during execution, line by line (so no need to recompile after each change, but lower performance)
- It was possible to use the same software on many computers, just recompiling it (so achieving portability). However each language had many “dialects” linked to the Compiler/Interpreter

Example

C language:

```
#include <stdio.h>
Int main()
{
    int l=0;
    int Result=0;

    while (Buffer[l] <> EOL)
        if (Buffer[l++] == Target)
            Result=1; //found!

    return Result;
}
```



4 – Object-Oriented Programming

- The focus moved from the language syntax to the programming style, in order to enforce a good software structure and therefore reduce the possibility of errors and increase maintainability
- Functions and procedures were replaced by Objects and Classes, which expose properties but hide the details of their internal mechanisms. They can be used selecting appropriate values for their parameters, customising their behaviour according to needs
- Programming tools are much more sophisticated and include strong programming support and powerful debugging tools
- Software runs supported by a framework which adapts to the specific Operating System and Processor
- Examples are C++, ADA, JAVA, some versions of Pascal

Example

JAVA language

```
public class Factorial
{
    public static void main(String[] args)
    {
        final int NUM_FACTS = 100;
        for(int i = 0; i < NUM_FACTS; i++)
            System.out.println( i + "! is " +
                                factorial(i));
    }

    public static int factorial(int n)
    {
        int result = 1;
        for(int i = 2; i <= n; i++)
            result *= i;
        return result;
    }
}
```

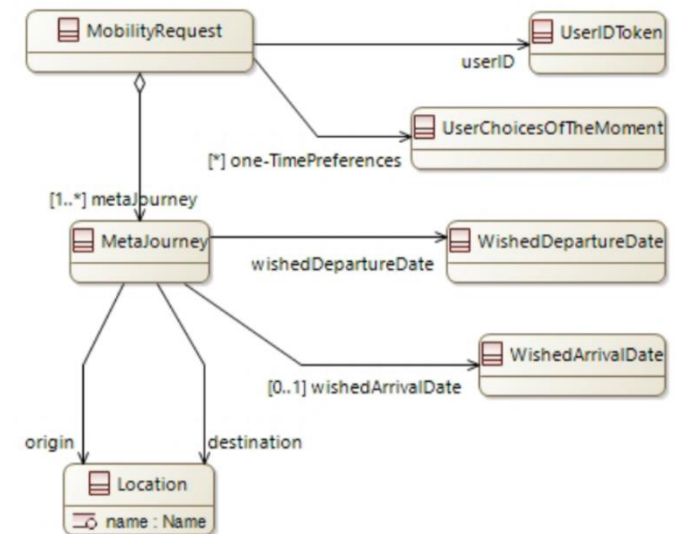


5 – Model-based Programming

- OO programming was still quite generic. Any reasonably large project required to start with defining a specific set of Objects and Classes, tailored to the needs of the specific Domain
- This creates a model of the business domain, reflected in the programming environment, which allows to quicker create the needed software, using ready-to-use bricks
- In some cases, for large projects, domain specific modelling languages could be created
- The programmer can concentrate on the business logic of the software. Programmer's interface can be textual or graphical.
- Examples are: Eclipse Modelling, UML, RMPL, etc.
- However: when the model is built using programming code, making changes in it can be very difficult and time consuming; when the model is graphical, some traditional coding is usually needed to turn it into a usable final application

Example

UML



6 – Ontology-based Programming

- Instead of building a specific software in order to create the model of a domain, it can be better to define it by means of data
- First step is to model the concepts used in a domain defining their semantic meaning and the relationships between them: this can be done using logic predicates
- The result of this process is an Ontology, which ensures an unambiguous, common understanding of the knowledge associated to the domain and can be easily modified or extended
- To complete the model, following a fully semantic approach, a number of Rules can be defined which define the business logic and can also be easily modified
- The software program becomes a generic engine which behaves according to the provided semantic data
- Supporting languages are OWL, SWRL, RDF, etc.
- Powerful tools are available (e.g. Protégé)

Example

OWL

```
<owl:Class rdf:ID="System">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty
rdf:resource="#hasSystem"/>
      <owl:someValuesFrom
rdf:resource="&core;System"/>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty
rdf:resource="#OperatedBy"/>
      <owl:allValuesFrom
rdf:resource="&core;TrainOperatingCompany"/>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty
rdf:resource="#OperatedBy"/>
      <owl:someValuesFrom
rdf:resource="&core;TrainOperatingCompany"/>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf rdf:resource="&owl;Thing"/>
</owl:Class>
```



IT2Rail



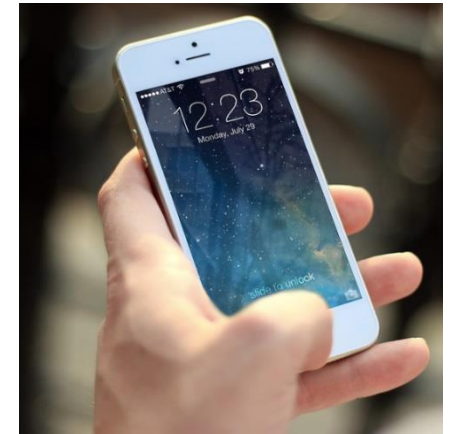
This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No: 636078

? – Natural Language Programming

- The final goal can be to have computers listen to our spoken language, understand it and self-program themselves in order to find the needed data, elaborate them as necessary and provide us with the expected answer
- Integration with the Internet of Things will allow computers to be aware of the environment where the software operates
- Programming will become more similar to teaching
- Maybe several intermediate steps will still be needed before Artificial Intelligence develops up to such level
- However, whatever the steps, they will build up on the previous ones, as happened so far
- First examples are: Cortana (Microsoft), Alexa (Amazon), Google Assistant; however they are now more user interfaces rather than programming interfaces

Example

Book me a hotel in
Milan for tomorrow
night



Conclusions

- There are no final conclusions possible
- Computers are moving from fixed equipment to portable, mobile, wearable, implantable
- Integration with IoT will create a more pervasive environment
- Advances in AI will help managing the complexity behind
- Software is more and more a critical asset
- State-of-art methodologies and technologies are required to keep the software up to increasing needs and expectations





Shopping
Real time
Innovation
Open Interfaces
Door to Door
Seamless Travel
Business Analytics

Digital

Multimodal
Ticketing
Tracking
Web of Transportation
Travel Companion
One-stop Shop

Connectivity

Technical Enabler

Cloud

Re-accommodation
Attractive Railway
Services

Interoperability

Thank you very much for your kind attention

