Meta-MES: A Factory Abstraction Architecture for I4.0

Abstract

• Smart factories can be seen as complex cyber-physical production systems (CPPSs)
  • their design, implementation, management and evaluation needs some abstraction strategies to focus on the relevant aspects instead of on the low-level details
• This tutorial proposes an abstraction methodology, and related tools:
  — It starts from the way to build a complete model of the CPPS, based on SysML
  — there is the description of protocols (like OPC-UA) to see the CPPS as a service-oriented architecture,
  — where IIoT data are collected by an ad-hoc architecture,
  — the model allows the automatic configuration of this data collection architecture with the generation also of the digital twin of the production line,
  — finally, the integration of different Manufacturing Execution Systems (MESs) produces an integrated view through the so-called Meta-MES
• The theoretical part of the tutorial is completed:
  — by the visit of an actual production line

What is a CPPS?

• Cyber-physical production systems (CPPS) consist of autonomous and co-operative elements and subsystems that are connected based on the context within and across all levels of production, from processes through machines up to production and logistics networks

Main CPPS characteristics

• Intelligence (smartness): the elements are able to acquire information from their surroundings and act autonomously and in a goal-directed manner
• Connectedness: the ability to set up and use connections to the other elements of the system, including human beings, for co-operation and collaboration, and to the knowledge and services available on the Internet
• Responsiveness: towards internal and external changes
CPPS and smart factory

- Smart factories can be seen as complex cyber-physical production systems (CPPSs).
- Their design, implementation, management and evaluation needs some abstraction strategies to focus on the relevant aspects instead of on the low-level details.

ICE Laboratory

Sensor: IIoT vs IoT

- **Sensor**: a device, module, machine, or subsystem
  - Detect events or changes in its environment
  - Send the information to other electronics
- **IoT** (Internet of Things):
  - A system of interconnected computing devices, mechanical and digital machines, objects that are provided with unique identifiers and the ability to transfer data over a network without requiring human-to-human or human-to-computer interaction
  - Embedded systems such as sensors and actuators
- **IIoT** (Industrial Internet of Things):
  - IoT + Industry 4.0

Industry 4.0 and the communication

- Standard automation pyramid
  - from actuators and sensors to enterprize systems
  - result of the Third Industrial revolution
- At the lowest level, different vendors provide different fieldbus protocols
  - Fieldbuses like Profibus DP, Modbus-RTU
  - Industrial Ethernet networks like PROFIBUS, EtherCAT, Modbus TCP, EtherCAT/DP
- **Challenge**: communication technology for all the levels of the pyramid.

Solution: OPC Unified Architecture

- OPC UA offers two communication patterns:
  - Client/Server
  - Publish/Subscribe
- Designed for secure, reliable and interoperable M2M communication
  - Service-oriented architecture (SOA) follows the request/response paradigm.
Client/Server VS PubSub

- Different aspects are considered when choosing the best model:
  - Real-time requirements
  - Information fan-out
  - Computational resources
  - Cloud integration

Pub/Sub Cloud OPC-UA

Analysis confirmed PubSub’s applicability in:
- Cloud scenarios
- Real-time scenarios

IoT vs Industrial IoT - I

1. Communications:
   - IoT devices use standard IT protocols like Wi-Fi, Ethernet, ...
   - IIoT devices support industrial protocols like Profinet, Ethercat, ...

2. Safety and Security:
   - IoT devices access the Internet directly, so they must be secured
   - IIoT devices live in the OT network, so they must take care of safety rather than security

3. Reliability:
   - It is not so critical for IoT devices
   - IIoT devices must guarantee high reliability to keep safe the production plant

AN EXEMPLIFICATION CASE: ICE LABORATORY

IoT vs Industrial IoT - II

4. Architecture:
   - IoT devices communicate with a public cloud to access user information
   - IIoT devices use a private or hybrid cloud which contains company data

5. End devices:
   - IoT devices sense data (like steps, blood pressure, ...) for a consumer-type ecosystem
   - IIoT devices provide precise measures to both cloud and PLCs

6. Costs:
   - IoT devices must be as cheap as possible
   - IIoT devices need to be certified so their cost can be slightly higher

Connect devices to the network

PROBLEM:
- Absence of a unique standard for communication between sensors and central processing
- The sensors cannot be connected directly to the IP/internet network

SOLUTION: Gateway
- Support different protocols
- Connect devices to the internet
- Provides an additional level of security
- Pre-process data
- Reduce transmission size
Connect devices to the network

The choice of protocols depends on:
- Wired or wireless
- Sensor-gateway distance
- Bandwidth
- Latency
- Energy consumption

ICE Lab: Data Integration Hub

The choice of protocols depends on:
- Wired or wireless
- Sensor-gateway distance
- Bandwidth
- Latency
- Energy consumption

ICE lab: IoT Data Viewer

ICE lab: IoT and IIoT Data Viewer

ICE lab: IoT and IIoT Data Viewer

SysML

A Unifying Systems Language

A Language to document the properties from different disciplines to describe the whole solution
Systems Modeling Language (SysML): origin

- Extension (dialect) of UML
- Supported system's viewpoints:
  - Structure
  - Behavior
  - Parameters
  - Requirements
- Elementary modeling entity: diagrams

**Activity Diagram (ACT)**

- Activity diagram shows system dynamic behavior using a combined Control Flow and Object (data) Flow model
  - Control Flow: flow of functional behaviors
  - Object Flow: data flow of object inputs/outputs
- Composed of Activity Nodes
- ACT Purpose:
  - Specify dynamic system behaviors that satisfy system Functional Requirements
  - ACT diagrams are simulatable

**State Machine Diagram (SMD)**

- State Machine diagram shows the sequences of States that an object or an interaction go through during its lifetime
  - In response to Events (Triggers)
  - May result in side-effects (Actions)
- Composed of States
- SMD Purpose:
  - Specify dynamic system behaviors for time-critical, mission-critical, safety-critical, or financially-critical objects
  - SMDs are simulatable

**Sequence Diagram (SD)**

- A Sequence diagram is a dynamic behavioral diagram, showing:
  - Communication Collaboration among distributed objects or processes using:
  - Messengers: send messages along corresponding behavior
  - Supports sequence sending
- A message represents the communication between objects
- Purpose:
  - Specify dynamic system behavior in specifying collaborations using an abstracted model (Part)
  - SDS are simulatable

**Use Case Diagram (UC)**

- A Use Case diagram shows:
  - Communications among system transactions (Use Cases) and external actors (Actors)
  - Actors may represent persons, organizations, software systems, or hardware systems
  - Defines the system scope
- A Use Case represents a system transaction with an external system user [Actor]
  - Sometimes considered as high-level functional requirements
- Purpose:
  - To provide a high-level view of the subject system
  - Convey the top-level system requirements in non-technical terms for all stakeholders
Analysis

June 16 2022

Modeling CPPS with SysML

June 16 2022

Methodology Overview

June 16 2022

Taxonomy DIN – Business Process Model and Notation

June 16 2022

Automation Markup Language

June 16 2022

ISA-95

June 16 2022
Case-study: UNIVR ICE Laboratory

- To exemplify the system modeling methodology, we adopt the transportation system of the ICE Laboratory as a case study.
- We concentrate on a subset of components.

```plaintext
Transportation System Diagram
```

Exploiting Models for CPPSs Design

- Models produced using the proposed methodology may be used for:
  - Verification and Validation of SysML models based on formal methods
  - System Analysis and Optimization through design-space exploration
    - Intrinsic of any PBD flow
    - Optimization problems and formal models can be built on top of SysML models
  - Code Generation:
    - Hardware-software integration
    - Control software implementation
      - E.g., PLC software consistent with the IEC 61131-3 standard
    - Simulation

Generation of Simulation Scenarios

- Generate the code for simulation
  - From SysML diagrams
- Code import in a plant simulation tool
  - E.g., Tecnomatix Plant Simulation
- A two-phased task:
  - Structural Information Extraction: extraction of plant topology from SysML structural diagrams
  - Behavioral Information Extraction: translation of behavioral diagrams to C code implementaions

Containerization and Kubernetes

- Cloud native reference architecture
- Traditional architecture

```plaintext
Cloud Native Architecture
```

```plaintext
Traditional Architecture
```

```plaintext
Simulation Diagram
```
The container model

What is Kubernetes

- Project that was spun out of Google as an open source container orchestration platform.
- Built from the lessons learned in the experiences of developing and running Google’s Borg and Omega.
- Designed from the ground-up as a loosely coupled collection of components centered around deploying, maintaining and scaling workloads.
- Known as the Linux kernel of distributed systems.
- Abstracts away the underlying hardware of the nodes and provides a uniform interface for workloads to be both deployed and consume the shared pool of resources.
- Works as an engine for resolving state by converging actual and the desired state of the system.

Decouples Infrastructure, Scaling and Self Healing

- All services within Kubernetes are natively Load Balanced.
- Can scale up and down dynamically.
- Used both to enable self-healing and seamless upgrading or rollback of applications.
- Kubernetes will ALWAYS try and steer the cluster to its desired state.
Key concepts: PODs

- Atomic unit or smallest "unit of work" of Kubernetes.
- Pods are one or MORE containers that share volumes, a network namespace, and are a part of a single context.
- They are also Ephemeral

Key concepts: Services

- Unified method of accessing the exposed workloads of Pods.
- Durable resource
  - static cluster IP
  - static namespaced DNS name
- They are not Ephemeral

Architecture Overview

Data Integration Hub

Introduction - I

- Prerequisite: a plant equipped with OPC-UA servers providing
  - equipment's status
  - by native or custom OPC-UA servers
  - environment data (IoT and Industrial IoT)
- Goal: monitor, log, analyze and control the plant status
- Our solution:
  - OPC-UA Data Integration Hub that relies on Kubernetes
Introduction - II

- Architecture goals:
  - OPC-UA servers monitoring
  - multi-protocol support for data ingestion
  - data logging
  - data analytics and filtering
  - data upload to different cloud providers (hybrid cloud approach)
  - provide a unique interface to access data
  - recipe execution and scheduling on supported equipments

Frontend – OPC-UA Client

- Frontend Node
- Reconfigurable:
  - (OPC-UA server url)
  - variable list to subscribe, for each:
    - sampling interval
    - data format and unit
    - custom tags
  - topic name to publish data to
- At least one instance per machine

Frontend – MQTT Broker

- Vernemq MQTT Broker
  - native integration with kubernetes
  - multiple instances can be spawned
  - actually 2 in cluster
- MQTT2Kafka Agent (python)
  - transfers messages to Kafka
  - topic translation mechanism
  - MQTT topic/data/list
  - Kafka topic

- No performance data available yet
Data Access Frontend – Data Monitor

- Subscribed to all Kafka topics
  - reads each message and stores the last value of each variable
- Allows to query the latest values for each variable through HTTP GET
- Endpoints
  - /data -> list measurements
  - /data/measurement -> list fields
  - /data/measurement/field -> get value(s)

Data Access Frontend – Grafana

- Grafana is an analysis and monitoring solution for every database
- We created different dashboards to show stored data from InfluxDB

Broker – Kafka

- A streaming platform that supports multiple producers and consumers
  - topic concept
  - consumer group concept
- Possibility to store temporarily all topic messages (1 week)
- Actually deployed with 3 instances
- Used to move data from ICE lab to consumers (db, data monitor, ...)

Broker - RabbitMQ

- Message Oriented Middleware adopted to handle the platform commands
  - deveoped with three replicas
- Messages are moved through queues
- One queue per RPC service (e.g. OPC-UA clients)
  - conveyor_rpc_queue
  - cell4_rpc_queue
  - cell5_rpc_queue
  - conveyor_rpc_queue
- Three replicas

Backend – InfluxDB Agent

- Python application
  - subscribes to provided Kafka topics
    - possibility to specify group-id
  - rewrites incoming data to DB Abstraction Layer
    - possibility to send chunks of data to reduce write frequency
- Multiple instances to cover all kafka topics
- Two writing modes
  - synchronous: sends immediately each value
  - asynchronous: acts as buffer sending chunks

Frontend – Backend - Telegraf

- Go application born to collect, aggregate, process and write metrics
- It can read data from Kafka topics and write it to the DB Layer (image 1)
- Can accept data formatted according to syslog protocol and write it to a Kafka topic (image 2)
- ...
Backend – DB Abstraction Layer

- Application able to partition data to multiple databases
  - configurable
  - supports NoSQL (InfluxDB) and SQL (Postgres) dbs
  - exposes API to write data and perform queries
- (this node has not been yet implemented!)

Backend – InfluxDB2

- NoSQL Database optimized to ingest time series
- Supports multiple buckets to store data
- Very powerful query language: flux
- Integrated web interface to perform queries

Backend - ICETom

- Java application that is bundled with Mindsphere APIs
  - the dataselector part select chunks of data from InfluxDB
  - selected data is aggregated as configured
  - data is pushed to Mindsphere using its API

Data Integration Hub generation

Service-oriented architecture
What is SOA?

• Service-Oriented Architecture (SOA) defines a way to make software components reusable and interoperable through shared services:
  - Each component implements a simple task such as retrieving information or performing an action
  - Services are accessible through an Enterprise Service Bus (ESB)

• Exposed by software components:
  - Services are accessible through an Enterprise Service Bus (ESB)

• Loose coupling
• Location transparency

Service-oriented Manufacturing

• Service-oriented manufacturing applies the concept of SOA to smart manufacturing:
  - Manufacturing resources connected with SOA principles
  - Each machine exposes its interface as services
  - Ease the application of modern production paradigms
  - Changing application manufacturing

Service-oriented Middleware (SOM)

• Service-oriented middleware (SOM) to ease the application of modern production paradigms:
  - Adapt the previous software extending its capabilities
  - Easy integration of new technologies
  - Hide the complexity necessary to support new technologies
  - Work orders as a sequence of services
  - Service orchestration performed by another component (e.g. Meta-MES)
  - OPC UA Client offers machine capabilities as services

Digital Twin
Industry 4.0 and Digital Twins

- Digital twin as a key part of Industry 4.0
  - Multiple industrial applications across all phases of production: manufacturing and operations, maintenance and service

US DIGITAL TWIN MARKET SIZE 2014-2025

- Definition by NASA
  “Integrated, multi-physics, multi-scale, probabilistic simulation of a vehicle or system that uses the best physical models, sensor updates, fleet history, etc. to mirror the life of its flying twin”
  - First definition
    Given by Michael Grieves, University of Michigan
    Presentation to industry for the formation of a Product Lifecycle Management (PLM) center
  - Definition extension
    Classify digital twins into prototypes, instance and aggregate, still early theoretical

First definition
Given by Michael Grieves, University of Michigan
Presentation to industry for the formation of a Product Lifecycle Management (PLM) center

Definition by NASA
“Integrated, multi-physics, multi-scale, probabilistic simulation of a vehicle or system that uses the best physical models, sensor updates, fleet history, etc. to mirror the life of its flying twin”
  - Concept first applied in the 1970s during the Apollo 13 program
  - Rapidly account for changes to the vehicle while exposed to the extreme conditions in space
Digital twin as key technology for future vehicles by NASA and US Air Force for structural health management.

Multidisciplinary physics-based methodology.

First conference. Exploration of key technologies, mechanisms, implementation methods Definition of digital twin-driven design.

So what is a Digital Twin?

- Virtual representation that interacts with the physical object throughout its lifecycle to provide intelligence for prediction, evaluation, optimization, etc.

- 5 ingredients:
  - Physical space plus virtual space
  - Their connection for virtual-physical interaction
  - Data from virtual and physical domains used for comprehensive information capture
  - Functions for unified management and on-demand usage:
    - Detection, judgment, prediction
    - All parts interact and collaborate with each other to tackle complex problems

Parts of a Digital Twin

- **MODEL**: Digital companion is made of a set of models. Reproduce with high fidelity the properties, behaviors, and rules of the physical object. Operate autonomously in the virtual space.
- **DATA**: Ability to predict problems on physical side validate performance before systems deployed.
- **CONNECTIONS**: Enable seamless interactions between digital and physical elements.
- **SERVICE**: Encapsulate functions of the digital twin into services of easy and convenient usage.
Digital twin

- A digital twin is a digital replica of a living or non-living physical entity.
- A digital replica can represent physical assets, processes, people, places, systems and devices.
- Can be used for various purposes.
- Types of Digital Twin:
  1. Autonomous
  2. Connected (digital shadow)
  3. Hybrid

Autonomous

- Independence from the real line
- The line can be:
  - Already existing
  - A new model
- ADVANTAGES:
  - Errors or bottlenecks detection before production validation
  - Number reduction of physical prototypes with virtual validation
  - Cycle time optimization through simulation
- SETTINGS:
  - Speed
  - Power

Connected (digital shadow)

- Digital twin communicates with the real plant with sensors.
- The line must exist.
- ADVANTAGES:
  - Production line monitoring even when far from the physical line
  - Failures or bottlenecks monitoring
  - Production process optimization thanks to the production data collection
  - Creation of statistics of intermediate and finished products produced by the line.
- SETTINGS:
  - The digital twin utilizes the OPC UA protocol for the communication with machines.

Hybrid

- Digital twin where some machines are real others are hypothetical.
- The line exists in the middle.
- ADVANTAGES:
  - Statistics of a hypothetical production
  - Optimization of production cycles.
- SETTINGS:
  - The digital twin communicates with the existing machines with the OPC UA protocol.
  - Time of hypothetical machines.
  - Power of hypothetical machines.

OPC Unified Architecture (OPC UA) is a machine to machine communication protocol for industrial automation.
MES

Functional layer between enterprise resource planning (ERP) and process control systems

- Dispatch work instruction from the ERP
- Ensure resource availability
- Meet production requirements
- Reduce human errors
- Reduce human errors

Essential in today’s competitive and rapidly changing manufacturing environment

- Product Tracking and Genealogy:
  - Track all the items in the production process
  - Warehouse management
  - Factory floor

- Process Management:
  - Managing the production process from order release to finished goods
  - Work orders
  - Inventory management
  - Resource allocation

- Resource Allocation & Status:
  - Manage the allocation and status of resources
  - Warehouse
  - Inventory

- Performance Analysis:
  - Analyze production data
  - Optimize production processes
MES – Functionalities (3)

- Data Collection and Storage:
  - Facilitate the acquisition of data
  - Equipment
  - Personnel management
  - Quality control systems
  - Collects data from different task areas
  - Consolidation of data
  - Data correlation
  - Data visualization

- Quality Management:
  - Quality monitoring
  - Preventive actions
  - Verification
  - Nonconformance workflows

- Dispatching
- Production Units
- Product Tracking & Genealogy
- Process Management
- Reduce Allocation & Status
- Data Collection & Storage
- Quality Management
- Labor Management
- Performance Analysis

MES – Functionalities (4)

- Labor Management:
  - Manage the employees
  - Work time logging
  - Personnel qualifications
  - Personnel certifications

- Performance Analysis:
  - Define and track key performance indicators (KPIs)
  - Advanced analytics
  - Data visualization
  - Performance monitoring
  - Reporting

MES - Limits

- MES are rigid systems with narrow, clearly defined features and system architectures
  - Hard to customize
  - Time-consuming
  - Expensive licenses

- Manufacturing workflow needs to be adapted to fit the MES
  - Easier to change the manufacturing operation rather than the MES

- MES have a blind spot for human data
  - Yearly human interaction
  - Production processes are manually

Cloud-native architecture

The current market is characterized by rapid changes in consumer demands. To respond to such issues, we need manufacturing:

- Easily reconfigurable
- Supports the integration of new technologies transparently
- Based on independent microservices
Cloud-native architecture

This architecture has many advantages:
- Seamless integration with existing cloud architecture
- Automated reconfiguration
- Easy integration of new technologies
- Autonomous execution and supervision of production orders
- Runtime dynamic scheduling
- Support for advanced data analysis
- Support for real-time decision-making

MES and Meta-MES Architecture

Recipes

- Machine Function
  - Each machine behavior can be described as a sequence of basic actions and basic services exposed
  - Avoid modeling the same sequence of actions in different Tasks multiple times
  - Increase the flexibility of the production line
  - They have input and output parameters
  - Each function has a private scope
  - Inside SysML, they are represented with a direct graph

- Tasks
  - Tasks are represented as a directed graph
  - Nodes represent actions that are executed when a visit takes place
  - An action can be a Machine Function, OPC UA action (read/write, method, subscription), or a control action (var, if, add, end)
  - Each action has input and output parameters
  - Inside SysML, tasks are represented with a direct graph
Recipes

Recipes are represented as a direct acyclic graph:
- Nodes represent tasks that can be executed on a working cell.
- Edges between nodes describe the relation of type “start after”.
- A directed graph inside a node can be executed.
- Can also represent the intermediate state of the production process and parameters for each machine (energy consumption, precision, etc.).

Dynamic Scheduling

Takes as input the optimal static schedule and unexpected events (new work orders, delays, machine breakdown, etc.). Schedule tasks with three different heuristics:
- Dynamic Assignment
- Production Extension
- Schedule adaptation

Dynamic Assignment

Assigns the runtime task in the free space between operations

Schedule Adaptation

Swap the order of two tasks if this reduce the total execution time

Meta-MES Architecture

Hybrid Scheduling

Scheduling with two phases:
- Static scheduling based on an optimization algorithm that produces an optimal solution used as a basis.
- Dynamic scheduling adapts the produced schedule to unexpected events every seconds.