DINASORE: A Dynamic Intelligent Reconfiguration Tool for Cyber-Physical Production Systems

Eliseu Pereira, João Reis and Gil Gonçalves
SYSTEC, University of Porto

Eclipse SAM IoT 2020
Security | AI | Modelling

Supported by: BRAIN-IoT

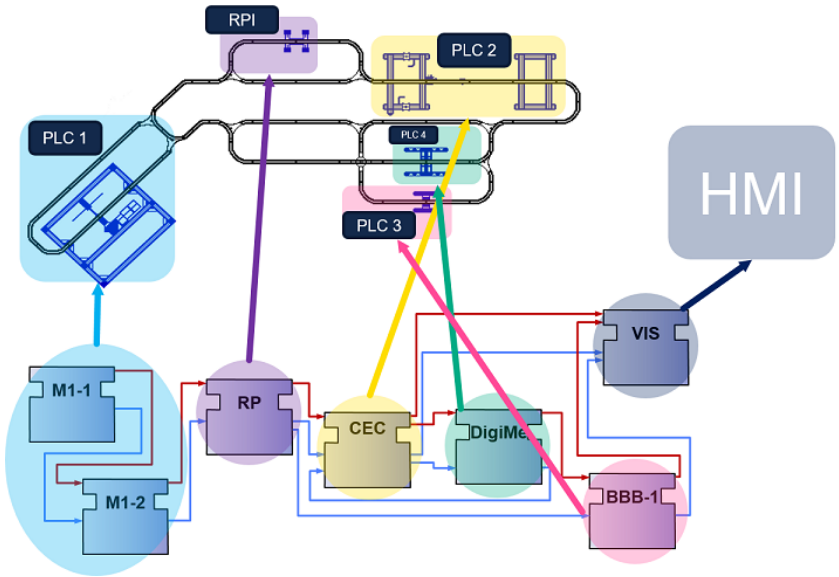
Agenda

- **Introduction**
  - Industry 4.0
  - IEC 61499
- **Implementation**
  - System Architecture
  - DINASORE
  - OPC-UA
  - Development Process
- **Test Case Scenarios**
  - Collision Detection in Servo Motors
  - UR5 & Gripper Control
  - Manufacturing Applications
  - Performance Evaluation
- **Conclusions & Future Work**
  - Contributions
  - Future Work

Introduction

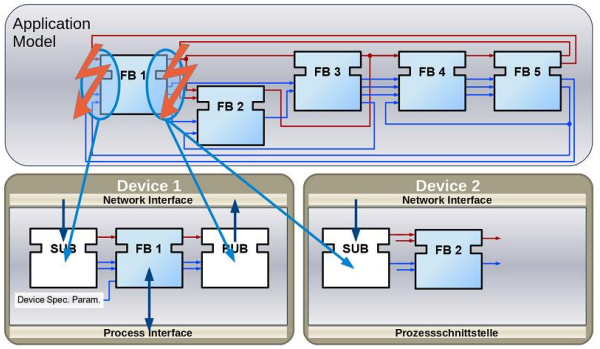
Industry 4.0

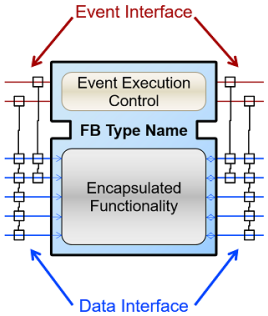
- Digitalization of industrial equipment (sensors, machines)
- **Reconfiguration** of **large** Cyber-Physical Production Systems, enabling:
  - Quick modification of requirements (products).
- Existent reconfiguration **tools** or **programming languages**:
  - **IEC 61499**: industrial standard to design distributed CPPS;
  - NodeRED, Eclipse Kura.



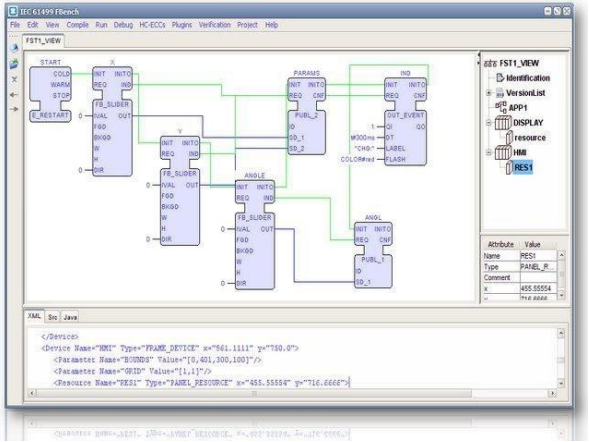
IEC 61499

- Industrial Standard;
- Distributed Orchestration;
- Encapsulation of software in Function Blocks (FBs);
- **Development Environment (IDE)**:
  - Orchestrate, Map, and Deploy.
- **Runtime Environment (RTE)**:
  - Execute FBs according to its Execution Model.

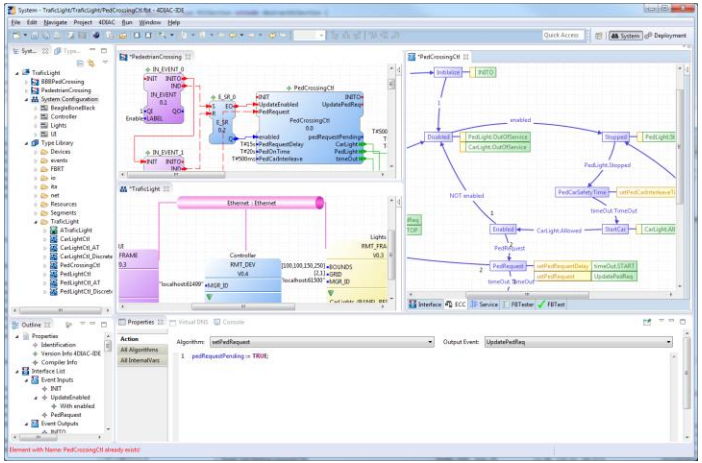




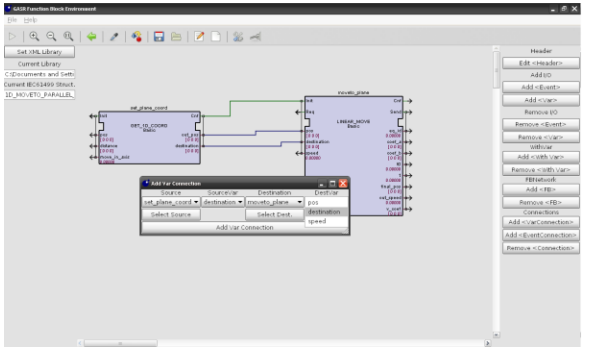
IEC 61499 – Development Environment



FBench
Focus on FB Validation
Add new Plugins
FBRT & FBDK RTEs Integration



4DIAC-IDE
Most Popular IDE
Eclipse based Tool
FORTE RTE Integration



GASR-FBE
Few Documentation
Java based Tool

IEC 61499 – Runtime Environment

Advantages
Distributed Architecture
Industrial Standard
Fog/Edge Oriented

Disadvantages
Languages Support (Python)
Third Party Integration
Archived Projects

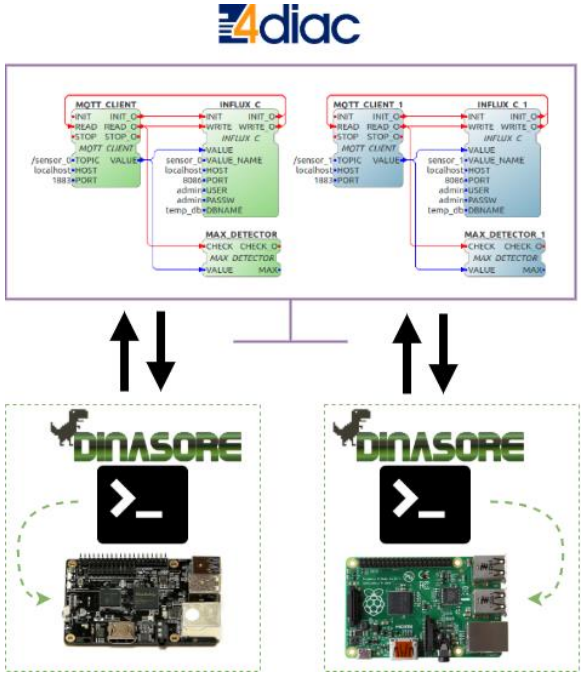
Runtime Environment	Programming Language	Portability	Execution Model	Reconf	Monitoring
FORTE [36]	C++	Inter-RTE	A1	✓	✓
nxtIECRT	C++	Inter-RTE	A1	✓	✓
FUBER [5]	Java	✗	A0	✓	✗
FBDK [40]	Java	Inter-RTE	A1	✓	✗
Archimedes [38]	Java, C++	✗	A1	✓	✓
ISaGRAF [7]	IEC 61131-3	✗	A4	✗	✗
RTFM-RT [23]	C	✗	A1	✗	✗
FBBean [30]	Erlang	✗	A2	✓	✓
Icaru-FB [28]	C	✗	A4	✓	✓

	Multitasking Implementation			
	Not Used	**Not Controlled**	**Time Slice**	**FB Slice**
Dynamic Order	BSEM (A0)	MTR (A1)	BS-PMTR (A2)	BS-NPMTR (A3)
Fixed Order	CBEM (A4)	x	CB-PMTR (A5)	CB-NPMTR (A6)

Implementation

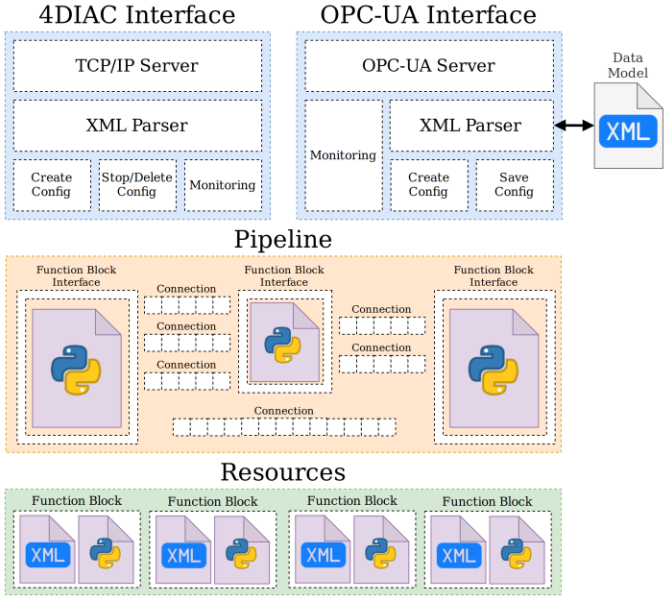
System Architecture

- Adoption of the **4DIAC-IDE** as Development Environment;
- **DINASORE** executes in each device as Runtime Environment;
  - FBs implemented in **Python**;
  - Third-party integration with **OPC-UA** applications;
  - Available online: github.com/DIGI2-FEUP/dinasore



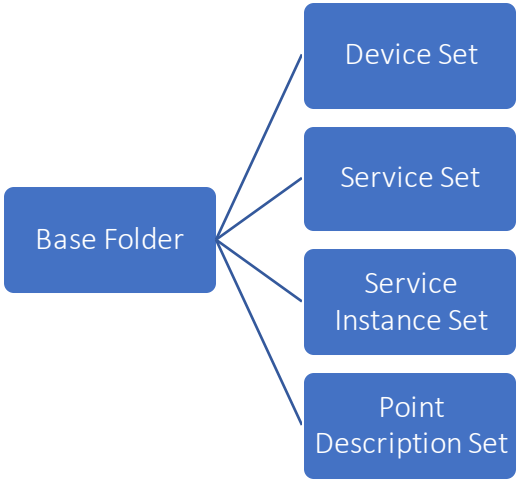
DINASORE

- Execution model based on the **producer-consumer** pattern:
  - Each FB performs in a **thread** that receives events, executes, and produces events.
- **FB resources** stored locally:
  - **Python file** to implement functionalities;
  - **XML file** to define the FB structure.
- 4DIAC integration using **TCP/IP sockets** and **XML messages**, allowing:
  - Create, Stop and Delete pipelines of FBs;
  - Monitor variables and trigger events of FBs.



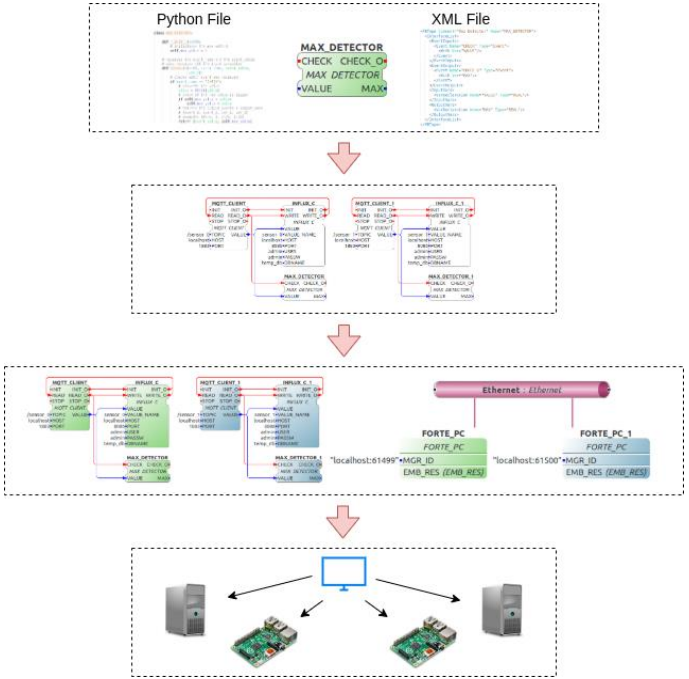
OPC-UA

- The OPC-UA integration facilitates the **connectivity** with external industrial platforms/entities;
- Data-model representation maps each FB into a different category, **device**, **service**, **start point,** or **endpoint**.
- The actual FBs pipeline is **stored locally** using the data-model (XML file), enabling the DINASORE **restart** if crashes.



Development Process

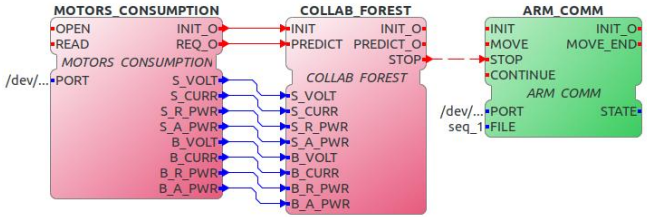
- The main **4 steps** to developing a CPPS using the DINASORE and the 4DIAD-IDE are:
  - Develop the **FBs sources files** (Python and XML);
  - **Orchestrate** the FBs using the 4DIAC-IDE (connect the FBs between each other);
  - **Associate** each FB with an **RTE device** (mapping);
  - **Deploy** the FBs to the respective RTE devices.



Test Case Scenarios

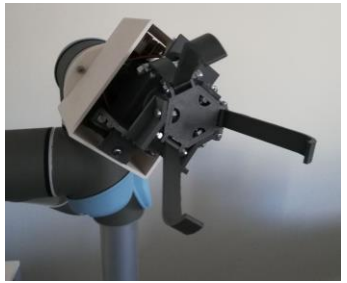
Collision Detection in Servo Motors

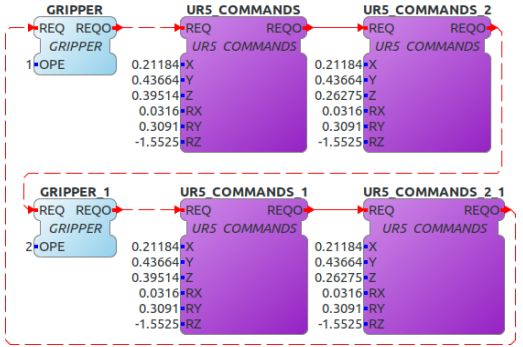
- Collision detection in a robotic arm (AL5D) based on servo motors;
- Real-time collection of the servo motor RMS consumptions: **voltage**, **current**, **real** and **apparent power**;
- Collision prediction using the **Random Forrest** classifier (0 – no collision, 1 - collision);
- In case of collision, the classifier sends an event to the **controller FB** to stop the robotic arm.



UR5 & Gripper Control

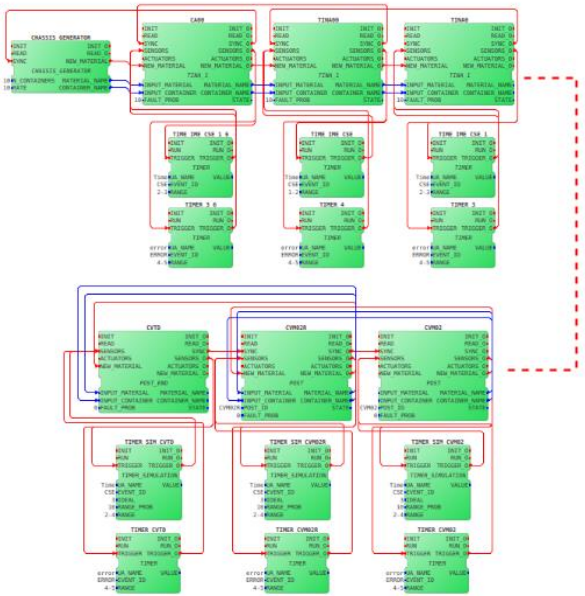
- **Moving object** (**pickup**) using a UR5 robotic arm and 3D printed gripper;
  - Gripper controlled by a Raspberry Pi and a servo motor.
- API controls the robotic arm position (X, Y, Z, Rx, Ry, and Rz);
- GPIOs control the servo motor position to open and close the gripper.





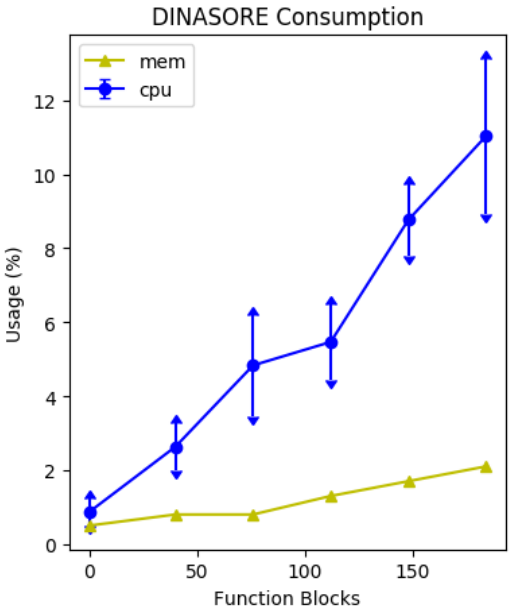
Manufacturing Applications

- **Simulation** of a sequential production process;
- Each **station** uses 3 FBs to simulate its behavior:
  - One responsible for emulating the station **state machine**;
  - Other 2 to replicate the **operation time** and the **error time**.
- Easy **replication/addition** of new stations to simulate new scenarios.



Performance Evaluation

- Evaluate how the DINASORE (core program) **performs** (CPU and RAM) when the **number of FBs increase**;
- **Escalate** the number of FBs (stations) using the previous scenario (factory simulation);
- More complexity causes more resources consumed (18 FBs / 1% of CPU usage);



Conclusions & Future Work

Conclusions & Contributions

- The proposed framework increases the **flexibility** of the traditionally closed and hard to reconfigure industrial systems, using **Python** as a general programming language;
- The validation scenarios prove the DINASORE useability for **robotic systems**, **sensorization/digitalization** platforms, or **data analytics** pipelines;
  - Using FBs for **MQTT** or **TCP** communication, **classification** and **regression** algorithms, control of **GPIOs**.
- The DINASORE transforms a **local heavyweight application** into a **distributed** solution performing in a cluster of devices.

Future Work

- As future work, the goal is to continue **developing new FBs**, applied in future scenarios;
  - Implement a **speculative computing** execution model in the DINASORE;
  - Integrate with an **optimization algorithm** to obtain the optimal device placement of FBs.

References

[1] T. Strasser et al., "Framework for Distributed Industrial Automation and Control (4DIAC)," in 2008 6th IEEE International Conference on Industrial Informatics, 2008, pp. 283–288, doi: 10.1109/INDIN.2008.4618110.

[2] L. I. Pinto, C. D. Vasconcellos, R. S. U. Rosso, and G. H. Negri, "Icaru-fb: An IEC 61499 compliant multiplatform software infrastructure," IEEE Transactions on Industrial Informatics, vol. 12, no. 3, pp. 1074–1083, 2016.

[3] L. Prenzel and J. Provost, "FBBeam: An Erlang-based IEC 61499 Implementation," in IEEE International Conference on Industrial Informatics (INDIN'19), 2019.

[4] K. Thramboulidis and A. Zoupas, "Real-time Java in control and automation: a model driven development approach," in 2005 IEEE Conference on Emerging Technologies and Factory Automation, 2005, vol. 1, pp. 8–pp.

[5] L. Prenzel, A. Zoitl, and J. Provost, "IEC 61499 runtime environments: A state of the art comparison," in 17th International Conference on Computer Aided Systems Theory (EUROCAST 2019), 2019.

Thanks for your attention!

Eliseu Pereira (eliseu@fe.up.pt)
João Reis (jpcreis@fe.up.pt)
Gil Gonçalves (gil@fe.up.pt)