Deliverable 2.3
Multi-agent infrastructure

Document version : Final
Submission Date   : 30/03/2018
Dissemination Level : Public
Contribution to   : WP2
Document Owner    : IPB
File Name         : GOOD MAN Deliverable 2.3
Revision          : 0.4

Project Acronym   : GOOD MAN
Project Title     : Agent oriented zero defect multi-stage manufacturing
Grant Agreement n. : 723764
Call              : H2020-IND-CE-2016-17
Project Duration  : 36 months, from 01/10/2016 to 30/09/2019
Website           : go0dman-project.eu
Revision History

<table>
<thead>
<tr>
<th>REVISION</th>
<th>DATE</th>
<th>INVOLVED PARTNERS</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>15/02/2018</td>
<td>IPB</td>
<td>Definition of the deliverable structure, document generation and creation of ToC</td>
</tr>
<tr>
<td>0.2</td>
<td>23/03/2018</td>
<td>IPB</td>
<td>Draft version of the deliverable.</td>
</tr>
<tr>
<td>0.3</td>
<td>23/03/2018</td>
<td>LOC</td>
<td>Contributions inclusion from LOC</td>
</tr>
<tr>
<td>0.4</td>
<td>26/03/2018</td>
<td>IPB</td>
<td>Sections improvement.</td>
</tr>
</tbody>
</table>

List of Contributors:
Giacomo Angioni (LOC), José Barbosa (IPB), José Augusto Carvalho (IPB), João Paulo Coelho (IPB), Adriano Ferreira (IPB), Getúlio Igrejas (IPB), Paulo Leitão (IPB), José Lima (IPB), Ricardo Peres (UNINOVA).

Disclaimer: The information in this document is subject to change without notice. Company or product names mentioned in this document may be trademarks or registered trademarks of their respective companies.

All rights reserved.
The document is proprietary of the GOOD MAN consortium members. No copying or distributing, in any form or by any means, is allowed without the prior written agreement of the owner of the property rights.
This document reflects only the authors’ view. The European Community is not liable for any use that may be made of the information contained herein.

GOOD MAN project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 723764.
Executive Summary

This deliverable, entitled “Multi-agent infrastructure”, contains a brief description of the Multi-Agent System (MAS) implementation, as specified previously in D2.1 of the GO0DMAN project.

The MAS infrastructure offers the agents backbone to later be customized and deployed in each of the use cases in the GO0DMAN project. The agent architecture comprises a society of distributed, autonomous and cooperative intelligent agents representing the production components disposed along the multi-stage manufacturing system, allowing the distributed data collection and the balancing of the data analysis for monitoring and adaptation among cloud and edge layers, to enable the earlier detection of process and product variability, and the generation of new optimized knowledge by correlating the aggregated data.

The document described the overall GO0DMAN system architecture, where the MAS is a central part, identifies the several frameworks possibilities for the agent implementation and shows some of the aspects coded in the agent side, as a representation of the overall development. Furthermore, the document points to a GIT repository where the developed code is shared as also one video demonstrating the operability of the developed MAS.

This deliverable document provides a baseline for the MAS customization in WP5, WP6, WP7 and WP8.
# Table of Contents

1. Introduction ......................................................................................................................... 8
   1.1 Contextualization ........................................................................................................... 8
   1.2 Objective of the deliverable .......................................................................................... 8
   1.3 Structure of the deliverable ......................................................................................... 8

2. Multistage Production and Zero-defect Manufacturing Distributed Architecture ............... 10
   2.1 Generic System Architecture ....................................................................................... 10
   2.2 Agent System definition ............................................................................................... 11

3. Frameworks for the agents implementation ........................................................................ 13
   3.1 Java Agent Development Framework (JADE) .............................................................. 13
   3.2 JACK ............................................................................................................................ 13
   3.3 FIPA-OS ....................................................................................................................... 13
   3.4 Smart Python multi-Agent Development Environment (SPADE) ................................. 14
   3.5 Java-based Intelligent Agent Componentware (JIAC) .................................................. 14

4. Agents implementation .......................................................................................................... 15
   4.1 The GO0DMAN Agent .................................................................................................. 15
   4.2 Resource Agent – Smart Inspection Tool interaction (OPC-UA) ................................. 15
   4.3 Resource Agent – Raw data collection (MQTT) .......................................................... 16
   4.4 IMA – Rules consulting (REST) ................................................................................... 18
   4.5 IMA – send data to Data Analytics and Knowledge Management (REST) ................. 18
   4.6 PTA – launching PAs ................................................................................................... 19
   4.7 Source Control Management System ........................................................................... 19
   4.8 A DC motor pre demonstrator ...................................................................................... 20

5. Conclusions .......................................................................................................................... 22

References .................................................................................................................................... 23

Annex A ....................................................................................................................................... 25
List of Figures

Figure 1 – Global GOODMAN system architecture ................................................................................. 10
Figure 2 - GOODMAN DC motor multi-stage production system ................................................................. 20
Figure 3 – DC motor testbed MAS instantiation .......................................................................................... 21
Figure 4 – Exemplification of a Multi-agent System [14]. ........................................................................... 25
List of Tables

Table 1 – Agent development frameworks comparison................................................................. 14
Acronyms

AMS  
Agent Management System

CPS  
Cyber-Physical Systems

DB  
Data Base

DF  
Directory Facilitator

FIPA  
Foundation for Intelligent Physical Agents

FIPA-OS  
Foundation for Intelligent Physical Agents - Open Source

GOOD MAN  
aGent Oriented Zero Defect Multi-stage mANufacturing

GRACE  
inteGration of pRocess and quAlity Control using multi-agEnt technology

HMI  
Human-Machine Interface

IMA  
Independent Meta Agent

IoT  
Internet of Things

MAS  
Multi-Agent System

MES  
Manufacturing Execution System

MQTT  
Message Queuing Telemetry Transport

OPC-UA  
Open Platform Communications - Unified Architecture

PA  
Product Agent

PTA  
Product Type Agent

RA  
Resource Agent

SIT  
Smart Inspection Tool

UI  
User Interface

URI  
Uniform Resource Identifier

ZDM  
Zero Defect Manufacturing
1. Introduction

1.1 Contextualization

Nowadays, the modern and competitive companies must be able to track rapid technological changes while carrying out the manufacture of products with complex features, which commonly requires the assembly of a large number of components. Moreover, the dynamic nature of today’s manufacturing environments, compels organizations to an incessant reassessment in an effort to respond to continuous challenges in the field of manufacturing management.

Multi-stage manufacturing systems [1], which are typical in important industrial sectors, such as automotive, household appliance and semiconductor manufacturing, are inherently complex. Among other characteristics, it is common to have multiple stages with mixed sequential and/or parallel configurations, feedback/feedforward loops, and mixed data types that arise from multiple processes. In this context, the application of the zero defect manufacturing (ZDM) philosophy [2], together with recent technological advances in CPS, namely Internet of Things (IoT), big data, and advanced data analytics, presents significant challenges and opportunities to develop new methodologies aiming at the continuous improvement of process efficiency and product quality.

In the actual European manufacturing context, the development of advanced technology and the following up-take by the industry is of strategic importance. This technological improvement aims to boost competitiveness while targeting several aspects of the manufacture processes such as reduce waste and costs, and increase processes efficiency and quality tracking. In this context, the EU H2020 GOOD MAN (aGent Oriented Zero Defect Multi-stage mANufacturing) project (see http://go0dman-project.eu/) aims to integrate and combine process and quality control for multi-stage manufacturing systems using a distributed system architecture built upon an agent-based CPS and smart inspection tools designed to support ZDM strategies.

In this approach, the multi-agent system (MAS) [3] infra-structure, combined with data analytics, provides real-time and early identification of deviations allowing to prevent the occurrence of defects at a single stage and their propagation to downstream processes, enabling the global system to be predictive, by early detecting faults, and proactive, by self-adapting to different conditions.

This approach is aligned with Industry 4.0 trends, contributing for achieving a dynamic and continuous system improvement in multi-stage manufacturing environments, addressed by the ZDM philosophy.

1.2 Objective of the deliverable

This deliverable contains the outcome of Task 2.3, entitled “Implementation of the multi-agent system infrastructure”, which addresses the development of the MAS infrastructure as specified during the T2.1.

1.3 Structure of the deliverable

The document is divided into 5 chapters. After this brief introduction, Chapter 2 overviews the GOODMAN overall architecture, particularly highlighting the connections between the MAS with the remaining blocks of the system. Chapter 2 also recalls the main functionalities of the MAS, as defined previously in the project execution. Chapter 3 presents a generic analysis of the available MAS development frameworks. Chapter 4 presents some of the aspects used during the agent development, particularly those regarding the connection of the agents with the remaining of the
GOODMAN building blocks, namely with the data analysis, knowledge management and smart inspection tools. Chapter 4 also points out to a repository where the MAS developed code can be downloaded and makes an initial remark on a video to be developed as proof-of-concept and pre-validator. Finally, Chapter 5 presents some concluding remarks.
2. Multistage Production and Zero-defect Manufacturing Distributed Architecture

This section briefly describes the overall generic architecture adopted in the GO0DMAN project. After this, there is a dedicated part where the Multi-Agent System (MAS) specification is recalled from the previous deliverables and where the infrastructure is depicted.

2.1 Generic System Architecture

Figure 1 depicts the GO0DMAN overall architecture, where the MAS is a key part. One can divide the analysis of the architecture in two parts, namely:

- Operative Technology (OT)
- Information Technology (IT)

On the OT side, the data collection from the information, potentially available at the physical process, is accomplished by the Resource Agents (RA), through the close interaction with the Smart Inspection Tools (SIT). Besides this, the RA is also interacting with the processing physical resources and/or with the human operators via the use of Human Machine Interfaces (HMI).

As specified in D2.1 [4], the MAS is responsible for the distributed data collection, local and global analysis of the data being collected as also for the representation of the product types that a given company is able to produce and by the product instances itself. Additionally, the MAS is also
responsible for the bridge between the OT and IT world, being also this approach aligned with the current industrial trends.

At a higher level, one can find the wide spectrum building blocks, namely the Zero Defects Manufacturing (ZDM) data analytics and the ZDM knowledge management, that are responsible for analysing data and providing knowledge to it, respectively.

With this, the knowledge generated after the data has been analysed, can origin the creation of rules that the agents can use to locally, and rapidly, analyse the data being collected.

2.2 Agent System definition

The structure of the four types of agents was implemented accordingly with the D2.1 specification, namely for the:

- Independent Meta Agent (IMA)
- Product Type Agent (PTA)
- Product Agent (PA)
- Resource Agent (RA)

Recalling some basic functions that each one of the agents should be able to provide, one can describe:

- Independent Meta Agent
  - Aggregation of the data collected in a distributed way from the system's agents.
  - Management of the subscription of optimisation services requested by local agents, i.e. PAs and RAs.
  - Monitoring the system performance to detect improvement opportunities.
  - Triggering the batch data analysis offered by the Data Analytics Environment, running at the cloud level, aiming to identify meaningful correlations and new knowledge towards the optimisation of the production system.
  - Propagation of new or adjusted knowledge, provided from the Knowledge Management Environment via the rule repository, to the local agents.

- Product Type Agent
  - Launching PAs according to the production orders coming from MES.
  - Collection and storage of data related to the execution of the products in the production line.
  - Monitoring the on-going production of products related to a specific model.
  - Optimisation and adjustment of the process plan associated to the product model.
  - Traceability of the produced products in the production line.

- Product Agent
  - Collection and storage of production data (i.e. process and inspection data) related to the execution of the product along the production line.
• Adaptation of the process and inspection parameters of operations to be executed by the resources, according to the local knowledge and historical data.

• Monitoring the evolution of the product production along the line to detect, amongst others, possible deviations from the plan and quality degradation.

• Pre-processing the collected data to filter those that are sent to IMAs (mitigating in this way the latency and bandwidth).

• **Resource Agent**
  
  • Collection and storage of data related to the process stations, using proper Internet of Things (IoT) technologies, e.g., OPC-UA (Open Platform Communications - Unified Architecture) and MQTT (Message Queuing Telemetry Transport). In case of OAs, the integration is performed through a proper HMI (Human-Machine Interface) that can be also be interconnected using the previously technologies.

  • Adaptation of the operations' parameters, according to the local knowledge and historical data from the resource perspective.

  • Monitoring the evolution of the resource performance to detect, amongst others, possible quality of service degradation.

  • Pre-processing the collected data to filter those that are sent to IMAs (mitigating in this way the latency and bandwidth problems).

For further details on each of the agent specification, the reader is advised to consult the D2.1 [4].
3. Frameworks for the agents implementation

In order to implement the MAS, several frameworks for the agent development are available. This section makes a brief description of the most used frameworks available. Of special interest is also the study of the ones which license scheme allows the consortium the future deployment without any legal constraints.

3.1 Java Agent DEvelopment Framework (JADE)

JADE is an open source framework, oriented towards the development and implementation of multi-agent systems compliant with the Foundation for Intelligent Physical Agents (FIPA) specifications related to agent management, communication and message transport. The agent system is composed by a main container comprising the Agent Management System (AMS), Direct Facilitator (DF), as well as a Remote Method Invocation (RMI) registry (used for intra-platform communication). Additional agent containers can be created, being able to run in the same host, or on remote hosts connected to the main container via wired or wireless networks. Moreover, JADE allows each agent to be dynamically discovered by other agents and communicate with each other using asynchronous messages. The structure of these messages is based on the FIPA standard Agent Communication Language (ACL), containing fields such as message context and waiting time. Agents are identified by a unique global name, have a life cycle, beliefs and capabilities that are resources implemented by the programmer [5].

3.2 JACK

JACK Intelligent Agents is an agent development platform, developed by Autonomous Decision Making Software fully integrated with Java programming language. JACK consists of four main components namely: (1) JACK Agent Language (JAL), that incorporates the Java language; (2) JACK compiler that translates JAL into Java code to be executed into the JACK runtime engine; (3) JACK Kernel that is the runtime execution of the JACK compiler and (4) JACK Development Environment that is an integrated graphical user interface for the development of JACK agent applications. JACK is designed for closed systems and the agents are designed to work with each other. JACK agents need to know the location of other agents in the system [6], [7].

In order to improve interoperability JACK extensions have been developed, namely, JACK TEAMS and FIPA JACK [8].

3.3 FIPA-OS

FIPA-OS is an open source, java agent platform that enables the development of FIPA compliant agents. Similarly, to JADE framework, FIPA-OS has as core components: Directory Facilitator (DF), Agent Management System (AMS), Agent Communication Channel (ACC) and the Internal Platform Message Transport (IPMT). It allows to build agents using two agent shells that are implemented in Java base classes. The first shell is oriented towards the message transport, retrieval and buffering, whereas the other the second shell, is oriented towards the management of ACL messages.

During its development form 1999 until 2003 two versions of FIPA-OS were released: Standard FIPA-OS with compatibility for Java 2 and Java 1.1; MicroFipa-OS designed to be executed on PDA’s or pocketPC operating systems [9].
3.4 Smart Python multi-Agent Development Environment (SPADE)

SPADE is an agent development platform developed using Python compliant with FIPA standard. The communications in SPADE use Jabber protocol that is an instant messaging internet standard protocol allowing to exchange messages and structured information among the peers. SPADE provides presence notification, which allows entities to declare their current status to those with they share a bond (e.g. ‘Available’, ‘Busy’, etc.). Another feature of SPADE is the Multi-user Conference (MUC) that can be used to create “private communication rooms” allowing to share data among a selected amount of entities. For instance, in a virtual auction, the a auctioneer agent can create a password-protected channel and provide entrance to a given selection of agents [10].

3.5 Java-based Intelligent Agent Componentware (JIAC)

JIAC is a Java oriented agent architecture that aims to ease the development and operation of large-scale distributed applications. The agents in JIAC are programed using JADL++ (JIAC Agent Description Language), which features knowledge on the ontology language OWL, as well as, scripts to implement plans and protocols. The architecture also implements dynamic service discovery and supports component exchange during runtime. Every agent comprises a set of components that enables the agent to execute basic functionalities, implement abilities and access the environment. During runtime, the platform allows to monitor all the present entities in the systems using a generic management framework [11].

<table>
<thead>
<tr>
<th>Platform</th>
<th>License</th>
<th>FIPA Compliant</th>
<th>Still Maintained</th>
<th>Being Developed</th>
</tr>
</thead>
<tbody>
<tr>
<td>JADE</td>
<td>Freeware</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>JACK</td>
<td>Commercial</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>FIPA-OS</td>
<td>Freeware</td>
<td>✓</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>SPADE</td>
<td>Freeware</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>JIAC</td>
<td>Freeware</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

After this analysis, and based on the consortium previously knowledge, the JADE framework was selected as the one where the GOODMAN MAS will be developed.
4. Agents implementation

As described before, the JADE framework was selected for the MAS development part. This section briefly describes the agent implementation taking particular attention to the interactions between the agents and the remain of the GOODMAN building blocks.

4.1 The GOODMAN Agent

As specified in D2.1. [4], all the agents developed inherit key features from a generic GOODMAN agent, namely functions of agent initialization procedures, store data, transfer data or finalization features.

Below, one can find one of such functions, namely the agent registration in the system.

```java
public void doRegister(Agent agent, String agentType)
{
    boolean executed_registration = false;
    DFAgentDescription dfd = new DFAgentDescription();
    ServiceDescription sd = new ServiceDescription();
    sd.setType(agentType);
    sd.setName(agent.getName());
    sd.addOntologies("GOODMAN_DATA_MODEL");
    dfd.setName(agent.getAID());
    dfd.addServices(sd);
    try {
        DService.register(agent, dfd);
        executed_registration = true;
        eu.h2020.go0dman.support.BasicMethods.Log().info("The agent " + agent.getName() + " has register in the DF (first try) (" + agent.getAID() + ")");
    } catch (Exception e) {
        eu.h2020.go0dman.support.BasicMethods.Log().error(agent.getLocalName() + " registration with DF unsucceeded (first try).".);
    }
    if (!executed_registration)
    {
        try {
            DService.deregister(agent, dfd);
            DService.register(agent, dfd);
            eu.h2020.go0dman.support.BasicMethods.Log().info("The agent " + agent.getName() + " has register in the DF (second try) (" + agent.getAID() + ")");
        } catch (Exception e) {
            eu.h2020.go0dman.support.BasicMethods.Log().error(agent.getLocalName() + " registration with DF unsucceeded (second try).".);
        }
    }
}
```

In the depicted code excerpt it is possible to observe that the agent register basic information, namely its type and name as also that the agent is instructed to use the GOODMAN data model.

4.2 Resource Agent – Smart Inspection Tool interaction (OPC-UA)

This interaction provides the agent with pre-processed data (i.e. already processed in the SIT(Smart Inspection Tool)) and is achieved by means of using the OPC-UA protocol. As an example, it is depicted in the following code excerpt the RA subscription to the necessary data items.
Object obj = null;
    try {
        obj = parser.parse(new FileReader("./src/main/resources\" +
        agent.getLocalName() + ".json"));
    } catch (IOException | ParseException e1) {
        // TODO Auto-generated catch block
        e1.printStackTrace();
    }

    JSONObject jsonObject = (JSONObject) obj;
    JSONArray topicList = (JSONArray) jsonObject.get("TOPIC_LIST_OPCUA");
    Iterator<String> iterator = topicList.iterator();
    while (iterator.hasNext()) {
        NodeId nodeId = NodeId.parse(iterator.next().toString());
        BasicMethods.log().info("I've added to the OPC-UA server: " + nodeId.toParseableString());
        ReadValueId readValueId = new ReadValueId(nodeId,
        AttributeId.Value.uid(), null, QualifiedName.NULL_VALUE);
        MonitoringParameters parameters = new
        MonitoringParameters(uint(clientHandles), 1000.0, null, uint(10),
        true);
        MonitoredItemCreateRequest request = new
        MonitoredItemCreateRequest(readValueId, MonitoringMode.Reporting,
        parameters);
        subscription.createMonitoredItems(TimestampsToReturn.Both,
        newArrayList(request)).get();
    }

    Similarly, the write function when the RA optimizes the parameters of the SIT is accomplished using the partial code representation.

    Variant v = new Variant(i);
    DataValue dv = new DataValue(v, null, null);
    CompletableFuture<List<StatusCode>> f = client.writeValues(nodeIds,
    ImmutableList.of(dv));

4.3 Resource Agent – Raw data collection (MQTT)

This connection allows the RA, and the system, to be provided with the raw signal data, used to ensure that future traceability issues can be analyzed exploring raw data. To accomplish this, the RA uses a MQTT connection. A code excerpt is provided in the following example.

Object obj = parser.parse(new FileReader("./src/main/resources\" +
agent.getLocalName() + ".json"));
JSONObject jsonObject = (JSONObject) obj;
JSONArray topicList = (JSONArray) jsonObject.get("Topic List");
Iterator<String> iterator = topicList.iterator();
String topic;
while (iterator.hasNext()) {
    try {
        topic = iterator.next().toString();
        int subQoS = 1;
        myClient.subscribe(topic, subQoS);
    }
BasicMethods.log().info("Topic " + topic + " has been added.");
} catch (Exception e) {
    e.printStackTrace();
}

This example shows the subscription of the data topics from the RA side. To this extend, the agent is parameterized and in a .json file that is iterated in order to subscription. Note that, for this case, the subscription quality of service is 2, ensuring that that is delivered exactly once.

If the agent has rules to be processed to any of these data points, it executes them by launching a new agent behavior (i.e. RuleVerification behavior) using the following code excerpt.

```java
agent.addBehaviour(new RuleVerification(BasicMethods.ruleCondition));
```

And uses the data model for the decoding of the rules that are needed to be executed.

```java
String auxOutput = BasicMethods.ruleCondition.getAssociatedResult().getResult();
Collection<GMRuleCondition> ruleCondition = ruleManagement.getAssociatedConditions();
for (GMRuleCondition rule : ruleCondition) {
    Collection<AMLInternalLink> amLink = rule.getToProperty();
    for (AMLInternalLink am : amLink) {
        auxVariable = am.getName();
    }
    boolean conditionVerified = false;
    switch (rule.getOperator().toString()) {
        case "GREATER_THAN":
            if (!BasicMethods.variablesValues.get(auxVariable).isEmpty()) {
                if (Float.compare(Float.parseFloat(BasicMethods.variablesValues.get(auxVariable)), Float.parseFloat(rule.getValue())) > 0) {
                    conditionVerified = true;
                } else {
                    conditionVerified = false;
                }
            } else {
                break;
            }
            case "LESS_THAN":
            if (!BasicMethods.variablesValues.get(auxVariable).isEmpty()) {
                if (Float.compare(Float.parseFloat(BasicMethods.variablesValues.get(auxVariable)), Float.parseFloat(rule.getValue())) < 0) {
                    conditionVerified = true;
                } else {
                    conditionVerified = false;
                } else {
            conditionVerified = false;
        }
    }
    auxMap.put(i, conditionVerified);
    i++;
}
Briefly, based on the defined GO0DMAN Data Model and on the created rules, the agent needs to check whether the received data is associated with any created rule. Therefore, the agent gets all the rules associated by executing `getAssociatedConditions`, and with this, checks the rule to be signal to be executed. In the cases of a positive rule identification, the agent takes the necessary actions (`getAssociatedResult`), e.g., by warning the user of this issue.

### 4.4 IMA – Rules consulting (REST)

IMA has a polling mechanism that allows to check whether any particular agent has associated rules. The following code excerpt represents the procedure to discover the agents that are available in the system (i.e. PTA, PA and RA agent types) and the search of rules in the rules repository for the PTA agent types.

```java
DFAgentDescription[] PTA_AGENTS = eu.h2020.go0dman.support.BasicMethods.searchForAvailableAgents(myAgent, "PTA");
DFAgentDescription[] PA_AGENTS = eu.h2020.go0dman.support.BasicMethods.searchForAvailableAgents(myAgent, "PA");
DFAgentDescription[] RA_AGENTS = eu.h2020.go0dman.support.BasicMethods.searchForAvailableAgents(myAgent, "RA");

// query the rule repository service for available
for (int i = 0; i < PTA_AGENTS.length; i++) {
  try {
    Collection<GMManagementRule> ruleList = eu.h2020.go0dman.agent.IMA.RuleManagement
      .ruleListGet(PTA_AGENTS[i].getName().getLocalName().toString());
    if (!((ruleList == null)) {
      for (GMManagementRule rule : ruleList) {
        sendRuleToAgent(PTA_AGENTS[i].getName().getLocalName(), rule);
      }
    }
  } catch (Exception e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
  }
}
```

In the previous excerpt of code, it is also possible to see that if a rule for a given agent is found, the IMA will propagate it to the designated agent.

### 4.5 IMA – send data to Data Analytics and Knowledge Management (REST)

In order for the agents to send data to the Data Analytics and Knowledge Management, a REST post is executed. The following code excerpt uses the BOC endpoint as an example for the IMA to propagate data to the Knowledge Management tool.

```java
static String REST_URI = "https://orbeet.boc-group.eu/DataAssetEngine/requestBin/";

public static void send(String content) {
  try {
    Client client = Client.create();
    WebResource webResource = client.resource(REST_URI);
    ClientResponse response = webResource.type("application/json").post(ClientResponse.class, content);
    if (response.getStatus() != 200) {
      Page 18 of 26
      Deliverable 2.3 Multi-agent infrastructure
    }
  }
}
eu.h2020.go0dman.support.BasicMethods.log().error("Failed : HTTP error code : " + response.getStatus());
    throw new RuntimeException("Failed : HTTP error code : " + response.getStatus());
} else {
    System.out.println(" < - " + response.getStatus());
    eu.h2020.go0dman.support.BasicMethods.log().info("Output from Server (Data Asset Engine) .... \n");
    eu.h2020.go0dman.support.BasicMethods.log().info(response.getEntity(String.class));
} catch (Exception e) {
    e.printStackTrace();
}

Briefly, IMA collects the received agent data and executes a POST method to the provided URI (Uniform Resource Identifier).

Similarly, the process is identical for the Data Analytics tool, being majorly necessary to change the URL endpoint.

4.6 PTA – launching PAs

The PTA is responsible for the launching of the PAs. The following code excerpt depicts the launch process of the PA from the PTA.

private void launchNewPA(String pta_type, String nC_12) {
    Runtime rt = Runtime.instance();
    p = new ProfileImpl();
    mainController = rt.createAgentContainer(p);
    Object[] arg = { pta_type };  
    try {
        ac = mainController.createNewAgent(nC_12, "eu.h2020.go0dman.agent.PA.PA", arg);
        ac.start();
        BasicMethods.log().info(" - " + myAgent.getLocalName() + " < - i've launched a PA named " + nC_12);
    } catch (StaleProxyException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}

In this way, every time there’s the need to launch a new PA, the PTA will create it by running createNewAgent method, passing the agent name and stating it as being of the type PA.

4.7 Source Control Management System

The development of the GO0DMAN MAS was versioned and is maintained using the GIT source control management.

The GO0DMAN MAS can be accessible through the link:

    ssh://git@gitlab.estig.ipb.pt:4589/GCAR/go0dmanMAS.git

More particularly, this helps the development team to keep track of the code developments, allowing multiple developers to make commits of code changes as also allows the project roll-back due to some broken code.
4.8 A DC motor pre demonstrator

During the development of the MAS infra-structure, a pre-demonstrator was built to act as a continued integrator and micro-step validator. In fact, the proposed “GO0DMAN DC motor multi-stage production system” (see Figure 2) enabled the development, test and validation of the several agents that compose the GO0DMAN MAS as also the diverse connections to the other system building blocks.

To this extent, the DC motor system mimics the multi-stage production system by setting three quality control stations, namely:

- Electrical Test: performing the necessary tests to verify electrically the quality of the motor;
- Mechanical Test: performing the necessary tests to verify mechanically the quality of the motor;
- Temperature Test: performing the necessary tests to verify the temperature operation quality of the motor.

Therefore, the MAS to manage this setup is composed by 3 RA (each one representing one quality control station), 1 PTA (representing the DC motor product type), 1 IMA (representing the wider knowledge on the system and acting as interface with the higher level blocks) and as many PAs as the current products being produced, as seen in Figure 3.

![Figure 2 - GO0DMAN DC motor multi-stage production system](image)

---

Figure 2 - GO0DMAN DC motor multi-stage production system
From this, one can see the graphical representation of the distribution of the GOODMAN MAS among the DC motor testbed.
5. Conclusions

This text deliverable intended to provide a short contextualization to the MAS development. In fact, the core part of the deliverable is the MAS infrastructure itself.

The core MAS specification was accomplished during the development of T2.3 of the GOODMAN project, being now necessary its customization for each one of the use cases. This customization will be carried out during the remaining of the project execution and accordingly with the integration WP5.

Lastly, and in order to have a MAS, and the whole building blocks, pre-validator, a DC motor testbed was developed, the MAS instantiated for this and their connections with the SIT, data analytics and knowledge management validated.
References


Annex A

This annex overviews the basic concepts of the Multi-Agent Systems (MAS) paradigm and its application to develop decentralized manufacturing systems.

Multi-Agent Systems (MAS) [3], [12] is a paradigm derived from the distributed artificial intelligence field that promote the distribution, decentralization, intelligence, autonomy and adaptation, contributing to achieve flexibility, robustness, responsiveness and reconfigurability. MAS provides an alternative way to design control systems, based on the decentralization and parallel execution of activities conducted by autonomous entities, known as agents, that ensures the promptly respond to condition changes, due to the inherent capabilities of agents to adapt to emergence without external intervention.

An agent can be defined as “an autonomous component that represents physical or logical objects in the system, capable to act in order to achieve its goals, and being able to interact with other agents, when it doesn’t possess knowledge and skills to reach alone its objectives” [13]. The most important properties of an agent are:

- Autonomy, i.e. the ability to perform their own decisions without human intervention.
- Cooperation, i.e. the ability to interact with other agents, acting together to achieve a global system goal, or shared goals.

Additionally, agents can exhibit other characteristics, such as intelligence, pro-activeness, adaptation and social behaviour.

Since rare applications consider agents in an isolated manner, these systems form multi-agent systems that can be defined as a “society of agents that represent the objects of a system, capable of interacting to achieve their individual goals when they have not enough knowledge and/or skills to achieve individually their objectives” [13]. Agents may have a counterpart representation, namely physical devices, e.g., robots or inspection stations, or logical objects, e.g., schedulers or orders.

In these systems, the overall behaviour emerges from the interaction among the distributed agents, with each individual agent contributing with its knowledge and skills, as illustrated in Figure 4 (note that each agent has only a partial view of the system). In this context, more robust solutions are achieved since a MAS solution distributes the control functions over a network of distributed agents, overcoming the problems related to performance bottlenecks or critical failures, which are typical problems associated to centralized systems.

![Figure 4 – Exemplification of a Multi-agent System](image-url)
Especially in industrial environments, the use of MAS focuses on the introduction of distributed intelligence that can be performed in automation devices (e.g. sensors, actuators, robots and machines), systems and infrastructures, effectively enabling the creation and interaction of cyber-physical components/systems [15]. In fact, MAS play a crucial role in the development of CPSs, allowing to design control systems in a decentralized manner based on the distribution of control functions by autonomous and cooperative agents [13], offering important characteristics, such as modularity, flexibility, robustness, reconfigurability, and responsiveness.

MAS technology is being applied to several industrial applications in a CPS context, namely smart production smart electric grids, smart logistics, and smart healthcare [16]. In particular, agent-based solutions for smart production have been developed and installed in industrial environments during the last two decades, as surveyed, e.g., in [17], [18], [19] and [16], and considered as a technology driver in several European funded projects. As examples, the FP5 PABADIS and FP6 PABADIS’PROMISE [20] projects targeted a distributed agent-based manufacturing control system and the EU FP7 GRACE project deployed an agent-based solution for integrating process and quality control in the washing machine production line of Whirlpool in Naples, Italy [21], showing benefits in terms of production efficiency, product quality and reduction of the scrap costs. The EU FP7 IDEAS project used MAS technology to enable the plug and produce deployment of modular equipment without reprogramming [22] and the EU FP7 PRIME project used a MAS framework to enhance assembly systems with standardized plug and produce process and control solutions to allow rapid reconfiguration and deployment, performance monitoring, self-awareness, and evolutionary system adaptation [23]. The EU FP7 ARUM project used MAS combined with service-oriented architectures (SOA) to develop knowledge-based applications that support mitigation strategies to respond faster to unexpected events in ramp-up production of complex and highly customized products, such as airplanes or shipyards [24].