IMIC'25 – The Flexibac Problem: development of a multi-agent solution using a model-based system engineering approach

J. Terlez[1], A. Boungab[1], W. Derigent[2]

[1]Master Ingénierie des Systèmes Complexes, Faculté des Sciences & Techniques, Université de Lorraine
jules.terlez6@etu.univ-lorraine.fr,
abdessamad.boungab3@etu.univ-lorraine.fr

[2]CRAN, CNRS UMR 7039, Université de Lorraine,
Campus Sciences, bd des Aiguillettes, 54506 Vandoeuvre-Lès-Nancy Cedex, France
william.derigent@univ-lorraine.fr

Abstract. The first edition of the IMI contest proposes the Flexibac Problem, a real-life problem, extracted from an industrial collaboration between the group La Poste (French postal services) and Nantes University, aiming to design the control procedures needed to optimize an existing mail sorting system in which a 6-axis robot is introduced. This paper presents the development of an online and reactive solution, relying on a holonic architecture. Beside the result obtained by this solution, the originality of our approach is to define this solution following a Model-Based System Engineering methodology, ensuring a rational and traceable development.

Keywords: holonic architecture, MBSE, Flexibac, IMIC

1 Introduction

The first Intelligent Manufacturing International Contest (IMIC'25) proposes a competition subject based on a real industrial problem, where the main idea is to design control procedures needed to optimize an existing mail sorting system. The IMIC website presents a complete description of the system, whose is schematized Fig. 1. In this system, the mail is stored in boxes arriving in buffer *A*. Due to their weight, the company La Poste decided to equip their installation with robots to handle some of the most common boxes and reduce the physical strain on workers and to lower the risks of musculoskeletal disorders. The robotic solution loads carts with these boxes ensuring each cart contains the same destination. As a result, depending on its destination, a box arriving in buffer *A* can either be sent to the robot (buffer *B*) and treated automatically, or be handled manually by human workers. The Flexibac problem establish several goals that should be attained by the contestants' proposals. Indeed, the proposed solution should maximize the number of boxes robotically handled while minimizing the

number of cart changes, to lower the operators' work. These two objectives should be targeted while considering all system constraints like buffer capacities, cycle times or cart capacities.

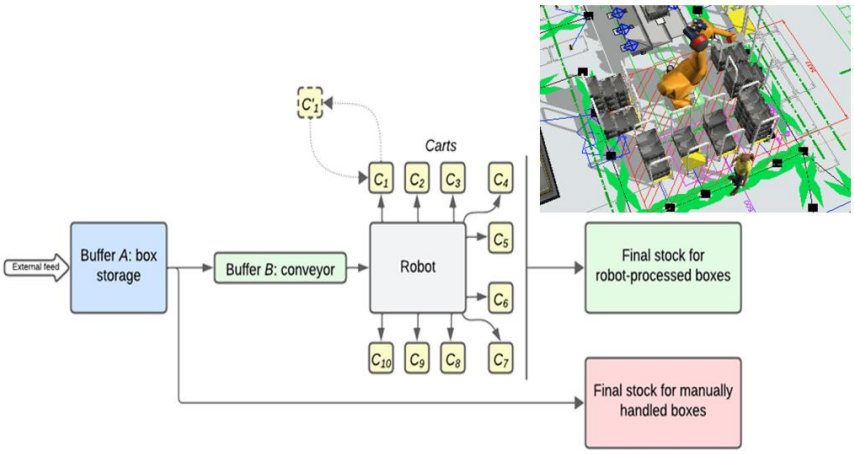

Fig. 1. Description of the Flexibac Problem

This article presents the design and results of the solution proposed by our team. One originality of this proposal is to use model-based system engineering (MBSE) to design the solution, thus guaranteeing all requirements have been considered and traced all along the development process. The rest of the paper is organized as follows: the solution development is detailed in section 2, and the test on different instances is realized in section 3. The last section presents some conclusions and perspectives on this work.

2 Specification of the solution with a MBSE methodology

The MBSE methodology employed in this paper is currently taught at the Master of Complex Systems of the University of Lorraine. This methodology is inspired from [1] and SysML [2]. It is composed of a series of interlinked diagrams, going from the skateholder needs to the final organic architecture of the system. The rest of this section will be divided into two parts, the first one dedicated to *Requirements Engineering* and the second to *System Architecture.*

2.1 Requirements Engineering

This MBSE methodology starts with the *Problem-Finality-Mission-System* (PFMS) diagram. Indeed, to correctly develop a system, the *problem* should be well described, as well as the global aim of the system designed to solve the problem, as known as *finality*. Once identified, a corresponding *mission* can be underlined for the corresponding *system*. Fig. 2 depicts each of these parts for the Flexibac case study.

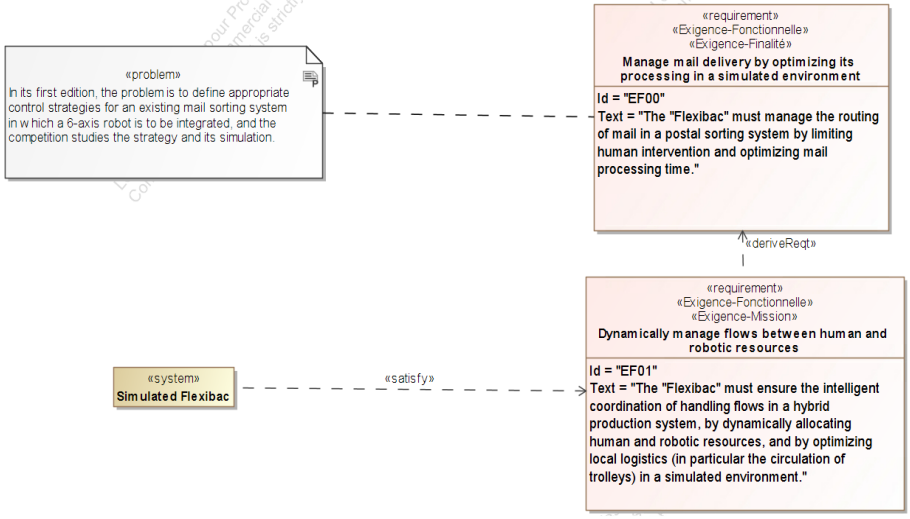

Fig. 2. PFMS Diagram

This first phase is then followed by the identification of the different system life phases as described in Fig. 3. The system passes through 4 different stages of life:

- Design and production: In this stage, the system is designed and realized.
- Exploitation: the system is run and used.
- Update: the system is updated and relaunched.
- Archiving: in this final stage, the system is stopped and archived.

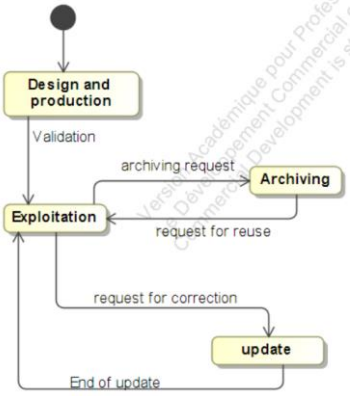

Fig. 3. System life phases

Listing all the different phases of life enables us to enumerate all the different stakeholders interacting with the system. Each phase could then be detailed by a use case diagram where the corresponding actors should be inserted. For the sake of brevity, only the description of the Exploitation phase is proposed here, in **Fig. 4**, where the different use cases of this phase are presented. The main use case is to *dynamically manage flows between human and robotic resources*. It is refined into 2 sub use cases, one to *manage the switching of bins between the*

(human/robot) in the simulated environment and the second one to *optimize the management of carts around the robot in the simulated environment.* The first one addresses the problem of transferring boxes from buffer *A* either to the robotic cell via buffer *B,* or to send them directly to human workers. The second one addresses the problem related to the management of carts around the robot, when one of them is full. Each of these use cases is then related to a requirement depicted Fig. 5. In addition to these functional requirements, other requirements (constraints, performances) are added to the requirement list as shown in Fig. 6.

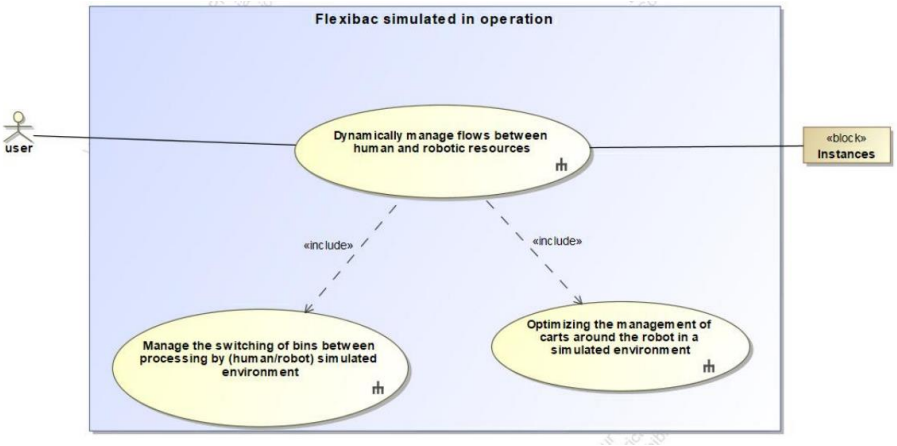

Fig. 4. Use case describing the Flexibac system in operation

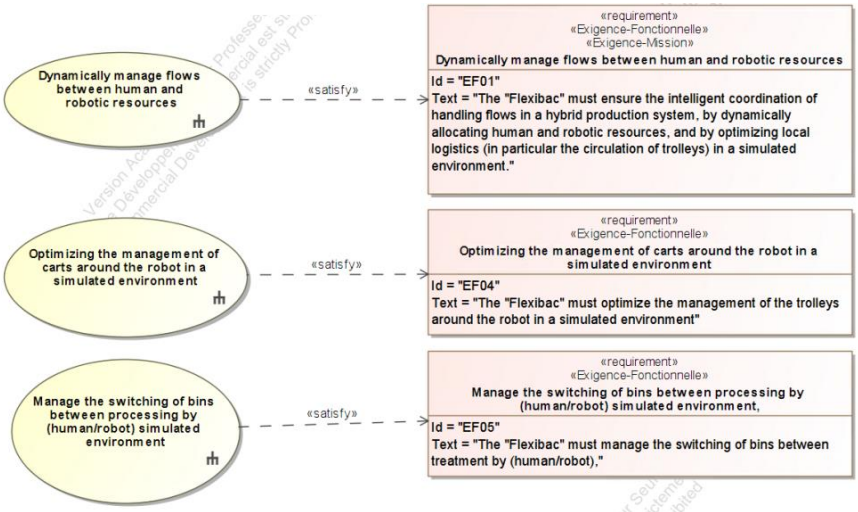

Fig. 5. Functional requirements

#	Id	Stéréotype Appliqué	Nom	Text
1	EC00	Requirement [Class] Exigence-Constrainte [Eler	respect all system constraints	The Flexibac must meet all system constraints (storage capacity, cycle time, cart capacity).
2	EC01	Requirement [Class] Exigence-Constrainte [Eler	Manage online scenarios	The Flexibac must allow online scenarios to be managed).
3	EP00	Requirement [Class] Exigence-Performance	maximize the number of boxes handled by the robot over	The Flexibac should maximize the number of boxes handled by the robot over a 24-hour period.
4	EP01	Requirement [Class] Exigence-Performance	minimize the number of trolley changes	The Flexibac should minimize the number of trolley changes required by operators.

Fig. 6. Other requirements

2.2 System architecture

Activity Diagrams are used to illustrate the flow of control and decisions. The Flexibac simulation system has been implemented following an agent-based modelling composed of different types of agents. The *BoxAgents* are created thanks to the data in the different instances, each *BoxAgent* representing a line in the instances' file. Every *BoxAgent* has 3 properties, that are its identification number, the time of arrival and the destination of the mail it contains. The *SwitchAgent* sends the different bins among the robotic cell and the workers. the *CartAgent* is responsible for changing. Finally, the *FlexibacControlAgent* coordinates the *SwitchAgent* and the *CartAgent*.

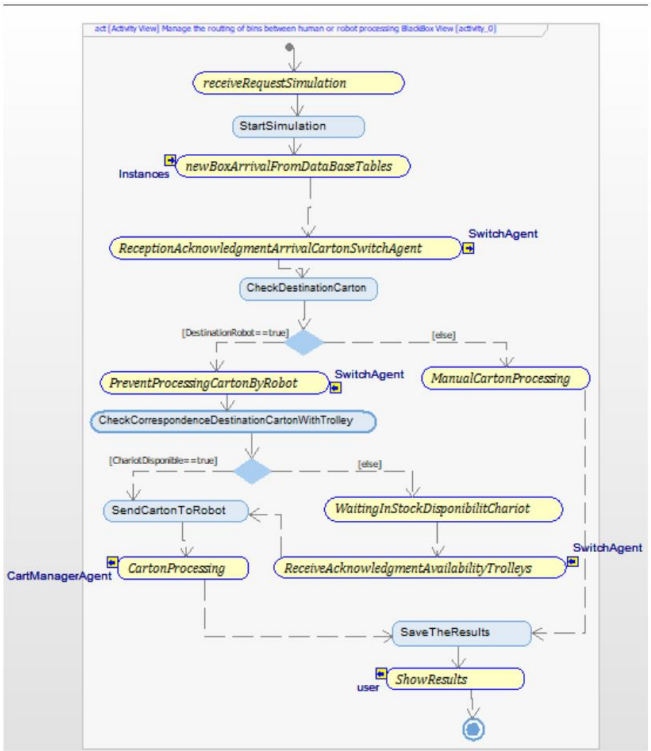

Fig. 7. Activity Diagram of the *Flexibac Control Agent*

These agents are supporting the different use cases presented before. Here also, for the sake of brevity, the use case *Manage the switching of bins between the (human/robot) in the simulated environment* is the only one being developed in the following. The resulting activity diagram is depicted below (Fig. 7) and details the different steps executed by the *Flexibac Control Agent*. This Activity Diagram shows the flow of activities starting with *receiveRequestSimulation* and *StartSimulation*. Key decision points include *CheckDestinationCarton* (leading to either *PreventProcessingCartonByRobot* via *SwitchAgent* or ManualCartonProcessing) and ChariotDisponible (determining if a carton is sent to Robot or wait until the robot is available again). The diagram concludes with *SaveTheResults* and *ShowResults* for the user. This diagram outlines the logical sequence of operations for routing bins.

3 Prototyping of the solution

The solution has been implemented under Anylogic, a software platform used to easily develop multi-agent system, as shown in Fig. 8.

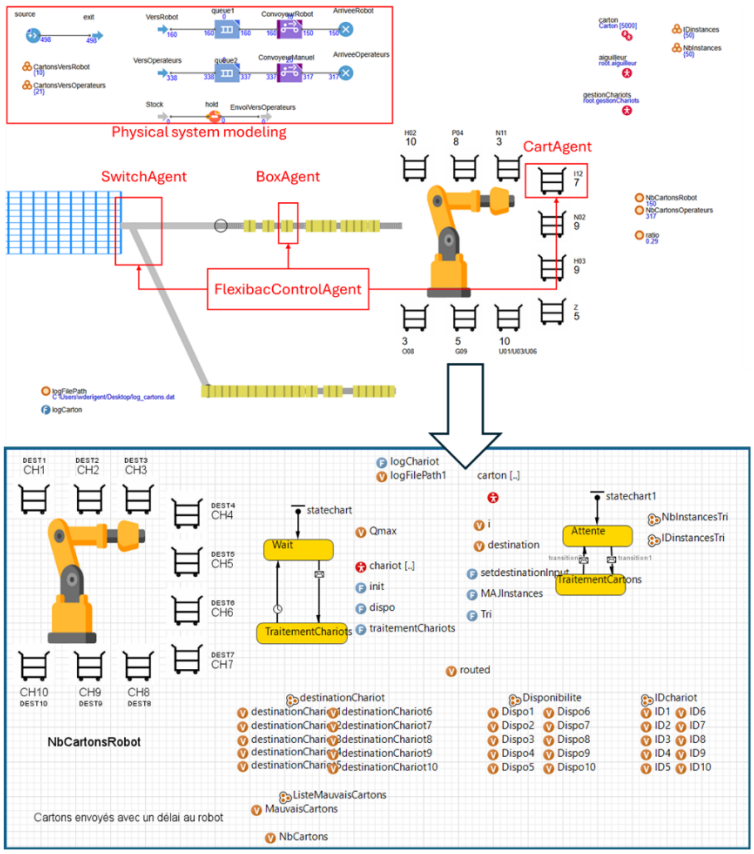

Fig. 8. Physical and logical descriptions of the Flexibac system

Under Anylogic, it is possible to define the physical structure of the simulation (i.e. the number and positions of the agents, the physical elements of the system), as can be seen on the upper part of Fig. 8. This first description is then completed by the logical behavior of the agents made using statecharts coupled with function programming.

4 Test of the solution

Due to lack of time, the proposed solution has been tested on only one instance Nc_5_50_10_1. For this simulation, the results were the following:

- 1291 boxes were treated by the robot,
- 3560 were treated manually.

The ratio of boxes treated automatically on the total of boxes was then: **25.8%**. As the expected ratio from la Poste is 50%, this first result is far from the company's expectations. Moreover, as can be seen, the total of boxes is not equal to the initial 5000 boxes. This is due to a limitation of our proposal. Indeed, when a box is sent to the robot, a cart is available for it. However, by the time it reaches the robot, this cart may have filled up. In our current implementation, for this specific case, the box is considered as lost. This case represents 2.9% of the total mail flow. In our current implementation, carts' destinations are chosen based on their frequency. The files given for the same instance type have been analyzed to order destinations from the most frequent to the less frequent. As a result, when considering instance 5_50_10, the most frequent destination is *H02* and the least frequent is *O03* (Fig. 9). All along the program execution, the *FlexibacControlAgent* maintains this list by considering the boxes already arrived. The new destinations given to carts are always in the 10 most frequent destinations. For the instance 5_50_10, 135 cart changes were necessary.

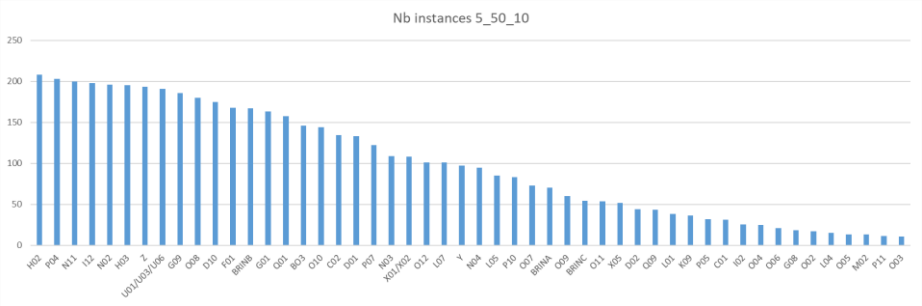

Fig. 9. Destinations ordered by frequency for instance 5_50_10

5 References

[1] J.-Y. Bron, *System Requirements Engineering: A SysML Supported Requirements Engineering Method*. John Wiley & Sons, 2020.

[2] S. Friedenthal, A. Moore, et R. Steiner, *A practical guide to SysML: the systems modeling language*. Morgan Kaufmann, 2014.