Towards an MDE approach of Digital Twin for Software Applications

Asbathou Biyalou-Sama, Matthieu Allon, Cédric Dumoulin, Blazho Nastov, Emmanuel Renaux

Towards an MDE approach of Digital Twin for Software Applications

Asbathou Biyalou-Sama
Univ. Lille, UMR 9189 CRIStAL
IMT Lille Douai
Axellience
Lille, France
asbathou.biyalousama.etu@univ-lille.fr

Cédric Dumoulin
Univ. Lille, UMR 9189 CRIStAL
Lille, France
cedric.dumoulin@univ-lille.fr

Emmanuel Renaux
Univ. Lille, UMR 9189 CRIStAL
IMT Lille Douai
Lille, France
emmanuel.renaux@imt-lille-douai.fr

Blazho Nastov
Axellience
Lille, France
blazho.nastov@axellience.com

Matthieu Allon
Axellience
Lille, France
matthieu.allon@axellience.com

***Abstract*— Digital Twin (DT) refers to a digital representation of a physical entity and its functions, behaviors and rules. Digital Twin (DT) technology, originally, emerges from manufacturing and aerospace sectors. With the advancement of Industry 4.0 and IoT, DT is applied in other domains like cyber-physical systems (CPS), building, and health. This scattering of the DT technology results in a lack of clear and concise definition of what is exactly a DT.In this context, we want to explore what could be a DT for Software Applications (SA). This paper describes the approach that we will follow to identify what could be a DT for Software Applications. On the basis of the features (of a DT) identified in the other domains, we propose ways to define a DT for Software Applications. In a first approach, we propose to use structural and behavioral UML diagrams to instrument the application (by defining software probes), and we visualize the results in a "dashboard". This gives us the "monitoring" features of the DT. From this approach, we want to explore the other features (simulation, prediction, optimization…) that will allow us to propose a more complete digital twin.**

Keywords—Digital Twin, MDE, Software Application

I. INTRODUCTION

Digital twins (DT) have found wide application in areas such as cyber physical systems (CPS) buildings, Iot, health ... A DT is composed of a *physical* part (the CPS, a building ...) and its *digital* counterpart (the twin). In these different domains, the digital twin allows, among other things, to monitor the *physical* part, to perform simulations on the *digital* part, or to interact on the *physical* part.

As we observed these DTs, we wondered ourselves *what a DT of a software application might look like and what such a DT might be used for*...

By Software Application (SA), we mean an application realized in an Object Oriented Language and for which we can obtain representations, even partial, in the form of UML diagrams.

In the absence of a formal definition for DTs, we have identified different points characterizing a DT. We then tried to transpose these characteristics to the domain of SA. One of the difficulties encountered is that in our case, the *'physical'* part of the DT is itself already digital. Despite this difficulty, we propose a first MDE approach in which we describe our DT with the help of three models: a model partially representing the application and in which we can place 'sensors', a model to describe how the data from the sensors are transformed, and a model to describe how to visualize the results. Thereafter, in order to follow the MDE process, we will automatically generate the code corresponding to the DT described in these models.

To quickly validate our approach, and to be able to experiment around the definition of a DT for SA, we propose a framework allowing "to build programmatically" (i.e. without automatic code generation) the DT conform to the models.

This short paper presents the state of progress of our thinking and our future investigations, which, we hope, will allow us to experiment around the notion of DT for SA. These experimentations and the developed tools will allow us, in the future, to propose a more precise definition of a DT for SA.

Section 2 describes the characteristics we have retained when studying DTs in different areas. In section 3, we cite references on which we have based our work. In Section 4, we detail our proposed DT for SA. Section 5 describes the implementation of the framework for the experimentations. In section 6, we describe the challenges that we have identified and that need to be solved in order to complete the definition of a DT for SA. Finally, we conclude and discuss the perspectives in section 7.

II. DIGITAL TWINS : DEFINITION THROUGH DOMAINS AND MAIN USAGES

In CPS, the digital twins consist of a set of models that represent the physical system. These models are digital representations of the physical system. They are able to capture real data from the system [1]. Digital Twins also offer a set of microservices that exploit the data from the system [2], [3]. These microservices provide, among other things, data visualization or system simulation. The Building Information Model (BIM) is considered as the digital twin for building [4]. It is a numeric plan where some values, such as temperature, may change over time. This plan is updated with data gathered from sensors placed on the real building. BIM is used to visualize the information and the behavior of various building components.

The study of these two domains highlights some common characteristics of digital twins:

- Digital representations: whether in CPS or in buildings, digital twins are primarily digital representations integrating various functions of the physical system.

- Data visualization: one of the main purposes of digital twins is to be able to trace and visualize what is happening in the physical system. For example, BIM makes it possible to visualize real information coming from the building.
- Data gathering: Digital twins gather data from the physical system using sensors.
- Data processing: data retrieved by the sensors are raw data. A processing and filtering phase is often necessary to obtain information more relevant for humans.

The digital twins are designed to serve different purposes. The literature reviews in [3], [5], [6] provide a list of these most frequent aims:

- Monitoring: this is the data visualization part.
- Simulation and prediction: simulate the real system is a purpose present in almost all digital twins. According to [7], digital twins are the next generation of simulation techniques. Simulation allows to observe the behaviors and the reactions of CPS or buildings in different scenarios.
- Optimization: a digital twin can also be used to optimize various resources of a system.

III. Existing conceptual methods and proposed architectures

We have studied several approaches for DT. We would like to cite the following two:

Reference [8] proposes to design digital twins for CPS using some dedicated tools: UML/P class diagrams for the Digital Twin Information System, MontiArc models for the physical system. A dedicated language is used to make the interconnection between the two systems. The implementations are generated from the different models.

Reference [9] proposes an approach for designing DT of organizations. It proposes to integrate the different entities of an organization, i.e. processes, people, etc., while also proposing different diagrams.

In terms of the architecture of a digital twin, [2] proposes a four layers architecture: communication, data representation, data computation and microservices. We use parts of this architecture as a start for our work.

These papers show that DT can be designed with an MDE approach, but they do not answer our question « what could be a DT of a SA »?

IV. Our proposal : towards an MDE approach of DT for software applications

In section 2, we have identified some common characteristics of DTs: digital representations, data visualization, data gathering and data processing. Now we propose the equivalent of these concepts for SA. We also introduce the models used to define a DT for a SA.

A. Equivalent concepts

1) Digital representation of software

In CPS and buildings, the digital representation is used to observe various characteristics of the system, either structural or behavioral. In software engineering, UML is used to describe these later characteristics. We use UML and its diagrams to describe the structural part (class diagram) and the behavioral part (activity diagram, collaboration diagram...). We do not describe the whole application, but only the parts that the user wants to observe. To obtain these diagrams and the UML model, we plan to use some reverse engineering tools.

2) Data visualization: monitoring

Data visualization is done in different formats depending on the target applications or the various domains. This can be textual presentation, graphic components or 3D visualization. In our context, we propose the concept of viewers to display data according to the user observation needs. Viewers are not responsible for processing or interpreting the data. The only operations performed are related to the display of data.

3) Data gathering

In CPS or buildings, physical sensors are used to gather real data such as temperature, position of a robot…

At the SA level, we propose to capture data related to the system (temperature, memory usage, CPU speed…) and also application-related data (values of variables or instances, number of instances, values of method parameters, exceptions thrown…). These data are captured in the application using « software probes » that are placed in the code.

4) Data processing

For data processing purpose, we propose the concept of operation boxes. They will be in charge of processing the raw data from the system: filtering, interpretation, extraction of information based on business rules.

B. Concept modeling

The modeling of a DT for a SA is done through the use of the previous concepts in three models (Figure 1-a) that we are going to describe.

1) Probe Model

The probe model (Figure 1-a left) is used to describe both the digital representation of the SA and the placement of software probes within that SA. This model is a partial digital representation, which can be structural or behavioral, of the application. We propose to use UML diagrams to represent the probe models. The probes can be placed on different elements of the SA, such as variables, classes, properties, activities...

In a first approach, we focus on placing probes on methods, in order to capture the input and output parameter values, as well as the system data.

2) Data manipulation and connection model

The data processing and connection model (figure 1-a center) allows both to describe the operation boxes and to connect these boxes to each other, as well as to probes and viewers. This model reuses the probes described in the probe model and the viewers described in the visualization model.

It models the flow of data, from probes to viewers through the various processing boxes.

3) Visualization model

This model (Figure 1-a right) allows the user to describe the set of viewers he wants to use to observe his digital twin. The user has the possibility to use simple viewers like text, image, etc. or to compose them in order to build viewers that

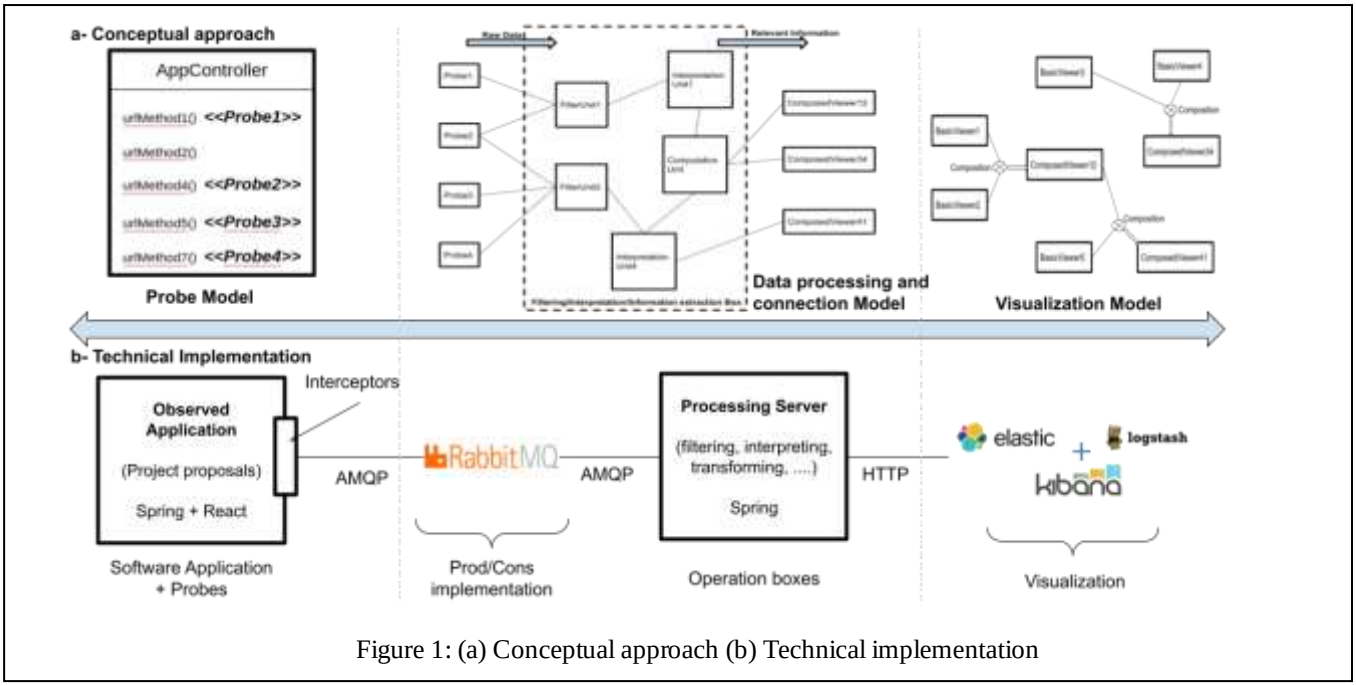

Figure 1: (a) Conceptual approach (b) Technical implementation

are more complex. This model describes the data displayed by the digital twin.

V. Technical Implementation of Our Approach

To validate our approach, we develop a prototype. Its architecture (Figure 1-b) is inspired by the three models: application and probes model, data processing model and visualization model. We add communication between the different parts. The prototype implements the metamodels for the three models, as well as a framework allowing to build a DT for a SA. Proposing a framework to build DT is important, as highlighted in [10]. To be complete, we test our framework on a real SA.

A. Metamodels

For the probe model, we use UML and its diagrams. The probes are designed using stereotypes that can be positioned on any UML element. The metamodels for data processing models and visualization models are realized with EMF, and the corresponding diagrams with Sirius.

B. Model to framework transformations

The code generation of the DT will be possible from the templates. This will be done later, when the models and the framework are stabilized.

C. Framework

The framework provides the basic building block to build DTs. The assembly and configuration of the building blocks will later be generated from the models. The framework uses existing technologies (figure 1-b): Kibana for viewers, a web server (in Spring) to implement the data processing blocks, RabbitMQ to communicate between probes and processing blocks, http requests to communicate between processing blocks and Kibana (via logstash and ElasticSearch).

The notion of software probe is realized thanks to "Spring interceptors". This allows placing probes on method calls without touching the existing code of the application to observe. The downside is that we are currently limited to applications developed with Spring. Other technologies are available to implement probes and will be studied later.

D. Targeted software application

To test the framework, we use a real software application that is in production. It is a web application developed in Spring and React allowing "computer science project proposals" for students in Masters.

VI. Challenges

We have identified several challenges.

A. What data to display?

The first challenge we identified was "what data could be visualized using the DT, and in which forms"? There is of course the basic data such as values of arguments, of variables or instances, CPU time, temperature, memory usage, etc. These data are immediately accessible, and their visualization is straightforward. It is also possible to combine data in order to obtain and display more complex data such as the one related to business processes, execution of activities... We could thus visualize the percentage of users in the different steps of a business process, some statistics on errors, withdrawals, paths followed...

Still by combining data, we can find new data that can hardly be accessible today: for example, we can combine the CPU temperature with the number of user instances in order to check the correct functioning of the processor, or to improve the energy consumption.

B. Improve the prototype

Our second challenge consists in improving the prototype of our approach. To achieve this we need to improve our framework to make it easier to use and to integrate into different types of applications: server, desktop... We need to complete the concepts used in our metamodels, allow the automatic generation of the DT code, and also propose to build the probe model by reversing the real application. We also need to explore other mechanisms for probes to observe other types of software applications.

C. Simulation and optimization

Our current proposal does not allow any simulation. This is our third challenge: what can we propose to perform software application simulation?

In an answer that we propose, we divide the software application into "interconnected blocks". Each block takes input and output data, and performs processing on these data. For example, blocks are classes, components, methods, packages...

Our current DT proposal allows monitoring and recording the inputs and outputs of these blocks. We can then consider different scenarios as in the following examples:

- We modify the behavior of a block and we reinject the recorded data in order to study the new implementation. This is close to unit tests, but here we work on "real" data. Changing the behavior can be done by providing new code, or by using a language to describe the output data in terms of inputs, such as OCL.
- The behavior of a block can also be simulated and modified by replacing the block with a mock. A mock allows to specify output data according to input data. A generic implementation is to record pairs of {input data, output data}, and then provide the output data based on the submitted inputs. With the help of our DT, we can record the inputs and outputs, and thus automatically create a mock of a block. Of course, the number of pairs of {inputs, outputs} we can record is limited.
- Going even further, we can imagine putting an Artificial Intelligence (AI) in the block, and instructing this AI using the input and output data. Once this "AI block" is instructed, we can replace the original block and observe the behavior of the rest of the blocks. Here we do not have the memory limit anymore, but we have to find out how to change the behavior of the block. One possible approach is to instruct the block with modified data.

If we know how to decompose the application into blocks, and reinject modified blocks, we can then perform simulations. We can also study the performance of the blocks in order to optimize the system.

D. Interaction with the application

Another challenge is to propose solutions to enable our DT to interact with the application. For example, the DT would allow changing configuration variables in order to adapt the SA to a new context.

E. Define a DT for a SA

Our last challenge will be to provide a more precise definition of what a DT for an SA is, and what it can bring.

VII. Conclusion & Perspectives

We are at the beginning of our work, and we have a real question we want to answer: what a DT of a software application might look like and what such a DT might be used for. To answer this question, we propose to experiment around the definition of a DT. The study of DTs in other domains allows us to suggest ways to define a DT for a SA. The DT is defined using three models describing the part of the application to be observed and the probes used to gather data, the data transformation, and finally the data visualization. A prototype using this approach is currently under implementation.

Our next work will be to search and deepen the solutions for the identified challenges. The framework will allow us to explore and test these solutions. The first aim of this short paper is to expose the ideas we want to develop and implement to be able to define what is a DT for a SA. The second aim is to initiate discussions.

References

[1] R. Vrabič, J. A. Erkoyuncu, P. Butala, and R. Roy, "Digital twins: Understanding the added value of integrated models for through-life engineering services," *Procedia Manuf.*, vol. 16, pp. 139–146, Jan. 2018, doi: 10.1016/j.promfg.2018.10.167.

[2] K. Y. H. Lim, P. Zheng, and C.-H. Chen, "A state-of-the-art survey of Digital Twin: techniques, engineering product lifecycle management and business innovation perspectives," *J. Intell. Manuf.*, Aug. 2020, doi: 10.1007/s10845-019-01512-w.

[3] C. Boje, A. Guerriero, S. Kubicki, and Y. Rezgui, "Towards a semantic Construction Digital Twin: Directions for future research," *Autom. Constr.*, vol. 114, p. 103179, Jun. 2020, doi: 10.1016/j.autcon.2020.103179.

[4] S. Kaewunruen and N. Xu, "Digital Twin for Sustainability Evaluation of Railway Station Buildings," *Front. Built Environ.*, vol. 0, 2018, doi: 10.3389/fbuil.2018.00077.

[5] M. Liu, S. Fang, H. Dong, and C. Xu, "Review of digital twin about concepts, technologies, and industrial applications," *J. Manuf. Syst.*, vol. 58, pp. 346–361, Jan. 2021, doi: 10.1016/j.jmsy.2020.06.017.

[6] Y. Lu, C. Liu, K. I.-K. Wang, H. Huang, and X. Xu, "Digital Twin-driven smart manufacturing: Connotation, reference model, applications and research issues," *Robot. Comput.-Integr. Manuf.*, vol. 61, p. 101837, Feb. 2020, doi: 10.1016/j.rcim.2019.101837.

[7] S. Boschert and R. Rosen, "Digital Twin—The Simulation Aspect," in *Mechatronic Futures: Challenges and Solutions for Mechatronic Systems and their Designers*, P. Hehenberger and D. Bradley, Eds. Cham: Springer International Publishing, 2016, pp. 59–74. doi: 10.1007/978-3-319-32156-1_5.

[8] J. C. Kirchhof, J. Michael, B. Rumpe, S. Varga, and A. Wortmann, "Model-driven digital twin construction: synthesizing the integration of cyber-physical systems with their information systems," in *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*, New York, NY, USA, Oct. 2020, pp. 90–101. doi: 10.1145/3365438.3410941.

[9] M. Caporuscio, F. Edrisi, M. Hallberg, A. Johannesson, C. Kopf, and D. Perez-Palacin, "Architectural Concerns for Digital Twin of the Organization," in *Software Architecture*, Cham, 2020, pp. 265–280. doi: 10.1007/978-3-030-58923-3_18.

[10] F. Bordeleau, B. Combemale, R. Eramo, M. van den Brand, and M. Wimmer, "Towards Model-Driven Digital Twin Engineering: Current Opportunities and Future Challenges," presented at the ICSMM 2020 - International Conference on Systems Modelling and Management, Jun. 2020. Accessed: Oct. 24, 2020. [Online]. Available: https://hal.inria.fr/hal-02946949