Modeling resilient cyber-physical processes and their composition from digital twins via Markov Decision Processes

(Discussion/Short Paper)

Giuseppe De Giacomo[1], Marco Favorito[1,2], Francesco Leotta[1], Massimo Mecella[1] and Luciana Silo[1]

[1]*Sapienza University of Rome, Via Ariosto, 25, 00185 Rome RM, Italy*
[2]*Bank of Italy*

Abstract

Cyber-physical processes are those processes in which (some) tasks are autonomously enacted by smart objects and have a physical effect. They are interesting in current Internet-of-Things (IoT) scenarios, in which the resilience of the overall process is crucial. Digital twins, widespread in smart manufacturing but also in many other novel scenarios, can be used as building blocks of cyber-physical processes. In this work, we focus on the orchestration of Digital Twins using an AI technique such as Markov Decision Processes (MDPs). We formalize stochastic composition of processes as LTL_f goals, we present a proof-of-concept implementation and exemplify in an Industry 4.0 scenario.

Keywords

Cyber-physical process, Service composition, Digital twins, Smart Manufacturing

1. Introduction

Nowadays, it is common in most organizations to organize business in terms of composing processes. Business Process Management (BPM) comprises the methods, techniques and technologies able to translate business process models into computer-supported activities, relinquishing routine management and control tasks from the organizational agents. *Resilience* is instead a property associated with an organization's capacity to continue its mission despite disruption, through mindfulness, resourceful agility, elastic infrastructures and recoverability. In this paper, we investigate the concept of resilience for BPM systems, in particular when such systems are seen as a composition of Digital Twins (DTs). A DT is a virtual representation that serves as the real-time digital counterpart of something that exists in the non-virtual (i.e., physical) world

PMAI@IJCAI22: International IJCAI Workshop on Process Management in the AI era, July 23, 2022, Vienna, Austria
degiacomo@diag.uniroma1.it (G. De Giacomo); favorito@diag.uniroma1.it (M. Favorito); leotta@diag.uniroma1.it (F. Leotta); mecella@diag.uniroma1.it (M. Mecella); silo@diag.uniroma1.it (L. Silo)
http://www.diag.uniroma1.it//degiacom/ (G. De Giacomo); https://marcofavorito.me/ (M. Favorito); http://www.diag.uniroma1.it/leotta/ (F. Leotta); https://sites.google.com/dis.uniroma1.it/mecellone/teaching (M. Mecella); https://luusi.github.io/ (L. Silo)
0000-0001-9680-7658 (G. De Giacomo); 0000-0001-9566-3576 (M. Favorito); 0000-0001-9216-8502 (F. Leotta); 0000-0002-9730-8882 (M. Mecella); 0000-0001-7250-8979 (L. Silo)

(e.g., a production system, a machine, or an organization). Notably, a process composed of DTs is a particular case of a so called *cyber-physical process* (CPP) [1]. Cyber-physical systems are characterized indeed by the presence of heterogeneous devices with different architectures, computing and communication capabilities; a dedicated CPP coordinates the working of agents during the steps of the process. Modern plant information systems and industrial machines may natively come out with their DTs[1]; in other cases DTs are obtained by wrapping actors that are already in place.

Inspired by the research on automatic orchestration and composition of software artifacts, such as Web services, in [2] it has been argued that a possible approach for addressing CPPs and developing novel automation techniques in smart manufacturing is the modeling of DT services and data as software artifacts, and that the principles and techniques for composition of artifacts in the digital world can be leveraged to improve automation in the physical one.

Consistently with the Roman model for service composition [3], they consider smart manufacturing scenarios where DTs of physical systems provide stateful services wrapping the functionalities of machines and tasks of human operators. Nevertheless, an inherent limitation of approaches based on the classical Roman model is the assumption that the available services, i.e., the services that can be used to realize the target service, behave *deterministically*. This assumption is often unrealistic, because in practice the underlying physical system modeled as a set of services might show non-deterministic behaviour due to the complexity of the domain, or due to uncertainty about the dynamics of such system. In these cases, the deterministic service model is not expressive enough to capture crucial facets of the system being modelled, i.e., the model is not properly enabling resilience. Moreover, the above-mentioned techniques work only when the target is fully realizable with no middle ground. Again, this is not properly addressing resilience, and instead it would be preferred a technique that, rather than returning no answer, returns the "best-possible" solution under the actual circumstances, the work [4] contributes in this direction.[2]

This paper illustrates a novel approach to model CPPs and composes DTs, by synthesizing them in a resilient way, extending [4] in such a way as to capture the non-deterministic behaviour of the available services.

2. Implementation and Case Study

To illustrate the proposed technique in a real scenario, we applied it to the manufacturing process of an electric motor. The main components of an electrical motor are the stator, the rotor and, in the case of alternate current motors with direct current power (e.g., electric cars) an inverter. These three components are built or retrieved in any order (no precedence constraints between these tasks) and then eventually assembled to build a motor (*alternate succession* constraint between Build/Retrieve tasks and the Assemble Motor task). After the motor is assembled, a running in test must be performed (*alternate succession* constraint between the Assemble Motor

[1] http://www.forbes.com/sites/bernardmarr/2017/03/06/what-is-digital-twin-technology-and-why-is-it-so-important

[2] This paper provides a solution technique that coincides with the exact solution if a composition exists; otherwise it provides an approximate solution that maximizes the expected sum of values of the target service's requests

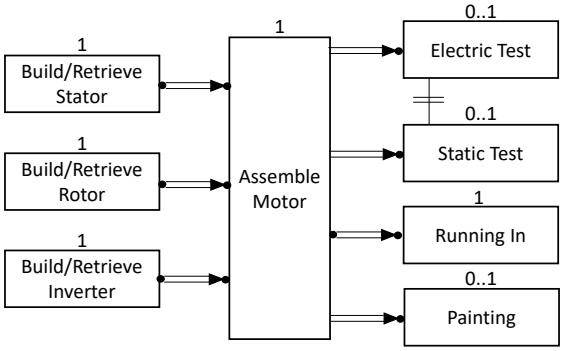

Figure 1: The electric motor manufacturing process represented using DECLARE.

task and the Running In task), and at most one (*not coexistence* constraint) between an electric test and a full static test (the latter comprises the former). In addition, optionally, the motor can be painted. The Painting, Electric Test, Static Test tasks optionally follow the Assemble Motor task (*alternate precedence* constraints). The process depicts the manufacturing tasks involved in a production of a *single motor* as indicated by the *existence constraints*. In order to apply our tool, we first translate the DECLARE manufacturing process in Figure 1 into the correspondent LTL_f formula, which is then turned into the correspondent DFA by using the `Lydia` tool [5][3].

The actors involved in the manufacturing process may break and are modeled as MDP with a READY state and a BROKEN state, or may not break and are modeled as MDP with only a READY state (in this case we have $p_i^b = 0$). Actors include machines, humans, suppliers and information systems. Each edge is labeled indicating the task, the probability of transition and the reward vector structured in a multi-objective fashion. We imagine to have two objectives we want to maximize, the economic saving and the overall quality of the product. These two components of the vector cost must combined by using a priority preserving function. In our example, we use negative rewards (i.e., costs) as they suit better the specific case. Probabilities and costs are *continuously computed* by the DT by applying, for example, techniques to estimate the Remaining Useful Life (RUL) in the case of machines.

In the READY state, a family of operations, denoted with [OP] can be executed. The execution of an operation has an economic cost denoted by $c_i^{[Op]}$ and a quality cost $q_i^{[Op]}$ on the product being manufactured. In some cases, the execution of [OP] may take the actor i to the BROKEN state with probability p_i^b. In this case an additional cost c_i^b, due to the necessity of disposing an incomplete piece is paid. In order to take the actor back to the READY state, a RESTORE[OP] task must be executed on the actor, which have a cost c_i^r depending on the actual conditions of the actor. Noteworthy, RESTORE[OP] operation is a τ action as it does not leave any trace and is implicit on the process depicted in Figure 1. In order for an actor in the BROKEN state to be used for building, it must be fixed, which in some cases is the most convenient thing to do.

More in detail: the stator and the rotor can be either built by machines that might break, or retrieved by the warehouse (for the rotor we have two different machines); the inverter is provided by a supplier, thus we only have a warehouse; the assembling and the painting can be

[3]The source code of Lydia can be found at https://github.com/whitemech/lydia

done either by a machine, that cannot break or by a human (the human may become convenient when the wearing status of the machine is very high); the running in can be performed by two different machines; the Electric Test and the Static Test are performed by a human engineer.

The warehouse has an important role in our case study. From one point of view, it allows to cope with the lack of a specific manufacturing machine (as in the case of the inverter component). On the other side, it takes into account cases where a component is build by a machine but then not employed because the process instance cannot finish successfully. This can happen when an actor selected to perform an operation (e.g., building the stator) breaks in reality. In this case, the process fails and we must reschedule, but already built component are placed in the warehouse. In this sense, the warehouse MDP could have, at a specific iteration, no available operation if it empty. On the other hand, if a product is available the economic cost will be zero, whereas the quality cost depends on the quality of the already available component.

The approach proposed has been implemented in freely available Python tool[4], the GitHub repository contains a Python script to reproduce the case study[5]. As illustrative examples, the script analyzes how the following factors influence the choice of actors and operations: priorities imposed between cost measures, probability of failure of certain actors, quality versus economic cost due to restore in the case of Broken actors, quality cost when multiple actors declare the same economic cost for the same operation.

Acknowledgments

This work is partially supported by the ERC Advanced Grant WhiteMech (No. 834228), by the EU ICT-48 2020 project TAILOR (No. 952215), by the PRIN project RIPER (No. 20203FFYLK), and by the JPMorgan AI Faculty Research Award "Resilience-based Generalized Planning and Strategic Reasoning".

References

[1] F. Leotta, A. Marrella, M. Mecella, Iot for bpmers. challenges, case studies and successful applications, in: Business Process Management - 17th International Conference, BPM, volume 11675 of *LNCS*, Springer, 2019, pp. 16–22.

[2] T. Catarci, D. Firmani, F. Leotta, F. Mandreoli, M. Mecella, F. Sapio, A conceptual architecture and model for smart manufacturing relying on service-based digital twins, in: IEEE international conference on web services (ICWS), IEEE, 2019, pp. 229–236.

[3] G. De Giacomo, M. Mecella, F. Patrizi, Automated service composition based on behaviors: The Roman model, in: Web services foundations, 2014.

[4] R. I. Brafman, G. De Giacomo, M. Mecella, S. Sardina, Service composition in stochastic settings, in: Conference of the Italian Association for Artificial Intelligence, Springer, 2017, pp. 159–171.

[5] G. De Giacomo, M. Favorito, Compositional approach to translate ltlf/ldlf into deterministic finite automata, in: ICAPS, AAAI Press, 2021, pp. 122–130.

[4] See https://github.com/luusi/stochastic-service-composition-with-ltlf-goals

[5] See the `docs/notebooks` folder in the GitHub repository.