Robust digital twin compositions for Industry 4.0 smart manufacturing systems

Davy Preuveneers
imec-DistriNet, KU Leuven
Leuven, Belgium
davy.preuveneers@cs.kuleuven.be

Wouter Joosen
imec-DistriNet, KU Leuven
Leuven, Belgium
wouter.joosen@cs.kuleuven.be

Elisabeth Ilie-Zudor
MTA SZTAKI
Budapest, Hungary
ilie@sztaki.mta.hu

***Abstract*—Industry 4.0 is an emerging business paradigm that is reaping the benefits of enabling technologies driving intelligent systems and environments. By acquiring, processing and acting upon various kinds of relevant context information, smart automated manufacturing systems can make well-informed decisions to adapt and optimize their production processes at runtime. To manage this complexity, the manufacturing world is proposing the 'Digital Twin' model to represent physical products in the real space and their virtual counterparts in the virtual space, with data connections to tie the virtual and real products together for an augmented view of the manufacturing workflow. The benefits of such representations are simplified process simulations and efficiency optimizations, predictions, early warnings, etc. However, the robustness and fidelity of digital twins are a critical concern, especially when independently developed production systems and corresponding digital twins interfere with one another in a manufacturing workflow and jeopardize the proper behavior of production systems. We therefore evaluate the addition of safeguards to digital twins for smart cyber-physical production systems (CPPS) in an Industry 4.0 manufacturing workflow in the form of feature toggles that are managed at runtime by software circuit breakers. Our evaluation shows how these improvements can increase the robustness of interacting digital twins by avoiding local errors from cascading through the distributed production or manufacturing workflow.**

***Keywords*-Cyber-Physical Production Systems, Digital Twin, Circuit Breaker, Feature Toggle**

I. Introduction

Over the past two decades, the scientific community has delivered a variety of middleware frameworks [1] that allow software engineers to make abstraction of the basic functionalities and interfaces of sensors and actuators of distributed context-aware systems. Beyond typical application areas, such as smart homes and offices, these enabling technologies sparked a digital transformation in the manufacturing world as well, often referred to as the 4^{th} Generation Industrial Revolution (Industry 4.0) [2], [3] or the Factory of the Future (FoF) [4]. These paradigm shifts envision smart factories where the Internet of Things (IoT) and Cyber-Physical Systems (CPS)-enabled manufacturing [5] provide the foundations for creating smart products through smart processes and procedures. Large factories connect hundreds – if not thousands – of sensors and devices, not only within the plant, but also with other factories and the outside world. Smart products will plan, control and optimize their own production process with minimal human intervention by harnessing ongoing developments in sensor technology, machine-to-machine communication [6], big data analytics [7] and machine learning [8], [9].

The purpose of this digital transformation is to enhance the transparency of the production process across the organizational boundaries of the manufacturing enterprise. Enhanced access to Industrial IoT (IIoT) [3] data will support business applications on any device, any time, from any location. In turn, the data-intensive nature of smart production systems will enable timely, accurate and detailed log trails resulting in a real-time augmented view on many systems and activities in a way that was not previously possible. A consequence is that the production floor has become an inherently complex intelligent environment, as the digital and physical worlds are heavily intertwined.

The interconnectivity not only plays a crucial role within the boundaries of the manufacturing enterprise to optimize production processes, but also beyond those boundaries. As a consequence, manufacturing is inherently a distributed control process. Traditionally, multiple controlling systems within the enterprise operated in a hierarchical mode with supervisory controllers to schedule, coordinate and optimize production processes, as illustrated in Fig. 1. To account for more flexibility and the ability to adapt to internal and external changes, there has been a shift towards a more a

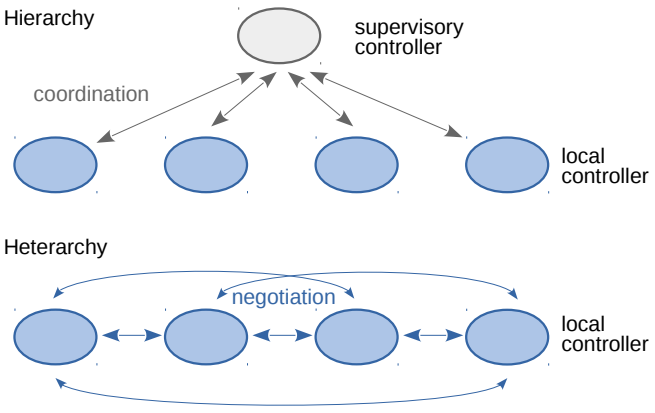

Figure 1. Hierarchical versus heterarchical distributed control systems for manufacturing

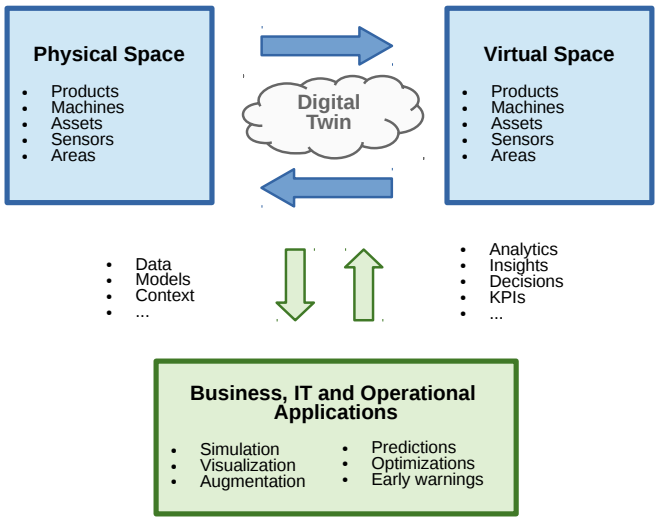

Figure 2. High-level representation of a digital twin in Industry 4.0 applications

heterarchical manner of operation where such supervisory controllers are not present. In the world of networked production with multiple stakeholders in complex manufacturing and production workflows, the overall control process will be inherently a hybrid combination of hierarchical and heterarchical cooperating processes. It is clear that an error or fault in one production process may interfere with processes elsewhere, possibly without a supervisory controller able to monitor or intervene in the negotiation or coordination between these processes.

To manage the ever increasing complexity of larger and larger automated production networks, the manufacturing world proposed the Digital Twin [10], [11] model to represent physical products in the real space and their virtual counterparts in the virtual space, with data connections to tie the virtual and real products together. Matthew Mikell and Jen Clark[1] of IBM defined the concept as follows:

> *The digital twin is the virtual representation of a physical object or system across its life-cycle. It uses real-time data and other sources to enable learning, reasoning, and dynamically recalibrating for improved decision making.*

As such, the Digital Twin concept enables an abstraction layer that simplifies simulation, augmentation, visualization and prediction of many Industry 4.0 processes, as depicted in Fig. 2.

During the development of such sophisticated digital twins, ensuring and maintaining consistency among the physical and virtual spaces is not straightforward when each of these spaces is driven by their own application logic and business rules. The problem of local errors may be further aggravated when multiple smart systems and processes in a manufacturing workflow are developed independently, but react to similar contexts, or create circumstances where multiple digital twins focusing on different objectives interfere with one another. Guaranteeing the reliability and robustness of digital twins - especially under unusual conditions that stress the assumptions of the individual system - remains a non-trivial challenge. Robust intelligent systems [12] always ensure acceptable performance, and this both under ordinary conditions and those that the developers and operators did not anticipate.

The main contribution of this work – building upon our previous work [13] – is (1) the addition of safeguards in the design and implementation of Industry 4.0 digital twins by means of feature toggles, and (2) improving their reliability at runtime by managing these feature toggles using software circuit breakers to avoid local errors from cascading through the distributed manufacturing workflow.

After reviewing relevant related work in section II and a motivating scenario in section III, we discuss our approach for safeguarding digital twins in smart factories in section IV. In section V, we illustrate and evaluate the approach on a prototypical Industry 4.0 workflow. We conclude in section VI summarizing our main insights and we identify interesting topics for further research.

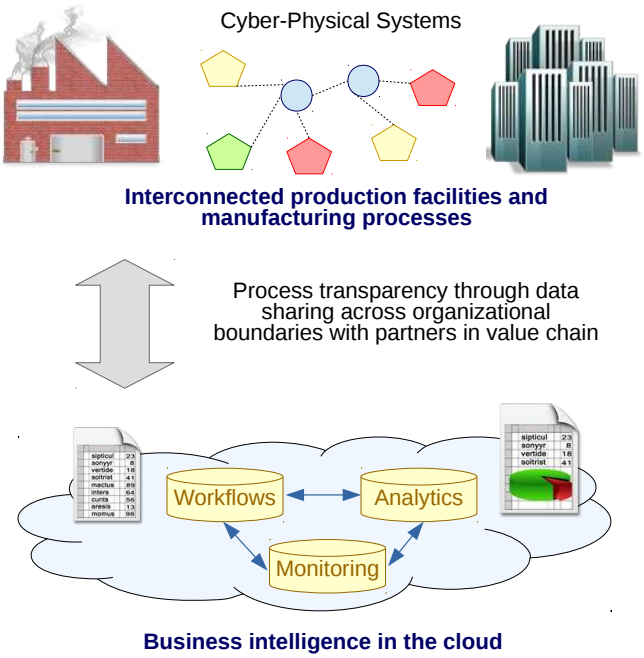

Figure 3. Networked production and concerted manufacturing processes

II. Background and Related Work

This section provides an overview of relevant concepts and related work in the area of digital twins, feature toggles and circuit breakers.

[1] https://www.ibm.com/blogs/internet-of-things/iot-cheat-sheet-digital-twin/

A. Industry 4.0 networked production and digital twins

With networked production as a key feature of Industry 4.0, people, machines and business processes will interact with one another to enable personalized products through flexible, resource-efficient and cost-effective manufacturing. Interconnected systems will be linked to cloud services for remote monitoring and data analytics to optimize production plans, enable proactive maintenance, and respond quicker to continuously changing customer requirements. The factory of the future will leverage data-accessing and data-processing services available on the internet to support data-intensive business processes and time-critical applications, as depicted in Fig. 3.

The notion of a digital twin [10], [11], [14] is not clearly defined. For some, it refers to a virtual representation of a physical object or system that captures all manufacturing defects, including the wear and tear of the object while in use [15], [16]. For others, it reflects a digital and historical profile of a product or process, based on massive and real-time measurements from connected sensors that can help engineers understand not only how products are used by real customers in real-time, but also how they will perform in future scenarios and changing circumstances [17]. The augmented views created by digital twins help optimize business processes and improve customer satisfaction.

While digital twins can support the entire product lifecycle, including product design, manufacturing, and service, the reliability of digital twins is a concern that is often overlooked. Indeed, the implicit assumption is that digital twins themselves, both in isolation and especially in complex compositions of distributed manufacturing workflows, do not fail. This is the challenge we aim to address in this work using some software engineering best practices, including feature toggles and circuit breakers.

B. Orchestration and choreography of distributed manufacturing processes and workflows

Orchestration and choreography [18] are two well-known composition strategies in the domain of service-oriented architectures to build business processes from multiple individiual web services. These concepts are to some extend related to the hierarchical and heterarchical modus operandi of manufacturing control systems. Orchestration, on the one hand, assumes control from one stakeholder's perspective when interacting with internal and external web services. Choreography, on the other hand, reflects a more collaborative interaction allowing each involved stakeholder to describe its part in the interaction.

Schulte et al. [19] propose virtual factories that bridge service-oriented computing and service-based workflows with the Internet of Things. Similar to the typical roles of a service requester, a service provider and a service registry in a service-oriented architecture, they propose service providers, service consumers and virtual factory brokers as essential roles of the virtual factory. The latter role is in charge of controlling and governing the virtual factory, and uses services to model manufacturing processes and assemble products based on outputs of factories from different business partners. The plug-and-play nature of virtual factories helps enterprises execute cross-organizational manufacturing processes as though they were being carried out within a single company.

In [20], Mangler et al. prosented a Domain Specific Language (DSL) underpinning the Cloud Process Execution Engine (CPEE)[2], a modular simple and lightweight service oriented workflow engine. The DSL is an alternative to the XML-based BPEL language, while the engine supports multiple execution languages (such as BPEL, YAWL, BPMN), interaction protocols (such as SOAP, REST, XMPP) and execution shaping to support ad-hoc changes and custom behavior at runtime. CPEE has been used in the frame of manufacturing applications.

More recently, Velasquez [21] surveyed the state-of-the-art on orchestration beyond services in cloud computing, but also addressing the Internet of Everything and fog computing paradigms. The work of Thramboulidis et al. [22] also falls into that category. They propose an IoT-based framework for manufacturing systems that allows manufacturing plant processes to be defined as compositions of primitive cyber-physical microservices adopting either the orchestration or the choreography pattern.

C. Feature toggles

Feature toggles [23] have been initially proposed in the frame of continuous software delivery to incrementally integrate new features into existing applications or to test bug fixes. Feature toggles enable rapid software releases while reducing the probability of integration conflicts. Even after the final software release, these feature toggles can be used to only enable certain features for specific users or to quickly disable a feature that is misbehaving. A feature toggle is typically a variable that is used in a conditional statement to enable or disable a piece of code for testing or release purposes. In the past, such feature toggles were used at compile time to exclude certain features from the application binary. Modern feature toggles, however, allow for switching them at runtime without recompilation of the software.

Martin Fowler[3] describes several types of feature toggles that mainly differ in dynamism and longevity. These feature toggles include *release toggles* to disable incomplete or untested codepaths in production systems, *experiment toggles* for multivariate user testing of different codepaths, and *ops toggles* to control operational aspects of a system's behavior.

Our research leverages feature toggles to characterize the behavior of digital twins at runtime, both under normal and

[2]http://cpee.org

[3]https://martinfowler.com/articles/feature-toggles.html

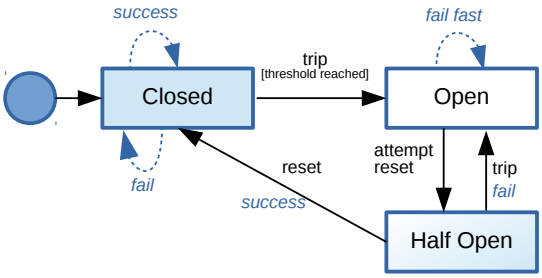

Figure 4. State transitions of a basic circuit breaker

exceptional circumstances. More specifically, certain feature toggles will deactivate automated actions that may negatively influence other digital twins.

D. Circuit breakers

Software-based circuit breakers [24] work in a similar fashion to the electrical ones that toggle a magnetic switch to protect the home automation appliances from excess current on the power lines. In such failure scenarios, the circuit breaker trips from the closed into the open state to break the flow of the current to the home automation appliance. This way, it protects the appliance from breaking down completely.

The state transitions of a circuit breaker are depicted in Fig. 4. Under normal operations, the circuit breaker is closed, allowing the digital twin to operate normally. If an error is detected, the circuit breaker trips and switches from the closed into the open state. As a result, it disables the underlying functionality of the digital twin from being executed, and the corresponding feature toggle from being switched. This will be illustrated in the next two sections with a motivating example.

III. Motivating use case of Industry 4.0 networked production

Our motivating networked production use case is an orchestration of production systems and services – from the sensor to the business level within the enterprise – with information sharing across the organizational boundaries of the company, possibly with other partners in the production workflow. Indeed, in an Industry 4.0 networked production workflow, multiple application and business processes interact with one another, as defined in Business Process Model and Notation (BPMN) in Fig. 5:

- **Vertical integration scenario**: Self-guided sensor and actuator networks control production and diagnostic activities in real-time. Cyber-physical systems deliver decision critical information to optimize production plans, reduce costs and improve efficiency.
- **Horizontal integration scenario**: Machines receive information about the production process from systems – including those of business partners in the value chain – to enable individualized production.

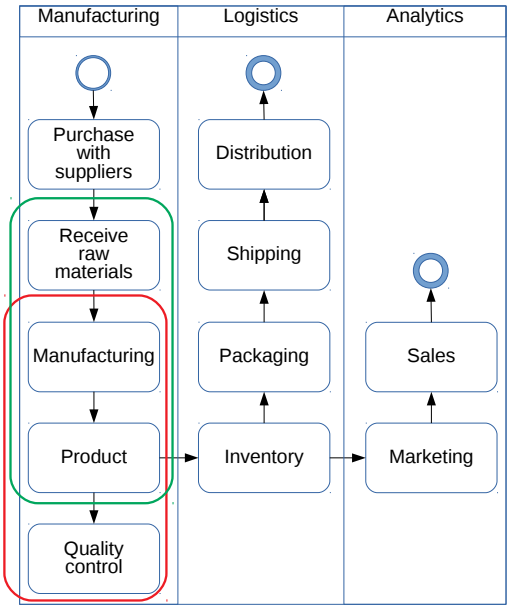

Figure 5. BPMN workflow of concerted processes in networked production

Zooming in on the manufacturing process in Fig. 5, assume one digital twin (marked in green) uses simulations to reduce cost and optimize the individualized production plan in terms of available raw materials and suppliers. Assume a second digital twin (marked in red) uses various product and environmental parameters and machine learning techniques for quality control and assurance. The goal of the second digital twin is to understand whether the cause of a faulty product is linked to, for example, the supplier of certain components or raw materials, or is rather due to specific temperature and humidity levels during the production process. By ingesting various streams of data, the digital twin can help establish correlations with previous faulty products, and learn patterns to predict whether the quality of the product (or a batch of products) will be subpar.

In isolation, both digital twins may operate without problems. However, due to the implicit dependency, errors may still cascade horizontally throughout the manufacturing system without the proper safeguards in place. This will be demonstrated in the next section with local and distributed failure scenarios.

IV. Safeguarding digital twins

In this section, we discuss our approach for safeguarding digital twins in smart factories.

A. Implementing digital twins

With the increasing amounts of data, manufacturing enterprises are exploring big data [25] and analytics technology [26] to (1) identify relevant contexts, (2) transform that

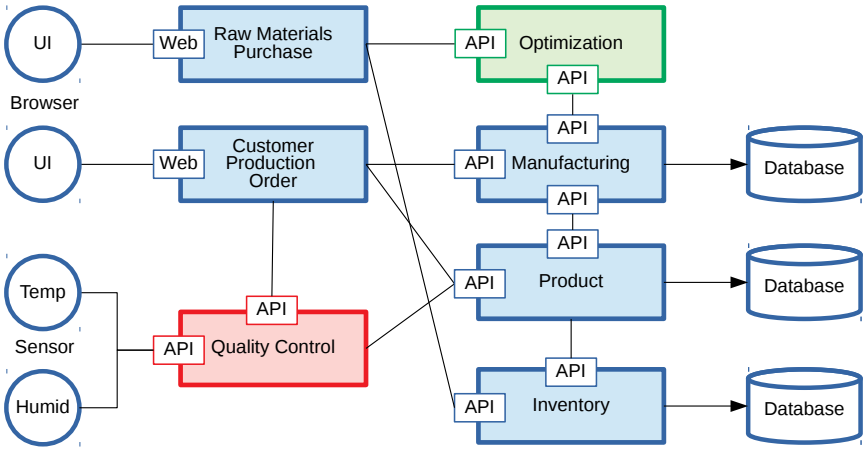

Figure 6. High-level decomposition of the virtual space of a microservice-based manufacturing workflow

data to identify useful patterns and deviations from patterns in real-time. Our proof-of-concept digital twin implementations adopt a microservice architecture [27], [28]. It offers more flexibility to build data intensive applications, when compared to monolithic service architectures. Key features of microservices are that they (1) only provide a limited and clearly defined set of functions, (2) are developed independently in the most appropriate programming language, (3) are stateless and manage their own database, and (4) all run autonomously in their own process.

With microservices, digital twins are developed as compositions of smart data processing endpoints and dumb pipes. The low coupling and high cohesion of such distributed choreographies of independent microservices simplify the introduction of new digital twin functionalities in the data processing workflow, and they allow for each microservice to be developed, deployed, modified and scaled out independently.

Fig. 6 gives a high-level overview of various microservices involved in the manufacturing workflow for individualized production, with the two digital twins marked in green and red, as in Fig. 5.

The behavior of each digital twin can be changed with a software feature toggle, as illustrated below:

optimization {*off, alarm, auto*}: auto
quality_control {*off, alarm, auto*}: alarm

In the *off* state, the feature toggle indicates the digital twin is inactive. When configured in the *alarm* state, the digital twin is active, but any trigger (e.g. occurrence of an error) creates an alarm for a human operator, rather than a fully automated response or mitigation effect without a human in the loop. The latter configuration is achieved with the *auto* state of the feature toggle. The initial default configuration of the feature toggle is also indicated per digital twin.

Each digital twin offers a RESTful interface that can be used by other microservices or digital twins to toggle the state at runtime. In a fully automated configuration of both digital twins, the *Quality Control* digital twin will indirectly trigger the *Optimization* digital twin.

B. Local and distributed failure scenarios

It is clear that whenever bad raw materials are found as the cause of faulty products in the quality assurance stage, this local error will influence the optimization process responsible for individualized production. Whenever raw materials are flagged as bad, they should not be used in future individualized production processes. However, the machine learning algorithms used for quality assurance are not always 100% accurate and may misclassify the cause of a faulty product. If a production flaw is due to a cooling problem during manufacturing (i.e. an environmental parameter), while the classifier mistakenly identifies a raw materials supplier as the cause (i.e. a product parameter), then without failsafes a local error may cascade through the distributed production or manufacturing workflow due to the interactions at the level of product parameters (see Fig. 7).

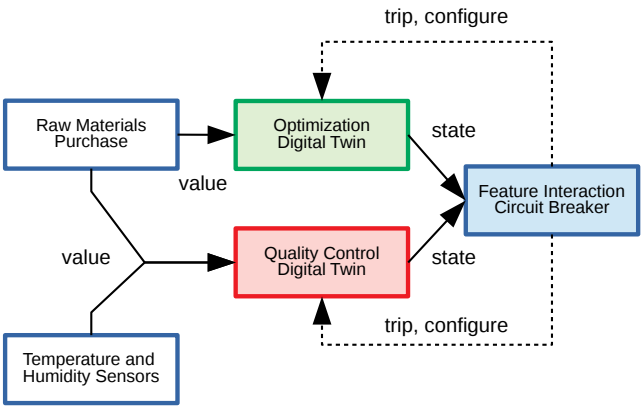

Figure 7. Interfering digital twins with undesired feature interactions

```java
private enum State { OPEN, HALF_OPEN, CLOSED }
private int backoffPeriod = 30000;
private State state = State.CLOSED;

public synchronized void reset() {
   state = State.CLOSED;
}

public synchronized void trip() {
   state = State.OPEN;
   new Timer().schedule(new TimerTask() {
      public void run() {
         state = State.HALF_OPEN;
      }
   }, backoffPeriod);
}

public synchronized void execute() {
   if (state == State.OPEN)
      return;
   try {
      // Implement quality assurance with ML
      // classification based on environmental
      // and raw material parameters
      evalProductionQuality();
      // Quality assurance succeeded, so reset
      // circuit breaker
      reset();
   }
   catch (Exception e) {
      // Quality assurance did not pass, so trip
      // circuit breaker
      trip();
   }
}
```

Figure 8. Java-based implementation of a software circuit breaker wrapping the quality control digital twin

A single digital twin can simply fail due to sensor/actuator connectivity problems. However, when inter-connecting all manufacturing systems and processes, the digital twins can conflict with one another as they depend on the same input parameters. We therefore introduce software circuit breakers into the design of our digital twins to improve the robustness of manufacturing workflows. Fig. 7 depicts a feature interaction circuit breaker that under normal conditions allow the *Quality Control* digital twin to have subpar raw materials to be excluded for further use, and subsequently trigger the *Optimization* digital twin to update the manufacturing workflow. Should the former be erroneous, then the *Feature Interaction* circuit breaker should block the latter from being triggered.

We use a software variant of circuit breakers to protect digital twins and manufacturing systems from malfunctions. We group different kinds of digital twin behaviors per feature toggle, and encapsulate their function in a circuit breaker.

After a local error has been detected, the software circuit breaker is able to reinstate the digital twin functionality after a period of time (i.e. a back-off period). To do so, the circuit breaker automatically shifts from the open into the half-open state and if the next execution of the digital twin succeeds, the circuit breaker resets to the closed state. If the next execution fails again, then the circuit breaker reverts back from the half-open to the open state until the next timeout of the back-off period occurs. Fig. 8 provides a high-level Java implementation with the two functionalities a typical circuit breaker has to offer, i.e. the trip() and reset() methods,

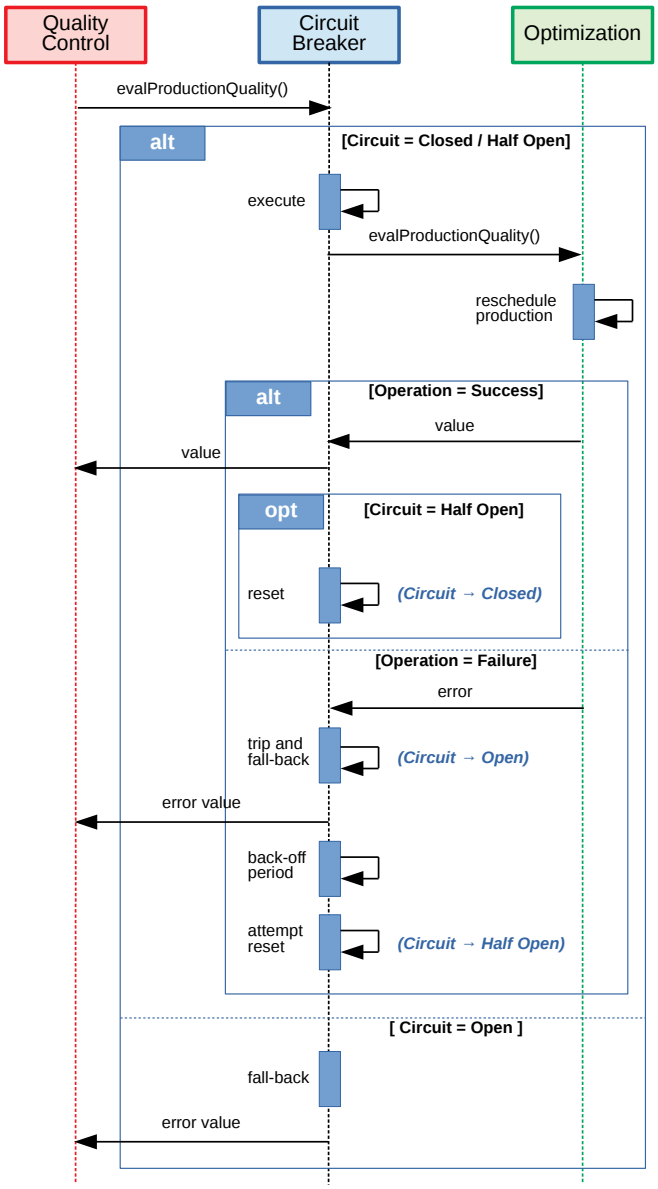

Figure 9. Interaction diagram of two interfering digital twins protected by a software circuit breaker

with a back-off period of 30000 msec to attempt a reset whenever the circuit breaker changes to the State.OPEN state. A more elaborate interaction is depicted in Fig. 9. Whenever the circuit breaker is in the State.OPEN state, the *Quality Control* digital twin does not trigger an update or modification in the *Optimization* digital twin.

The full implementation of the *Feature Interaction* circuit breaker is beyond the scope of this paper, but relies on (1) a data flow diagram that represents all information streams in the distributed manufacturing workflow, and (2) a dependency graph of which components a digital twin can manipulate directly or indirectly. Both combined, the

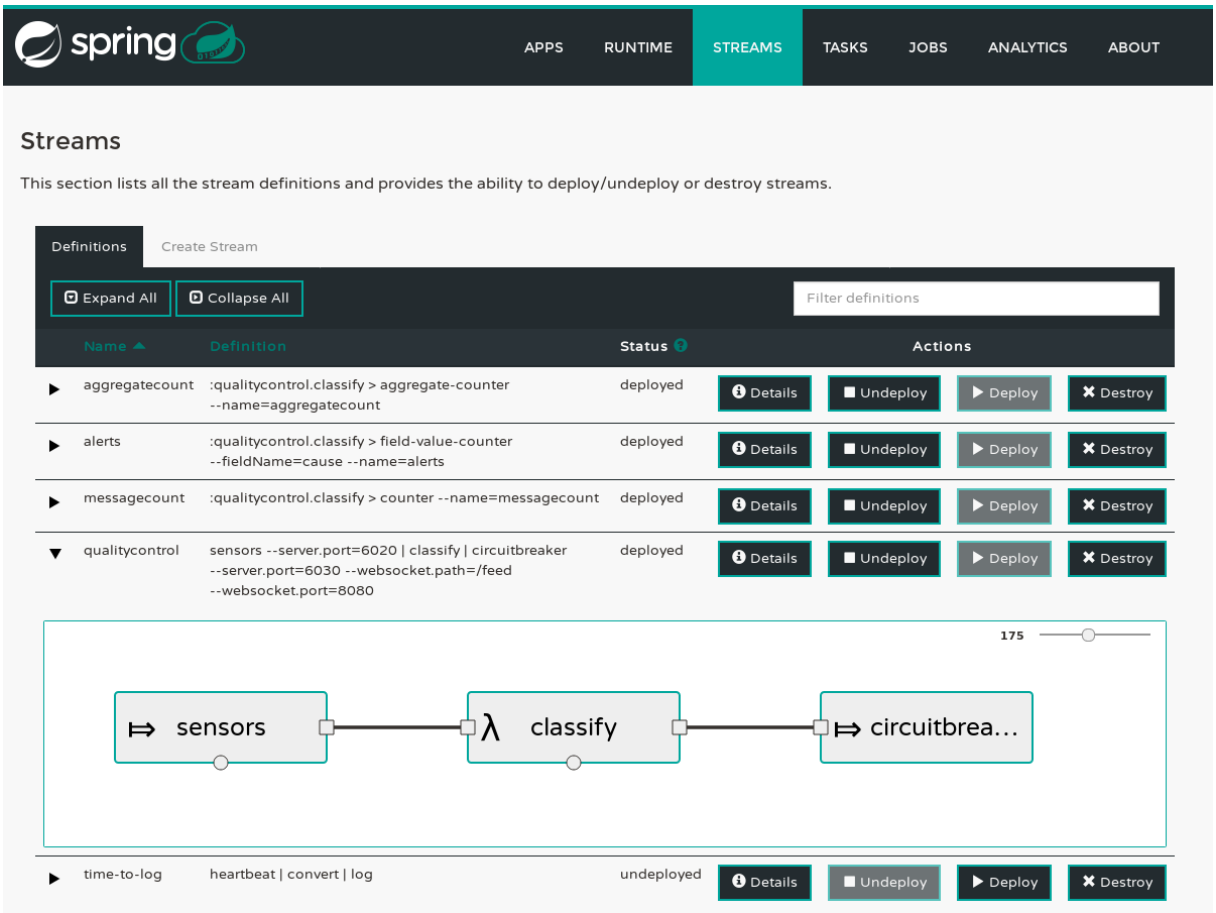

Figure 10. Streaming data integration and real-time data processing pipelines in Spring Cloud Data Flow

circuit breaker can identify whether the behavior of one digital twin may interfere with another. In a more full-fledged setup, the circuit breaker would carry out a detailed Root Cause Analysis (RCA). However, preliminary performance experiments showed that offering such functionality continuously and and in real-time would not only require more computational resources, but also a greater redundancy of sensor components in cases of cyclic dependencies.

V. Evaluation

The above example is a simplified version of the configuration we will evaluate in our experimental setup. In our experiments, we ran multiple simulations of an artificial but common 'print and labeling' manufacturing process, augmented with multiple digital twins and circuit breakers. In this manufacturing process, a customer-specific label is created in the form of a heat-sensitive plastic, which is then wrapped around a container and then exposed to a blast of heat so that the plastic shrinks and fits tightly around the object.

Table I
Simulation details for individualized networked production

Category	*Count*
Types of raw materials	15
Suppliers	4
Temperature and humidity sensors	10
Product variants	100
Manufacturing processes (virtual space)	3
Digital twins	5
Circuit breakers	5
Faulty products	5%
Quality control misclassifications	10%

A. Deployment and configuration

To evaluate the impact of our digitial twin safeguard solution, we rely on simulations rather than real world evaluations. The reason is three-fold: (1) simulations allow us to repeat experiments under the same operating conditions, and (2) the occurrence of real-world failures is undetermistic, and (3) purposeful introducing faults into a production system comes with a non-significant monetary cost.

We therefore simulate a variety of induced errors in the

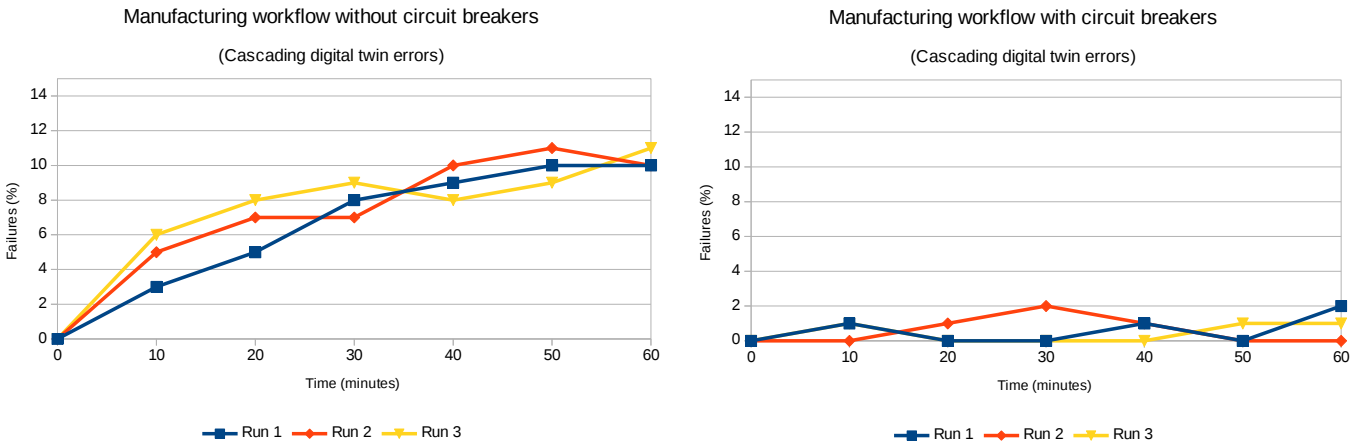

Figure 11. The impact of circuit breakers in digital twins for cascading errors

Quality Control and *Optimization* digital twins (see Table I for simulation details of the test deployment) with and without the circuit breakers, and measure the effectiveness and overhead:

- *Input*:
  - Missing input due to disconnected sensors
  - Incorrect input or outliers due to broken sensors
  - Malicious input from an adversary
- *Functionality*:
  - Faulty logic or implementation error
  - Disconnected actuators or hardware failures
  - Denial of service due to data floods

The misclassification case discussed earlier is an example of a faulty logic in the functionality of the *Quality Control* digital twin (e.g. due to an outdated machine learning model used for classification). The rescheduling executed by the *Optimization* digital twin not meeting its intended efficiency or cost saving objectives (e.g. due to inaccurate simulations of the manufacturing workflow) is an example of another.

All components in the virtual space as well as the digital twins have been implemented as microservices in the Spring Cloud framework[4]. More specifically, the streaming data integration and real-time data processing pipelines are deployed on top of the Spring Cloud Data Flow toolkit[5], as depicted in Fig. 10. For the sake of simplicity, all microservices were deployed on a single Dell PowerEdge R620 server with 2x Intel Xeon E5-2650 CPUs, 64GB of memory and 8 cores on each CPU with hyperthreading, resulting in 32 cores in total each running at 2GHz. The motivation for this deployment – compared to, for example, a distributed deployment in a Kubernetes container orchestration[6] – is the ability to measure the computational overhead of our solution compared to a baseline deployment with digital twins not safeguarded by the proposed circuit breakers.

[4]http://cloud.spring.io
[5]https://cloud.spring.io/spring-cloud-dataflow/
[6]https://kubernetes.io/

B. Experiments and results

We compared the impact of forced errors (e.g. mimicking hardware and network connection failures as well as implementation logic errors in the digital twin) on the ability to prevent digital twin failures from cascading further. Fig. 11 shows the results of 3 simulations, with and without circuit breakers. It is clear that the circuit breakers do not circumvent all digital twin errors from rippling through to other connected manufacturing processes and digital twins, the impact is nonetheless significant.

In a scenario where the quality assurance digital twin (which uses a machine learning model to classify the cause of the faulty product) is wrong in 10% of the cases, the errors eventually ripple through from the *Quality Control* digital twin to the *Optimization* digital twin, causing unnecessary costly reschedulings and reoptimizations of the distributed manufacturing workflow. The same simulations with circuit breakers in place reduce this undesired effect on average with about a factor 5.

In a second experiment, we measured the computational

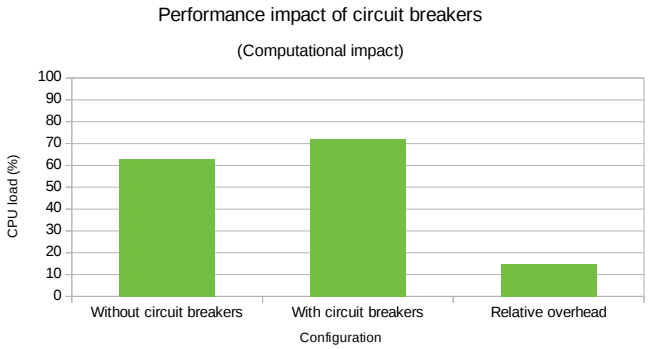

Figure 12. Measuring the overhead of the circuit breakers

overhead of the circuit breakers. As in our configuration each circuit breaker accounts for an additional microservice on our service, we expect a computational impact. For the given experiment, Fig. 12 shows the overall CPU load increasing from 64% to 72%, or a relative overhead of 15%. This relatively low performance overhead can be explained due to the computational intensiveness of the machine learning algorithm (i.e. a Random Forrest multiclass classifier) used to classify the cause of faulty products based on environmental and raw material parameters. The performance impact would be more significant if either the quality assurance method is less resource demanding, or if the circuit breaker would implement a computationally more intensive full-fledged Root Cause Analysis method.

C. Validity threats and discussion

The above experiments only provide insights of the feasibility and the main benefits of our solution. From a quantitative point of view, more statistical evidence and longitudinal studies are needed to be able to generalize to other smart manufacturing workflows. For that, we need to evaluate other scenarios considering:

- Larger distributed manufacturing processes with more complex dependencies between digital twins
- Multiple variations and implementations of circuit breakers protecting against different errors from propagating
- Different performance indicators and cost models of misbehaving digital twins, as not all errors should be treated equally

Furthermore, our circuit breakers aim to intercept local errors and prevent them from cascading to other systems and services. However, the circuit breakers themselves may – in theory – suffer from the same concerns (e.g. faulty triggering logic). Obviously, the goal is not to increase robustness and reliability by moving errors from one component to another, but rather to isolate them. The assumption made in our work is that the development of circuit breakers is less complex compared to the implementation of the digital twins that they are supposed to protect, such that the likelihood of errors in the former is lower. However, such assumptions are very hard to validate as they depend on the expertise of the process, software, and quality assurance engineers involved involved in the manufacturing workflow and cyber-physical production processes.

VI. Conclusion

In this paper, we investigated the impact of digital twins in Industry 4.0 manufacturing workflows, and to what extend failures in digital twins can jeopardize smart cyber-physical production processes. Building upon previous work, we proposed to add safeguards to digital twins in the form of feature toggles and software circuit breakers that are able to trap local errors and prevent them from propagating or cascading to other systems. We evaluated our solution on a simulated manufacturing process to validate the feasibility of the proposed solution. While our experiments demonstrate a positive impact on the robustness and reliability of the manufacturing workflow – and this with a limited performance overhead – further experiments are necessary to generalize the results to derive reusable insights, lessons learned and best practices.

One limitation of work is that it only traps errors and prevents them from propagating further, but it offers no mechanism to actually resolve them. Future research tracks could explore whether circuit breakers can help engineers reduce the Mean Time to Resolve (MTTR) in a realistic networked production environment.

Acknowledgements

This research is partially funded by the Research Fund KU Leuven. Work for this paper was supported by the European Commission through the H2020 project EXCELL (http://excell-project.eu/) under grant No. 691829. Work for this paper was supported by VLAIO through the SBO DiSSeCt project under grant No. IWT 150038.

References

[1] M. Knappmeyer, S. L. Kiani, E. S. Reetz, N. Baker, and R. Tonjes, "Survey of context provisioning middleware," *IEEE Communications Surveys Tutorials*, vol. 15, no. 3, pp. 1492–1519, Third 2013.

[2] J. Lee, B. Bagheri, and H.-A. Kao, "A cyber-physical systems architecture for industry 4.0-based manufacturing systems," *Manufacturing Letters*, vol. 3, pp. 18 – 23, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S221384631400025X

[3] A. Gilchrist, *Industry 4.0: The Industrial Internet of Things*, 1st ed. Berkely, CA, USA: Apress, 2016.

[4] S. Karnouskos, A. W. Colombo, T. Bangemann, K. Manninen, R. Camp, M. Tilly, P. Stluka, F. Jammes, J. Delsing, and J. Eliasson, "A soa-based architecture for empowering future collaborative cloud-based industrial automation," in *IECON 2012 - 38th Annual Conference on IEEE Industrial Electronics Society*, Oct 2012, pp. 5766–5772.

[5] L. Monostori, B. Kádár, T. Bauernhansl, S. Kondoh, S. Kumara, G. Reinhart, O. Sauer, G. Schuh, W. Sihn, and K. Ueda, "Cyber-physical systems in manufacturing," *{CIRP} Annals - Manufacturing Technology*, vol. 65, no. 2, pp. 621 – 641, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0007850616301974

[6] P. K. Verma, R. Verma, A. Prakash, A. Agrawal, K. Naik, R. Tripathi, M. Alsabaan, T. Khalifa, T. Abdelkader, and A. Abogharaf, "Machine-to-machine (m2m) communications," *J. Netw. Comput. Appl.*, vol. 66, no. C, pp. 83–105, May 2016. [Online]. Available: http://dx.doi.org/10.1016/j.jnca.2016.02.016

[7] P. O'Donovan, K. Leahy, K. Bruton, and D. T. J. O'Sullivan, "Big data in manufacturing: a systematic mapping study," *Journal of Big Data*, vol. 2, no. 1, p. 20, Sep 2015. [Online]. Available: https://doi.org/10.1186/s40537-015-0028-x

[8] R. A. Perez, J. T. Lilkendey, and S. W. Koh, "Machine learning for a dynamic manufacturing environment," *SIGICE Bull.*, vol. 19, no. 3, pp. 5–9, Feb. 1994. [Online]. Available: http://doi.acm.org/10.1145/182063.182067

[9] P. Priore, D. de la Fuente, J. Puente, and J. Parreño, "A comparison of machine-learning algorithms for dynamic scheduling of flexible manufacturing systems," *Eng. Appl. Artif. Intell.*, vol. 19, no. 3, pp. 247–255, Apr. 2006. [Online]. Available: http://dx.doi.org/10.1016/j.engappai.2005.09.009

[10] M. Grieves, "Digital twin: Manufacturing excellence through virtual factory replication," *White paper*, 2014.

[11] E. Negri, L. Fumagalli, and M. Macchi, "A review of the roles of digital twin in cps-based production systems," *Procedia Manufacturing*, vol. 11, no. Supplement C, pp. 939 – 948, 2017, 27th International Conference on Flexible Automation and Intelligent Manufacturing, FAIM2017, 27-30 June 2017, Modena, Italy. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2351978917304067

[12] A. Schuster, *Robust Intelligent Systems*, 1st ed. Springer Publishing Company, Incorporated, 2008.

[13] D. Preuveneers and W. Joosen, "Qoc^{2}breaker: intelligent software circuit breakers for fault-tolerant distributed context-aware applications," *Journal of Reliable Intelligent Environments*, vol. 3, no. 1, pp. 5–20, Jul 2017. [Online]. Available: https://doi.org/10.1007/s40860-017-0037-y

[14] F. Tao, J. Cheng, Q. Qi, M. Zhang, H. Zhang, and F. Sui, "Digital twin-driven product design, manufacturing and service with big data," *The International Journal of Advanced Manufacturing Technology*, Mar 2017. [Online]. Available: https://doi.org/10.1007/s00170-017-0233-1

[15] E. J. Tuegel, A. R. Ingraffea, T. G. Eason, and S. M. Spottswood, "Reengineering aircraft structural life prediction using a digital twin," *International Journal of Aerospace Engineering*, 2011. [Online]. Available: https://www.hindawi.com/journals/ijae/2011/154798/

[16] E. H. Glaessgen and D. Stargel, "The digital twin paradigm for future nasa and us air force vehicles," in *Proceedings of the 53rd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference*, 2012.

[17] M. Grieves and J. Vickers, *Digital Twin: Mitigating Unpredictable, Undesirable Emergent Behavior in Complex Systems*. Cham: Springer International Publishing, 2017, pp. 85–113. [Online]. Available: https://doi.org/10.1007/978-3-319-38756-7_4

[18] C. Peltz, "Web services orchestration and choreography," *Computer*, vol. 36, no. 10, pp. 46–52, Oct. 2003. [Online]. Available: https://doi.org/10.1109/MC.2003.1236471

[19] S. Schulte, D. Schuller, R. Steinmetz, and S. Abels, "Plug-and-play virtual factories," *IEEE Internet Computing*, vol. 16, no. 5, pp. 78–82, Sept 2012.

[20] J. Mangler, G. Stuermer, and E. Schikuta, "Cloud process execution engine - evaluation of the core concepts," *CoRR*, vol. abs/1003.3330, 2010. [Online]. Available: http://arxiv.org/abs/1003.3330

[21] K. Velasquez, D. P. Abreu, M. R. M. Assis, C. Senna, D. F. Aranha, L. F. Bittencourt, N. Laranjeiro, M. Curado, M. Vieira, E. Monteiro, and E. Madeira, "Fog orchestration for the internet of everything: state-of-the-art and research challenges," *Journal of Internet Services and Applications*, vol. 9, no. 1, p. 14, Jul 2018. [Online]. Available: https://doi.org/10.1186/s13174-018-0086-3

[22] K. Thramboulidis, D. C. Vachtsevanou, and A. Solanos, "Cyber-physical microservices: An iot-based framework for manufacturing systems," *CoRR*, vol. abs/1801.10340, 2018. [Online]. Available: http://arxiv.org/abs/1801.10340

[23] M. T. Rahman, L.-P. Querel, P. C. Rigby, and B. Adams, "Feature toggles: Practitioner practices and a case study," in *Proceedings of the 13th International Conference on Mining Software Repositories*, ser. MSR '16. New York, NY, USA: ACM, 2016, pp. 201–211. [Online]. Available: http://doi.acm.org/10.1145/2901739.2901745

[24] M. Nygard, *Release It!: Design and Deploy Production-Ready Software*. Pragmatic Bookshelf, 2007.

[25] M. Chen, S. Mao, and Y. Liu, "Big data: A survey," *Mob. Netw. Appl.*, vol. 19, no. 2, pp. 171–209, Apr. 2014. [Online]. Available: http://dx.doi.org/10.1007/s11036-013-0489-0

[26] IBM, P. Zikopoulos, and C. Eaton, *Understanding Big Data: Analytics for Enterprise Class Hadoop and Streaming Data*, 1st ed. McGraw-Hill Osborne Media, 2011.

[27] M. Fowler and J. Lewis, "Microservices," 2014. [Online]. Available: http://martinfowler.com/articles/microservices.html

[28] S. Newman, *Building Microservices*, 1st ed. O'Reilly Media, Inc., 2015.