The Uncertain Enterprise: Achieving Adaptation Through Digital Twins and Machine Learning Extended Abstract

Tony Clark

The Uncertain Enterprise: Achieving Adaptation through Digital Twins and Machine Learning Extended Abstract

Tony Clark

Aston University, UK
tony.clark@aston.ac.uk

1 Introduction

Systems, such as production plants, logistics networks, IT service companies, and international financial companies, are complex systems operating in highly dynamic environments that need to respond quickly to a variety of change drivers. The characteristic features of such systems include scale, complex interactions, knowledge of behaviour limited to localised contexts, and inherent uncertainty.

Knowing how to analyse, design, implement, control and adapt such systems is a difficult problem that lacks suitable mainstream engineering methodologies and technologies. Grand challenges such as Smart Cities, large-scale integration of information systems such as national medical records, and Industry 4.0 can only be achieved through the deployment and integration of information systems with existing systems.

The increasing connectedness of businesses and their reliance on software is leading to large-scale, networked, semi-autonomous interdependent system of systems. As a result, it is increasingly difficult to consider software systems in isolation, instead, they form a dynamically connected ecosystem characterised by a variety of interactions between them.

Any new system is thus deployed into a connected world and must be resilient in order to continue to deliver the stated goals by learning to suitably adapt to situations that may not be known a priori. Moreover, even system goals may change over time.

Traditional Software Engineering techniques tend to view the required system as having a fixed behaviour and being deployed into a well-understood operating environment. A typical development process expresses what the system must achieve in the form of a specification, how the system achieves the specified behaviour in the form of a design, and how the design is realised in terms of system implementation making appropriate use of underlying technology platforms. This works well when the characteristics of the system allows the design to be complete and when the environment into which the system is deployed is completely understood.

Enterprise systems must dynamically adapt. In the first instance such a system must adapt in order to achieve its goals within an environment that can

only be partially understood because of its complexity. Once the deployed system reaches a steady state, changes may occur in its environment or to its goals that require further dynamic adaptation. Approaches to adaptation include the following:

Product Line Engineering: This is an approach that aims to identify and include variability points into the design of a system [2]. The key SPLE techniques are not appropriate for addressing the Uncertain Enterprise since the variations must be known in advance.

Control Theory: Various forms of control theory have been developed in order to adapt physical systems. Generally these techniques measure the difference between observed behaviour and desired behaviour and translate these into a collection of control parameters that *nudge* the system into the right direction. A typical example of this approach is Model Reference Adaptive Control (MRAC) [1]. Traditionally these approaches use a collection of numerical equations to control real-time aspects of machinery; however, the architecture of the approach is appropriate for information systems providing there is an appropriate technology that can calculate the controls based on information system behaviour.

Rule Based: Rules of the form **if** `condition` **then** `adapt-action` can be used to encode knowledge about adaptation actions that are required when certain system conditions arise [3,4]. Like SPLE, this approach requires the conditions and actions for adaptation to be known beforehand, which is challenging in many cases of complex systems.

Architecture Based: An architecture-based approach organises a system as a collection of components that are co-ordinated via a manager. The manager can then change the way the components are co-ordinated depending on adaptation conditions. An example of this is the Monitoring, Analysis, Planning and Execution (MAPE) loop [5]. A key feature of this approach is to organise a complex system as a collection of decentralised components that can be co-ordinated in order to adapt.

Model Driven Development: Model Driven Software Engineering aims to generate systems from an abstract representation. Models can be used during run-time to allow a system to reason about itself and to adapt to changes in its goals or environment [7].

Most of these approaches rely on understanding the range of variability that is required and using an appropriate technology to encode the variability. Software has two significant dimensions of ambiguity that requires adaptation:

1. The behaviour of the system that must adapt may only be partially understood. Generally, it is the case that the required behaviour of a system is known, the components of the system are known together with their localised data and behaviour, however the algorithm required to co-ordinate the components in order to achieve the desired outcomes is vague, leading to the observation of *emergent behaviour*. The components must adapt in order to achieve the overall system behaviour.

2. When the environment changes or the goals of a system change it is often ambiguous as to how to modify the system components in order to continue operating. The system must be able to adapt in such a situation.

Dealing with both these kinds of ambiguity leads to a requirement for an approach that supports adaptation in the original system design so that it can be deployed and immediately adapt to its operating environment and then subsequently adapt to changes that occur. In all cases the behaviour of the system is driven by both its goals and the need to interact with its environment.

Given the requirement on information system adaptation, the key feature from existing approaches seem to be: organising information systems in terms of decentralised control (**Architecture Based**) together with an ability to compare the current execution history against the desired outcome in order to generate control parameter values (**Control Theory**).

The use of decentralised components (*actors* or *agents*) organised using an MRAC-style architecture that uses some form of machine learning to dynamically calculate the control parameter values produces the idea of *Digital Twin* that runs along side a complex system and dynamically adapts it to changes in the goals and its environment. Various forms of machine learning can be used depending on the circumstances, but a fully dynamic digital twin might benefit from the use of Reinforcement Learning [6] which is model-free and does not rely on previous execution histories of the system.

The importance and timeliness of applying Digital Twins to software and systems development is highlighted in the number of recent industry thought leadership editorials that describe the huge breadth and potential of this approach including Deloitte[1] , Simio[2] , Forbes[3] and Gartner[4].

Digital Twins can be applied in many different scenarios. Twins of physical systems can be used to provide a cost-effective way of exploring the design space for new products or optimisations. Twins of information systems can be used to achieve adaptation in complex ecosystems. Twins of populations (such as those modelled in the current Covid-19 pandemic) can be used to perform scenario playing where behaviour is inherently emergent.

It is desirable to envisage a situation where digital twins are used in a variety of modes for complex system analysis and development:

Analysis: A key requirement is to ascertain that a complex system is achieving its goals. A digital twin can provide a cost-effective solution through execution in a simulation environment producing traces that can be examined for occurrence of the desired (and undesired) behavioural patterns. A digital twin can also support what-if and if-what scenario playing to explore the system state space.

Adaptation: An existing system may expose a control interface that can be used for dynamic adaptation. A digital twin can be used to address the

[1] deloitte.com/us/en/insights/focus/tech-trends/2020/digital-twin-applications-bridging-the-physical-and-digital.html

[2] simio.com/blog/2019/11/14/top-trends-in-simulation-and-digital-twins-technology-for-2020/

[3] forbes.com/sites/bernardmarr/2019/04/23/7-amazing-examples-of-digital-twin-technology-in-practice/#4cd0672d6443

[4] https://www.gartner.com/en/documents/3957042/market-trends-software-providers-ramp-up-to-serve-the-em

problem of constructing the desired control inputs by running alongside the real-system and producing control commands based on a comparison of the observed and desired behaviour. This leads to the idea of a digital twin being used for continuous improvement of complex system behaviour through a variety of classical control theory and AI based techniques.

Maintenance: This is the single most expensive activity in a system lifecycle and can be responsible for over 60% of the overall costs. This is largely due to the present inability to explore the solution space effectively and efficiently. A digital twin can overcome this hurdle through what-if and if-what scenario playing to help arrive at a feasible transformation path from the "as is' state to the desired "to be" state in silico. Once the transformation path is vindicated, the necessary changes can be introduced into the real system in the right order thus providing assurances of correctness.

Design: A new complex system can start life as a digital twin that is used as a blueprint. The twin provides a specification of the behaviour for the real system and can be integrated with existing systems in the target ecosystem by observing their outputs. The design can then use adaptation to tailor its behaviour with respect to real ecosystem data.

This leads to a vision for future enterprise information systems based on **Digital Twins** that are used to address the various forms of uncertainty encountered in modern enterprise systems including: the behaviour of the system, the environment into which it is deployed and the goals against which it operates. Such digital twins are based on decentralised *agents* whose individual behaviours are controlled via *levers* and which expose execution *histories* to a reinforcement learning algorithm producing *controls* that satisfy dynamically changing *goals* and *environments*. In order to achieve this vision we require technologies that support the design, verification and run-time environments for such systems.

References

1. Itzhak Barkana. Simple adaptive control–a stable direct model reference adaptive control methodology–brief survey. *International Journal of Adaptive Control and Signal Processing*, 28(7-8):567–603, 2014.
2. Ana Eva Chacón-Luna, Antonio Manuel Gutierrez, José A Galindo, and David Benavides. Empirical software product line engineering: A systematic literature review. *Information and Software Technology*, page 106389, 2020.
3. Lauma Jokste. Comparative evaluation of the rule based approach to representation of adaptation logics. In *Proceedings of the 12th International Scientific and Practical Conference. Volume II*, volume 65, page 69, 2019.
4. Lauma Jokste and Janis Grabis. Rule based adaptation: literature review. In *Proceedings of the 11th International Scientific and Practical Conference. Volume II*, volume 42, page 46, 2017.
5. Nabor C Mendonça, David Garlan, Bradley Schmerl, and Javier Cámara. Generality vs. reusability in architecture-based self-adaptation: the case for self-adaptive microservices. In *Proceedings of the 12th European Conference on Software Architecture: Companion Proceedings*, pages 1–6, 2018.

6. Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.
7. Thomas Vogel and Holger Giese. *Model-driven engineering of adaptation engines for self-adaptive software: Executable runtime megamodels*. Number 66. Universitätsverlag Potsdam, 2013.