



# Automated generation of deployment descriptors for managing microservices-based applications in the cloud to edge continuum

James DesLauriers<sup>a,\*</sup>, Jozsef Kovacs<sup>a,b</sup>, Tamas Kiss<sup>a</sup>, André Stork<sup>c</sup>, Sebastian Pena Serna<sup>d</sup>, Amjad Ullah<sup>a,e</sup>

<sup>a</sup> Centre for Parallel Computing, University of Westminster, 115 New Cavendish Street, London, W1W 6UW, UK

<sup>b</sup> Institute for Computer Science and Control (SZTAKI), Hungarian Research Network (HUN-REN), Kende u. 13-17, Budapest, 1111, Hungary

<sup>c</sup> Fraunhofer Institute for Computer Graphics Research IGD, Darmstadt, 64283, Germany

<sup>d</sup> clesgo GmbH, Stuttgart, 70197, Germany

<sup>e</sup> School of Computing, Engineering & the Built Environment, Edinburgh Napier University, Edinburgh, UK

## ARTICLE INFO

### Keywords:

Cloud  
Edge  
IoT  
Descriptors  
Microservice  
Orchestration  
Deployment  
Digital twins  
Industry

## ABSTRACT

With the emergence of Internet of Things (IoT) devices collecting large amounts of data at the edges of the network, a new generation of hyper-distributed applications is emerging, spanning cloud, fog, and edge computing resources. The automated deployment and management of such applications requires orchestration tools that take a deployment descriptor (e.g. Kubernetes manifest, Helm chart or TOSCA) as input, and deploy and manage the execution of applications at run-time. While most deployment descriptors are prepared by a single person or organisation at one specific time, there are notable scenarios where such descriptors need to be created collaboratively by different roles or organisations, and at different times of the application's life cycle. An example of this scenario is the modular development of digital twins, composed of the basic building blocks of data, model and algorithm. Each of these building blocks can be created independently from each other, by different individuals or companies, at different times. The challenge here is to compose and build a deployment descriptor from these individual components automatically. This paper presents a novel solution to automate the collaborative composition and generation of deployment descriptors for distributed applications within the cloud-to-edge continuum. The implemented solution has been prototyped in over 25 industrial use cases within the DIGITbrain project, one of which is described in the paper as a representative example.

## 1. Introduction

The emergence of cloud computing, followed by the more recent rise of Internet of Things (IoT) devices has reshaped the way data can be collected and analysed. IoT devices collect huge amounts of data at the edges of the network, close to the data sources. However, the processing and storage capability of such devices is typically limited or non-existent. As a result, captured data is sent to more powerful computing environments for further processing, such as the cloud. Sending all that data to the cloud, though, introduces latency that may not be suitable in application scenarios where fast response time is required (e.g. in some manufacturing settings where quick response to certain machine conditions is essential to initiate changes or shut down the operation to avoid larger losses). To overcome this limitation, edge and fog computing have recently appeared as new paradigms, where computing capacity is placed closer to IoT devices and data sources.

This multi-layered setup, which includes IoT devices, potentially distributed edge and fog computing layers, and cloud computing facilities spanning multiple cloud sites and providers, offers new opportunities to process large amounts of data, but raises the complexity to develop, deploy and manage such applications. Data processing applications in this cloud-to-edge computing continuum are typically composed of a large number of interconnected microservices that need to be deployed, executed and in some scenarios even migrated between various layers and computing facilities.

In the last decade, several orchestration solutions have emerged to support the automated deployment and run-time management of microservices-based applications, first concentrating on single and multi-clouds and more recently targeting the entire cloud-to-edge continuum. These orchestrators typically require some form of deployment descriptor as input, detailing the application topology and the various

\* Corresponding author.

E-mail address: [j.deslauriers@westminster.ac.uk](mailto:j.deslauriers@westminster.ac.uk) (J. DesLauriers).

<https://doi.org/10.1016/j.future.2024.107628>

Received 13 July 2024; Received in revised form 12 October 2024; Accepted 22 November 2024

Available online 5 December 2024

0167-739X/© 2024 The Authors. Published by Elsevier B.V. This is an open access article under the CC BY license (<http://creativecommons.org/licenses/by/4.0/>).

policies (e.g. scaling or security) that govern the deployment and run-time operation of the application. The descriptors can be specific to the cloud technology that is used (e.g. Amazon's Cloud Formation Template [1]), specific to the actual orchestrator tool (e.g. Helm Charts or manifests in case of Kubernetes [2]), or more generic (e.g. TOSCA — Topology and Orchestration Specification for Cloud Applications — an OASIS standard [3]). While such deployment descriptors provide a relatively high-level abstraction for specialised developers, some scenarios require an even higher-level and more modular approach.

One example of such a scenario is provided by the European DIGITbrain project [4]. DIGITbrain develops a marketplace (called the Digital Agora) and an associated execution platform for manufacturing SMEs (small and medium-sized enterprises) where digital twin applications can be built and executed in the cloud-to-edge computing continuum. The digital twins, representing the behaviour of manufacturing machines/lines (or Industrial Products, as referred to in the project) are composed of lower-level building blocks (or Assets), such as Data (D), Model (M), and Algorithm (A). Each digital twin is composed of these Assets, building a so-called DMA Tuple. Additionally, each Algorithm can include multiple Microservices (Ms) that are independently built and can be executed on any suitable and appropriate compute device in the cloud-to-edge continuum. (In this article, following the DIGITbrain convention, names of DIGITbrain Assets (D, M, A, Ms) will always start with capital letters).

Two key features of the DIGITbrain concept are modularity and reusability. For such reasons, all Assets can be built (more or less) independently from one another, potentially by different providers, and then composed to a DMA Tuple representing the digital twin. At the end of this composition the DMA Tuple is described with a complex deployment descriptor that refers to multiple Microservices, their input and output data (typically including raw data coming from the factory floor) and a Model file (e.g. simulation model or trained AI model) describing the behaviour of the manufacturing machine/line. Microservices need to be deployed on a heterogeneous and distributed set of computing resources along the cloud to edge continuum, as defined in the final deployment descriptor.

The specific problem in the type of scenario described above is the authoring and composition of the deployment descriptor. Since Assets in DIGITbrain are authored independently and put together only in the final phase by the person composing the DMA Tuple, the creation of the final deployment descriptor needs to be automatic, based on information provided by the various Asset providers. Moreover, Assets are usually provided by experts of different profiles. While Microservices may be built by developers with more understanding of container technologies or even clouds or edge/fog computing, Models are typically created by simulation or AI experts and Data is managed by IT professionals close to the actual factory. Many of these actors may struggle with creating the deployment descriptor directly, especially when considering future dependencies with other — at that stage unknown — components of the DMA Tuple.

To overcome this problem, DIGITbrain adopts an approach where Assets are described by a rich set of descriptive metadata provided by the Asset developers as key/value pairs, followed by the automated creation of the deployment descriptor during the DMA composition phase. Asset developers need to provide key/value pairs that describe their Assets both for the human DMA Composer and also for the automated mechanism that will generate the deployment descriptor. Please note, that assuring the compatibility of the various Assets when composing a DMA Tuple is a complex task which requires the semantic understanding of these Assets. Currently, DIGITbrain leaves it to the human composer to understand and assess this compatibility, based on the provided metadata. Therefore, the automated deployment descriptor generation is not considering this semantic compatibility but only concentrates on the technical creation and usability of the generated outcome.

The rest of this paper describes how deployment descriptors are generated and composed automatically from key/value metadata provided by Asset providers in DIGITbrain. The metadata is translated and composed into a TOSCA-based Application Description Template (ADT) [5] that is consumed by a cloud-to-edge application-level orchestrator called MiCADO [6]. The resulting deployment descriptor (ADT) is published in the DIGITbrain Digital Agora and can be executed by the manufacturing company that wishes to simulate and analyse its manufacturing processes. While this article discusses concrete technologies, such as a specifically defined metadata structure, TOSCA and MiCADO, the solution itself is generic. The deployment descriptor generator, called ADT Generator, uses a highly flexible template-based structure that can be easily tailored or modified to other metadata formats or desired outputs.

This paper introduces a novel concept and solution for automatically composing and generating deployment descriptors for the cloud-to-edge continuum from a rich set of metadata. According to our knowledge, this is the first attempt to automate deployment descriptor generation while allowing for the collaboration of multiple actors, with potentially different technical backgrounds. The main contributions of our work are as follows.

- Demonstrate a generic and flexible approach to generating valid deployment descriptors from key-value pairs.
- Enable the composition of deployment descriptors for different tools with a focus on interoperability.
- Support the collaborative development of microservices applications, where different actors can publish different components of an application at different times.

The rest of this paper is structured as follows. Section 2 presents related work. Section 3 introduces background and related technologies by providing a short introduction to the DIGITbrain project, its metadata structure and the MiCADO cloud-to-edge orchestrator. Section 4 describes the ADT Generator and explains how the deployment descriptor is composed and created from the provided metadata. Section 5 presents an application example where an ADT is composed, generated and executed on cloud and edge computing resources. Finally, Section 6 concludes the paper and outlines future research directions.

## 2. Related work

Several solutions are available that enable the automatic generation of deployment descriptors, however, none support the collaborative nature of the kind of use case described in Section 1. This section discusses the various existing solutions in this space. The solutions are grouped depending on the component or abstraction level which the automatic generation is initiated from.

*From GUI.* One approach to producing deployment descriptors is with easy-to-use graphical interfaces that automate aspects of descriptor generation. For example, Eclipse Winery [7] is a web-based environment for graphically modelling applications and subsequently generating TOSCA topologies and plans managing them. Similarly, TOSCA Studio [8] is a model-driven tool that allows modellers to graphically design cloud applications to produce TOSCA models that conform to OCCI standards. TOSCA Studio also provides a built-in orchestration for deploying and managing applications at runtime.

*From architectural models.* Several research works facilitate the automatic generation of deployment descriptors from different architectural models. For example, Yussupov et al. [9] proposed generating deployment descriptors from BPMN (Business Process Model and Notation) [10] and TOSCA to model orchestration of functions and their automatic deployment. This solution, however, requires modellers to produce two different models. Firstly, to produce a generic BPMN-based model representing the function orchestrations, which could be transformed into a provider-specific deployment descriptor model

(e.g., ASL model for AWS). Secondly, to produce a TOSCA model to define a technology-agnostic function orchestration deployment model, which can be executed (i.e., deployed and orchestrated) by any TOSCA-compliant orchestrator. In the same realm, the authors in [11] proposed Caml2Tosca — a tool for automated generation of the TOSCA deployment model from UML — to reduce the gap and combine the notion of architecture modelling and application provisioning.

*From TOSCA.* The authors in [12–14] proposed solutions for the automatic generation of executable management workflows from declarative deployment models such as TOSCA. These solutions automatically transform TOSCA models into multiple executable BPMN-based plans that are provided as input to the orchestration engine responsible for the provisioning and holistic management of applications in heterogeneous environments. TORCH [15] is the underlying orchestration solution that converts TOSCA models into the BPMN plans.

Different to the aforementioned proposals, some solutions convert TOSCA to different resource-level orchestrator-specific templates. For example, Puccini [16] is an open-source front-end that translates TOSCA-based deployment models to a middle language called Clout and then Clout to an orchestrator-specific language (e.g., Kubernetes manifests), before being piped into the specific orchestration engine (e.g., Kubernetes CLI). The authors in [17] aim to separate application components from their hosting containers. For this purpose, their solution automatically generates deployable artefacts directly from TOSCA-based specifications. The artefact in this case could be a Docker compose file that packages both the application components and their solution-specific elements responsible for the component-level management. The artefact is then directly deployable using a Docker-based orchestration engine such as Swarm or Kubernetes. Similarly, the authors in [18] proposed TOSCA Light aiming to bridge the gap between the standard TOSCA model and production-ready deployment technologies such as Terraform and Kubernetes. More specifically, TOSCA Light transforms the TOSCA model to a corresponding representation that is compliant with the EDMM (Essential Deployment Metamodel) [19] — a set of core deployment modelling entities that are understandable by the vast majority of deployment automation technologies.

*From application components.* The AUTOGENIC [20] initiative aims to decouple the development of microservices from the underlying environment-specific runtime configurations. For this purpose, the AUTOGENIC solution automatically transforms the specific settings of microservices — provided by developers through a configuration model — to TOSCA-based dynamic and self-configurable microservices, targeting specific runtime environments.

Similarly, the SWITCH workbench [21], developed within the scope of a European research project with the same name SWITCH, provides an abstraction interface and infrastructure environment that facilitate the specification and management of the life-cycle of time-critical cloud applications. The various related aspects of application life-cycle management, such as user requirement specifications, application logic, and time-critical constraints during an application's deployment, execution and runtime, are handled independently by the different subsystems of the SWITCH workbench. To achieve the required integration between the subsystems, SWITCH uses TOSCA as the underlying modelling language to represent the information concepts provided by the parts of the system. The final output acts as the deployment descriptor used by one of the subsystems responsible for infrastructure planning, provisioning, deployment and execution of applications.

*Summary.* To summarise, the discussed approaches are diverse in their specificities. However, there is a common theme, i.e. the scope is limited to the standalone nature of a single application, where a deployment descriptor is transformed from one modelling template to another for a specific objective. In all cases, the components involved belong to one application, owned or managed by a single actor. In contrast, in this

paper, we follow an approach for marketplace-oriented collaborative environments, where the application and the corresponding deployment descriptors are composed by combining multiple independent components created at different times and belonging to different owners. Such an environment raises specific challenges as the description and specification of the independent elements and components need to have the flexibility to support composition and substitution into different deployment descriptors at later stages of the process.

There is a scarcity of research into the collaborative authoring of application deployment descriptors. As such, it is not possible to directly compare the solution presented here to others; the contribution of this paper is in fact the enablement of that collaborative authorship. Several earlier projects are described in the following section and the work herein extends and improves the approaches developed there. We have been able to validate the effectiveness of this solution through its successful application in the twenty-five real life use cases that feature in the DIGITbrain project.

### 3. Background technologies

#### 3.1. DIGITbrain and previous projects

The servitization of specialised software applications has been a key driving factor in the software industry over the last two and half decades. This effort was especially boosted by the introduction of the service models: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). One of the pioneering SaaS solutions was Salesforce's Customer Relationship Management (CRM) in 1999, followed by Amazon Web Services introduction of IaaS in 2002 and Fotango's launch of Zimki (first public PaaS) in 2005.

The European Commission also recognised this opportunity and started to push this topic in the research funding, especially in the manufacturing industry with the initiative “ICT Innovation for Manufacturing SMEs” (I4MS). The first I4MS projects (e.g. the CloudSME [22] and CloudFlow [23] projects) were focused on the cloudification of engineering applications like 3D modelling or 3D simulation for specific use cases [24]. A next batch of I4MS projects (e.g. the CloudiFacturing [25] project) moved from the engineering to the production domain, aiming to support the cloudification of software application and the connection to factory processes and data. At this point in time, it was recognised that the resulting cloudified solutions were too specific to the concrete use case and that reusability was limited and time consuming. Hence, the DIGITbrain project proposed a novel approach by breaking down such cloudified solutions into building blocks or modules that can be recombined or recomposed according to the specific needs of the individual use case, maximising the reusability and increasing the flexibility to support different scenarios.

DIGITbrain (“Digital twins bringing agility and innovation to manufacturing SMEs by empowering a network of DIHs with an integrated digital platform that enables Manufacturing as a Service”) is a research project funded by the European Commission's H2020 Programme. The primary aim of the project is to extend the traditional digital twin concept towards the Digital Product Brain that steers the behaviour and performance of an Industrial Product (mechatronic system or manufacturing machine) by coalescing its physical and digital dimensions and by memorising the occurred (physical and digital) events throughout its entire lifecycle.

DIGITbrain provides two major advances beyond the state of the art of digital twins. First, it enables constructing a digital twin from its building blocks, such as Data, Model and Algorithm, offering potential reusability of these Assets. Second, it supports the further analysis of events that occur during the execution of a digital twin by collecting, memorising and analysing these events and the specific conditions in which they occur (e.g. under what conditions in the past did a certain temperature exceed 100 degrees). The project aims to support digital twin developers and end-user manufacturing companies to speed up

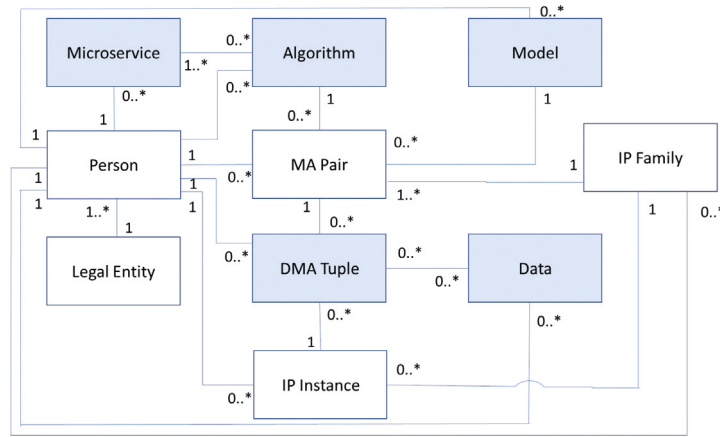


Fig. 1. High-level structure of DIGITbrain Metadata.

the development process by reusing already existing building blocks, to construct and execute digital twin applications from the convenience of a high-level graphical user interface, and to enhance, inform and accelerate the decision-making process with intelligence based on historical information collected over the lifecycle of the Industrial Product. As a result, a better understanding of Industrial Products (manufacturing lines) can be gained, leading to faster and more flexible reconfiguration and adaptation of manufacturing facilities, and supporting the provisioning of Manufacturing as a Service where the best manufacturing facility can be found and tailored to the specific requirements of an industrial customer on-demand.

### 3.2. DIGITbrain metadata structure

The novel deployment descriptor generation process that is described in this paper is primarily related to the composition of digital twins from already published Assets. This composition is made possible and supported by a rich set of metadata published and stored within the DIGITbrain Solution. Therefore, in this section a short overview of the DIGITbrain metadata structure is provided. This metadata is partially used by the ADT Generator during the automated process of generating the deployment descriptor, and it is also utilised by the human composer who creates the DMA Tuple and assures semantic compatibility of the utilised Assets.

The DIGITbrain metadata specification (Fig. 1 provides a high-level overview of the metadata structure while more details can be found in [26]) describes ten different entities. Six of these are Assets that are published and offered (potentially on a commercial basis) by providers and applied in DMA Tuples created and utilised by the targeted manufacturing end users. The common characteristics of Assets is that these are all created potentially independently from one another, keeping reusability and commercialisation in mind. Assets are published in a repository and can be found, selected and utilised by digital twin developers to create executable DMA Tuples. The six DIGITbrain Assets are the following [27]:

**Microservice (Ms):** A DIGITbrain Microservice is an executable in a containerised form, provided as an Open Container Initiative (OCI)-compliant container in a private or public container registry. Its configuration can be specified by a Docker-Compose file or a Kubernetes manifest (the appropriate configuration needs to be included in the metadata of the Ms). Microservices could have specific computational requirements (e.g. requires a GPU or must run on an edge node) and can also have dependencies (e.g. requires some other Microservice to be collocated on the same virtual machine).

**Algorithm (A):** An Algorithm is a combination of one or more Microservices. At least one of the Microservices of an Algorithm evaluates the Model, while the others can do additional tasks, for example

map data from data sources into the Model, provide a proxy server or a database server, implement file transfer etc. The metadata of an Algorithm refers to all the Microservices it is composed of.

**Model (M):** A Model is a description of a certain behaviour of an Industrial Product, according to given characteristics and operation conditions. Models are consumed as input files and evaluated by Algorithms. The creation of a Model typically requires domain specific knowledge related to the applied modelling technique- (e.g. AI, reduced order modelling, co-simulation etc. [28]) and the Industrial Product. The creation of a Model could be done either by editing a description file (e.g. YAML or XML) or processing/analysing data or simulation results to generate, derive or deduce a behaviour description (e.g. trained AI model or reduced-ordered model). Models are intended to be developed independently from the Algorithms, so that a given Model can be potentially reused by different compatible Algorithms.

**Data (D):** Data, typically sensor data collected from the shop floor, are essential to digital twins. Such Data are processed by the Algorithms, based on the specific behaviours described by the Models. After processing, the output of the digital twin also needs to be reported back to the user or stored in an appropriate format and location. Therefore, the digital twin needs to know the location, format and specific protocol how data can be accessed. When referring to Data as an Asset in DIGITbrain, this refers to the actual data resource where input and output data are stored and/or streamed from/to. Such Data Assets are usually local to the manufacturing companies and located within the factory. Data can also be stored in appropriate private or public cloud resources.

**MA Pair:** An MA Pair is a combination of a Model and an Algorithm that describes a specific behaviour of a specific type of Industrial Product. Such specific types of Industrial Products are called Industrial Product Families (IP Families). An MA Pair describes the potential behaviour of every instance of the same IP Family. However, it does not refer to Data Assets yet and therefore it is not specific to any instance of that particular IP Family. As MA Pairs are relevant to many instances of the same IP Family, these can also be considered as reusable Assets (i.e. companies operating the same type of IP can reuse the same MA Pair and combine it with their own specific Data to create their DMA Tuple).

**IP Family:** It describes the characteristics of a specific type of Industrial Product (a type of manufacturing machine).

Besides Assets, the DIGITbrain metadata structure also identifies four additional entities that are required to describe digital twins in the form of executable DMA Tuples. IP Instance characterises an instance of an Industrial Product (a particular machine/manufacturing line). Person and Legal Entity (see Fig. 1) refer to the individual and the organisation associated with the registered entity (any of the six Assets, DMA Tuple, and IP Instance), respectively. The final metadata entity is the composed and executable DMA Tuple.



**Table 1**  
Most significant metadata fields/ parameters utilised by the ADT Generator.

Field/Parameter	Explanation	Structure/Example
DMA /"deployment"	Describes a virtual machine or an edge device where microservices must be deployed	<pre>"A_HOST": {   "type": "cloudbroker",   "cloudbroker": {     "deployment_id": ...,     "instance_type_id": ...,     "opened_port": "22, 8080",   } }</pre>
DMA /"DataAssetsMapping"	Associates microservices with their used Data Assets	<pre>"DataAssetsMapping": {   "MSID_FE": { "inputfile": "dataasset1" }   "MSID_BE": {     "input": "dataasset2",     "config": "dataasset3"   } }</pre>
DATA /"URI"	Stores the location of any data object	"http://host.com:8088/mydir/mydata"
MODEL /"URI"	Stores the location of any model object	"http://host.com:8088/mydir/mymodel"
ALGORITHM /"listOfMicroservices"	Contains the list of ids for microservices to be deployed	["MSID_FE", "MSID_BE"]
ALGORITHM /"deploymentMapping"	Associates microservices to hosts to run on	{ "MSID_FE": "A_HOST", "MSID_BE": "A_HOST" }
MICROSERVICE /"deploymentFormat"	Selects the format of specifying container deployment details	"docker-compose" OR "kubernetes-manifest"
MICROSERVICE /"deploymentData"	Definition of compose or manifest	<pre>{   "services": {     "gui": {       "image": "nginx:latest",       "ports": ["8080:8080"],       "restart": "always"     }   } }</pre>

The above metadata structure was essential when designing and implementing the ADT generator. The ADT Generator specifically utilises the technical metadata provided with the DIGITbrain Assets, and also the metadata, especially the deployment information, from the DMA Tuple (shaded boxes in Fig. 1 show metadata that is utilised by the ADT Generator, while metadata from the white boxes are for the human composer only). Therefore, it needed to be assured that all information required for the automated generation of the deployment descriptor is available. On the other hand, due to the template-based implementation of the ADT Generator, only the template needs to be modified to tailor the ADT Generator to changes in the metadata structure or to accommodate completely different metadata. The core of the implementation remains the same.

Table 1 summarises and provides details of the most significant metadata fields and parameters that are utilised by the ADT Generator. As visualised in Fig. 2, the DMA Tuple is referring to the Model, the Algorithm and all Data Assets it is composed of. In this figure, Assets that are composed of other Assets are shown in rounded rectangles, where atomic Assets are shown in rectangles. Note how the DMA Tuple metadata also contains information related to the Host(s) of the executable digital twin it represents, for example the type and characteristics of cloud and edge computing devices its Microservices need to be deployed on. In the current DIGITbrain concept, every DMA Tuple includes maximum one Model and Algorithm, but it can include multiple Data Assets, clearly mapping these Data between the Microservices of the Algorithm.

### 3.3. MiCADO cloud to edge orchestrator

The DIGITbrain Digital Agora supports a pluggable execution engine for the deployment of DMA Tuples. Because DIGITbrain is concerned with executing application containers on cloud resources, an execution engine that supports both container orchestration and cloud resource provisioning was required. This higher-level orchestrator also had to support deployment of containers to edge, as this was another requirement within DIGITbrain.

For the current implementation of the DIGITbrain Solution, MiCADO [29] was chosen as the primary execution engine. MiCADO was originally developed in the European Horizon 2020 COLA (Cloud Orchestration at the Level of Application) Project. Built on the open-source Kubernetes orchestration platform, MiCADO started life as an application-level cloud orchestrator, but over time it has been extended with support for edge [6], enabling orchestration across the cloud to edge continuum. This section will briefly describe MiCADO, insofar as is required for the reader to appreciate its role in this work.

The MiCADO platform is a high-level orchestrator that offers a simplified interface to a prepared Kubernetes cluster and a cloud resource provisioning tool. MiCADO also adds a myriad of its own features, from cloud-agnostic auto-scaling to additional layers of security. The Kubernetes cluster in MiCADO is configured with sensible defaults and add-ons that are ready to use, like the Kubernetes Dashboard and a Prometheus-Grafana monitoring stack.

Two different cloud resource provisioning tools are available in MiCADO - Hashicorp's Terraform, and a smaller tool called Occopus [30]. These provisioning tools enable MiCADO to dynamically provision

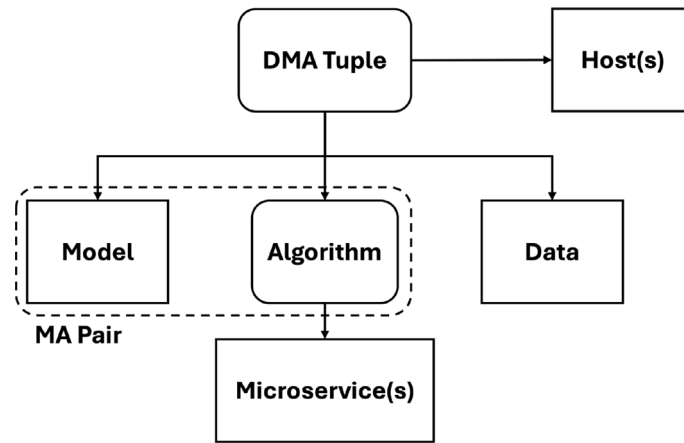


Fig. 2. Relationship of Assets with the DMA Tuple.

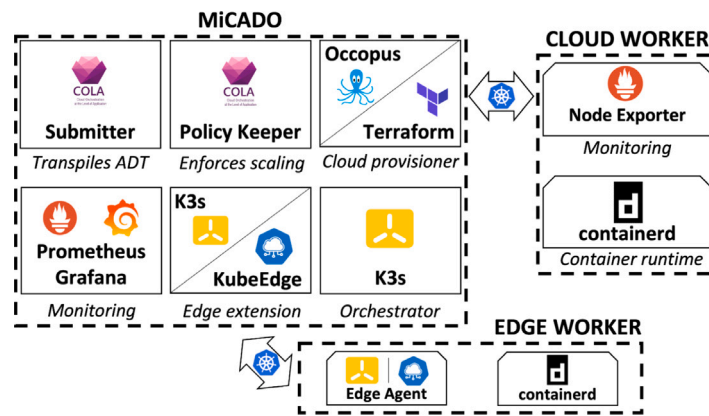


Fig. 3. Architecture of the MiCADO Cloud to Edge Orchestrator.

cloud infrastructure to host deployed applications. An internal component called the Policy Keeper enforces user-defined scaling rules to dynamically scale both containers and cloud resources at runtime. To bring orchestration to the edge, MiCADO relies on KubeEdge and K3S, two open-source CNCF incubator projects that extend the Kubernetes cluster towards one or more edge devices.

The interface to MiCADO is the Application Description Template (ADT) - a deployment descriptor that describes the application containers, cloud and/or edge infrastructure, and enforceable runtime decisions related to scalability, monitoring, and security [5]. The ADT is based on v1.x of the OASIS TOSCA (Topology and Orchestration Specification for Cloud Applications) Specification. Because TOSCA and the ADT are central themes in this paper, they are introduced in Section 4.1.1 below.

The architecture of MiCADO is visualised in Fig. 3. An internal component called the Submitter exposes a REST API and is responsible for interpreting an ADT and communicating information to Kubernetes and the cloud orchestrators to realise a successful deployment. The Submitter disseminates information in the ADT to Kubernetes, the cloud provisioner, the monitoring stack, the Policy Keeper and edge-related components. The cloud provisioner deploys and manages cloud worker instances, ensuring they are prepared with monitoring components and a container runtime. Kubernetes deploys and manages containers to these cloud workers. The Policy Keeper enforces any scaling rules, calling Kubernetes or the cloud provisioner as needed to scale a resource up or down. KubeEdge connects any edge devices to the cluster, creating additional workers where Kubernetes can deploy containers. Finally, the monitoring stack visualises CPU, memory and network metrics across the cluster, plus any application-specific metrics the user may have configured.

### 3.4. Topology and orchestration specification for cloud applications

The Open Oasis TOSCA (Topology and Orchestration Specification for Cloud Applications) specification aims to provide a generic language for describing applications in the cloud. It is an ideal framework to support the automated composition of deployment descriptors. TOSCA supports the description of all elements of a cloud application, from containers to cloud resources, meaning that different deployment descriptors could be unified into a single interface. Our justifications for choosing the TOSCA specification, including comparisons with other alternatives, are detailed in a previously published paper [5].

A TOSCA deployment descriptor is called a service template. The main component of this service template is the topology template, where the application is described. The topology consists of nodes, each one describing one element of an application — be it a virtual machine instance, a storage volume, a network interface, or some software. Nodes can express capabilities and requirements to create a rich description of the inter-relationships between all elements of the application. Nodes also define properties and an interface that together instruct an orchestrator what actions to take at different stages of the application lifecycle. Policies that target one or more nodes can further drive the runtime behaviour of the application once it has been deployed.

TOSCA is strongly typed. Each concrete node in the topology has a node type, and each policy has a policy type. These types can define default relationships, properties, interfaces and more, which are inherited by the node or policy that uses that type. A rich hierarchy of types can be defined to help abstract away complexity or provide re-useable structures for common components of an application.

A collection of type definitions is called a profile. The Simple Profile in YAML v1.3 is the current accepted standard, providing a reference for creating other custom profiles. Service templates will import one or more profiles, which are then used in the description of the application. Profiles are often relied upon by TOSCA orchestrators because they provide an easy way for the orchestrator to interpret nodes and policies that are described within a service template. MiCADO has a specific profile to support orchestration, which is introduced and elaborated in Section 4.

#### 4. Automated composition and generation of a deployment descriptor

Deployment descriptors contain the specific details that are required by an orchestrator or other tool involved in managing aspects of an application's lifecycle. They describe one or more aspects of the application to help realise a deployment, prepare some environment, dictate behaviour at runtime, or some other important functionalities. When more than one orchestration tool is required to realise the deployment of an application — for example a container and cloud orchestrator — then multiple deployment descriptors are needed too.

Authoring deployment descriptors is a complex task, requiring not only intimate knowledge of the application to be deployed, but also an understanding of each required orchestration tool, including the language, syntax and specification of its descriptor. In a typical microservices architecture, the complexity of this task increases with the number and requirements of microservices (containers) and cloud or edge resources (compute, storage, networking).

A microservices architecture may consist of many different containers, but these will typically be described in a single deployment descriptor, for example a Kubernetes manifest. Similarly, a different single deployment descriptor might describe all of the cloud resources that make up the cloud infrastructure required by the application. These descriptors can have both intra-dependencies (e.g. one container described to specifically interact with another) and inter-dependencies (e.g. one cloud resource intended to host a specific container). Because of these complexities and constraints, composition of the set of deployment descriptors required for an application is commonly carried out manually by the human person or team that owns that application.

##### 4.1. Collaborative deployment descriptors

Collaborative authoring of deployment descriptors, where different actors are specifying different aspects of the application at different times, is not compatible with this manual, human approach. In such cases, Microservices are single containers that are published individually and later assembled by a potentially different actor to create an Algorithm. The Algorithm is modular, in that it is composed of many potential reusable Microservices. The configuration of a cloud resource, again by a different actor, can take place long after a set of Microservices has been composed into an Algorithm. Based on our reading of the current state of the art, the collaborative authorship of deployment descriptors at different times and by different individuals was a novel problem, and so it necessitated a novel approach. A solution was needed that would automate the process of creating one deployment descriptor that expressed the overall structure of an application, but that was itself composed of existing descriptions of application components that could be orchestrated by different lower-level tools (Kubernetes and Occopus).

This section describes the concepts and technologies adopted to realise the automated composition and generation of deployment descriptors. The process that makes use of these concepts and technologies begins when an actor publishes an asset by providing the key-value metadata that describes it. The metadata for this single asset is stored, and this process is repeated for any actor that wishes to publish an asset, at any time. The next step of the process begins when a (potentially

different) actor wishes to combine already published assets to create their digital twin. At this time, the actor selects the desired assets and provides additional metadata to describe the specific interactions in this combination of assets.

Here, the automated generation takes over. First, the metadata that describes the cloud or edge resources is used to populate the fields of deployment descriptors that provisions a cloud instance or connects to an edge device. The metadata that describes microservices is used to populate the fields of deployment descriptors that describe the configurations of containers, referencing external sources for data or models according to the metadata of the other assets that were selected. At this point there are potentially multiple deployment descriptors saved as individual files, each describing a component of the digital twin that is to be deployed. The metadata that describes the specific interactions in a given set of assets populates a final deployment descriptor that references all of the individual deployment descriptors, with additional detail to describe any interactions or interdependencies between microservices and cloud or edge resources. All of these files are stored in a compressed archive format, which can be interpreted by a cloud orchestrator to realise the deployment of the digital twin.

The rest of this section provides details of the core concepts and technologies applied for the automated composition and generation of deployment descriptors.

##### 4.1.1. Application description template

The Application Description Template (ADT) [5] is the MiCADO specific name for a TOSCA service template. It adopts a profile based on the TOSCA Simple Profile and uses and extends many of the types defined in the 1.x versions of the specification. The ADT profile defines types for MiCADO components like Kubernetes, Terraform and Occopus, so that on submission, the appropriate actions can be taken to deploy and manage the different elements of the application. Nodes in the ADT describe two broad aspects of the application: the cloud or edge resources that make-up the compute and storage required by the application, and the application itself, in one or more OCI-compliant containers.

Historically, the ADT had always been authored as a single file, in YAML as per the TOSCA Simple Profile. All the necessary application components, such as cloud instances, edge nodes, and containers were defined as nodes in the one file, along with any relationships between nodes. At deployment time, that single file alone was fed to the Submitter component within MiCADO. Where the application description was a collaborative effort though, the definition of the application components and relationships were provided at different times — so potentially well before the single file would be created. These definitions were also provided by different actors, who would have no concept of what other components they should be defining relationships with. Collating all of this information together on-the-fly just before deployment would be a difficult and messy operation, and would not reflect well the modularity and reusability the solution was striving for.

Fortunately, TOSCA offered two key features that would permit DIGITbrain to adopt an elegant solution of sound design. These are described below. Please note, that while these features are widely applied in TOSCA, they were used in a different way and for different purposes than originally intended.

##### 4.1.2. Cloud service archives and substitution mappings

For larger applications in TOSCA a single YAML file quickly becomes bloated, disorganised, and difficult to manage. Instead, such applications can be described across multiple YAML files, where one file acts as the point of entry and refers to other files by importing them. These files are then all combined into one TOSCA Cloud Service Archive (CSAR) [31]. The CSAR is a zip-like archive providing a single, modular interface that can be shared, stored, and submitted to a TOSCA orchestrator. While the applications in our use case were not necessarily

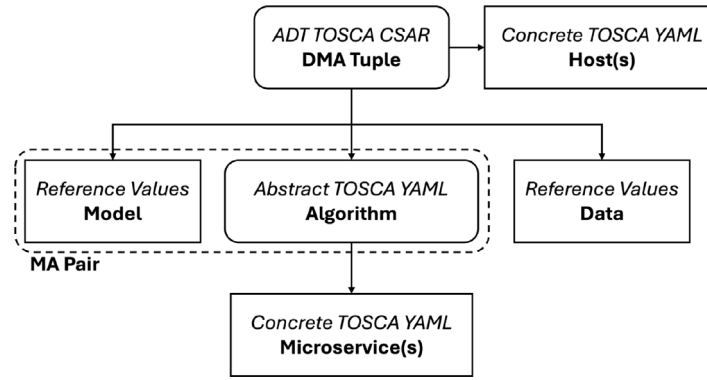


Fig. 4. Components of the DMA Tuple, with reference to their contribution to the CSAR.

large, this feature was key to our approach, because it meant that individual components could be described in their own individual files.

Describing applications in a CSAR is made possible by another feature of TOSCA called substitution mappings. Such mappings were originally intended to allow for the substitution of an abstract node with a concrete node. For example, some TOSCA might describe an application where a web server runs on a cloud instance with some minimum hardware. Since the exact specification of the cloud instance is not important, it can be defined as an abstract node, with the requirement that it meets the minimum hardware and serves the web server. When the time comes, a file with a concrete node describing a suitable cloud instance can be added to the CSAR. Substitution mappings will read the requirements on the abstract node and then orchestrate the concrete node to realise a successful deployment.

While not the original intention of the CSAR and substitution mappings, these features together suited the unique requirements of collaborative publishing very well. Published microservices, cloud instances and edge nodes would all be defined as concrete nodes, each within their own individual ADT. Any relationships that were needed for the overall application could be defined on the abstract nodes, and when substituted, they would apply to the concrete nodes instead. Since each ADT was a complete TOSCA-compliant file, they could each be independently validated and reused as needed.

For DIGITbrain, this meant that when the Algorithm provider had selected the Microservices that made up their Algorithm, and had specified any relationships between them, the ADT describing the overall application could be generated and added to the CSAR. This ADT contains abstract nodes that make reference to concrete ones, and any necessary relationships are defined between the abstract nodes. Microservices would already have their concrete descriptions in individual ADT files, and these could be added to the CSAR to support substitution mappings at deployment time. When a concrete Host is defined at the time the DMA Tuple is published, this too will be a concrete node in an ADT file, and this too can be added to the CSAR. This is visualised in Fig. 4, where the role of each Asset with respect to TOSCA is shown. At some time prior to the DMA Tuple compilation, Microservices will have been defined each as its own concrete TOSCA node, and the Algorithm will have been defined with abstract nodes as placeholders for the Microservice(s) and Host(s).

All of these elements of TOSCA have been adopted in DIGITbrain, resulting in a solution that uses the CSAR to provide a unified interface, substitution mappings to support collaborative publishing of Assets and a MiCADO-specific profile to ensure the final descriptions are properly orchestrated by Kubernetes and Occopus.

#### 4.2. ADT generator architecture

The automated composition and generation of the deployment descriptors in DIGITbrain is one of the key challenges described in the

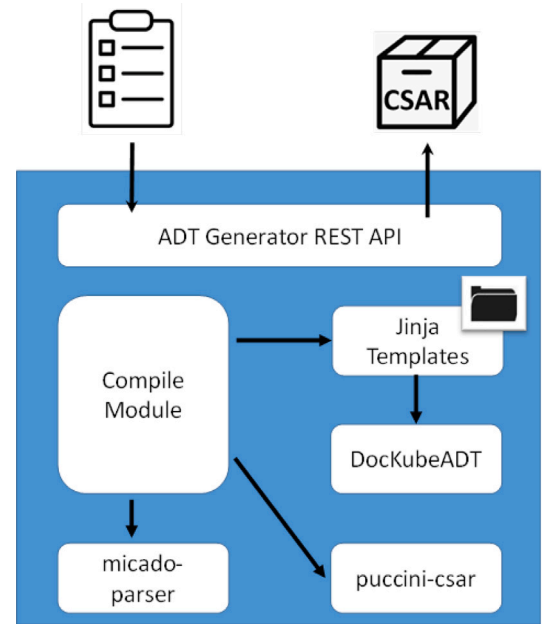


Fig. 5. Architecture of the ADT Generator.

paper. Based on the principles and concept of descriptor generation, a tool called the ADT Generator [32] has also been implemented to realise the functionality as part of the infrastructure.

The main purpose of the ADT Generator is to produce MiCADO compliant ADT descriptors based on the DIGITbrain Assets provided as input for a given application. The ADT Generator has been designed as a RESTful service, where the service receives the metadata for the Assets and produces and returns an ADT that can be submitted to MiCADO. As reflected in the architecture Fig. 5, the ADT Generator service integrates multiple components that will be described in the following paragraphs.

Compile module is the main component of the ADT Generator responsible to coordinate the steps of the ADT generation and compilation. It implements a workflow by invoking the necessary tools and libraries in the appropriate order.

Jinja templates as input for the jinja2 tool as a Python package is used to render the ADT, based on the key-value pairs of the metadata to fill placeholder values in a skeleton template specific to each Asset. By relying on Jinja, many changes or updates to the final output of the ADT Generator can be achieved simply by modifying the template, reducing the necessity for frequent changes to the codebase. Moreover, Jinja enables the flexible replacement of the generated deployment



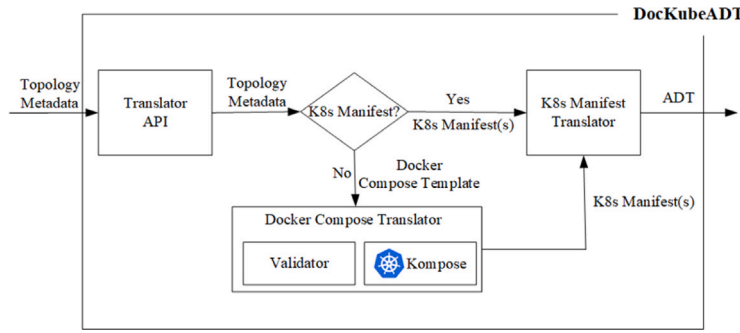


Fig. 6. Internal operation of DocKubeADT.

descriptor different from TOSCA in case another standard must be supported by the ADT Generator in the future.

DocKubeADT [33] is a Python package developed by the University of Westminster, which converts a Docker-Compose or Kubernetes manifest file to a valid MiCADO ADT. Originally, it was a standalone command line tool where the main goal was to ease the conversion of compose and manifest files to MiCADO ADT. DocKubeADT as a library is invoked during rendering of the Jinja template to produce a valid container configuration for a Microservice Asset. The output of DocKubeADT is then rendered into the generated ADT.

Puccini [16] is a TOSCA processor that makes available a tool for packaging individual TOSCA templates into a compressed TOSCA Cloud Service Archive (CSAR). The input to this tool is the set of individual Assets, and the output is a CSAR, or multi-file ADT, describing the overall architecture of the application to be deployed by MiCADO which contains the combination of microservices, algorithm, data, model, and deployment.

MiCADO Parser [34] is a library and command-line interface used inside the MiCADO platform for parsing and validating ADTs. To validate the generated output of the Compile module, the `micado-parser` library is invoked as a final step. This component wraps the functionality of the OpenStack TOSCA Parser [35], extending it with functionalities and additional validation specific to the latest TOSCA versions used by MiCADO. The OpenStack TOSCA Parser supports v1.2 of the TOSCA Simple Profile in YAML to create an in-memory graph of TOSCA nodes and their relationship.

REST API is the interface exposed by the ADT Generator. Because of the independent nature of Assets, the API does not return a traditional, single-file ADT for MiCADO. Instead, each Asset is compiled down to a respective descriptor file that leverages TOSCA's Substitution Mappings to bring together these separate pieces at deployment time. The API contains routes for compiling metadata of the Microservice and Algorithm Assets as well as the Deployment description, pulling values from Data and Model metadata to complete these as required. Also, there is a route for compiling all the incoming Assets and descriptions into one comprehensive CSAR file describing the entire application. The response contains the outcome of the compilation, a filename for download, and a link to a detailed log of the compilation processes and steps.

#### 4.3. DocKubeADT internal operation

DocKubeADT is one of the most complex tools integrated by ADT Generator and is considered as the heart of the ADT generation. To understand the operation of ADT Generator, a deeper insight of the DocKubeADT tool is necessary since containers, environment variables, arguments, ports, volumes, policies are all converted by this tool in several steps.

DocKubeADT is invoked by the ADT generator to translate the container topology defined in the Microservice metadata (based on either a Docker Compose template or Kubernetes (K8s) manifest), to

the format expected by the MiCADO ADT. It can be used both as a standalone tool or a library. To use it as a tool, we simply need to execute DocKubeADT while passing the Compose or manifest file as an argument. It will then provide the ADT file as an output. To use it as a library, DocKubeADT provides an API, which requires the compose or manifest as an input. Subsequently, it generates the relevant portions of the ADT. Since DocKubeADT is used by the ADT Generator as a library, we provide an overview of it, in Fig. 6, from this perspective.

In DocKubeADT the main strategy is to focus on converting Kubernetes manifest description into an ADT. As we use manifest format as input for producing the ADT, different formats can only be supported (e.g. docker compose) if intermediate conversion is also performed. As such, the conversion of Docker Compose into Kubernetes manifest is handled by an external tool called Kompose. So, overall the Docker Compose is converted in two steps, while manifest is directly converted as one single step. As seen in Fig. 6, DocKubeADT has three main components to implement this strategy: Translator API, Docker Compose Translator and Kubernetes Manifest Translator.

Translator API is invoked during rendering of the Jinja template to produce a valid container configuration for a Microservice Asset. The API requires the container topology of the Microservice as an input. The topology can be defined as a Kubernetes manifest or Docker Compose template. When using a Kubernetes manifest format, both the single and multi-part YAML files are accepted. Once the input is received, depending on its format, the topology is forwarded to either the Docker Compose or K8s manifest translator component.

Docker Compose Translator converts a Compose template to a Kubernetes Manifest. This is achieved using the open-source Kompose tool [36]. However, before invoking Kompose, the Docker Compose data is validated. The validation process involves checking whether multiple services are defined in the data. If multiple services are defined, the data is not processed. That is, per invocation, only a single service definition is allowed in the template. This aligns with the process of defining each Microservice Asset separately in the DIGITbrain project.

After initial validation, some important information about the volumes Compose file is captured. The Kompose tool does not handle the concept of mount propagation, which is important if two different containers on the same host need to share a volume on the host. The relevant information is collected from each defined volume in the Compose file, so alterations can be made by the K8s Manifest Translator in the next step.

Once this validation is performed and information is gathered, the template is converted to a Kubernetes manifest and passed to the K8s manifest translator.

K8s Manifest Translator can be invoked by either the Translator API or the Docker Compose Translator. It expects K8s manifest and converts them to the relevant portions of an ADT. This translation process involves nesting each Kubernetes manifest within its own node, using a custom TOSCA Node Type that MiCADO identifies as a raw Kubernetes manifest during deployment and execution.

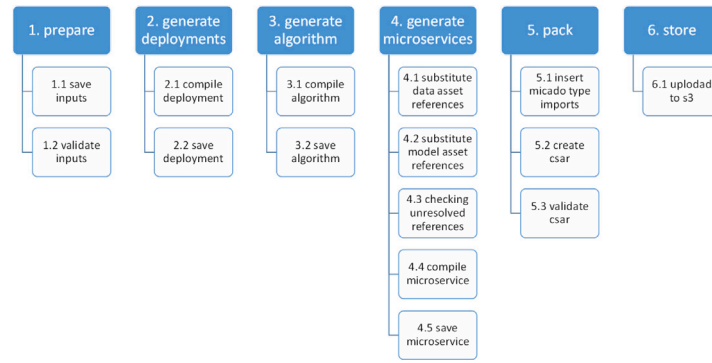


Fig. 7. Workflow of compilation in ADT Generator.

After nodes are created, additional optional information might be added. DIGITbrain Microservice providers have the option to add static files, much in the same way they would by mounting a local directory in the Docker Compose. If required by the provider, files are created at this time as Kubernetes ConfigMaps, which are simply static files mounted into a container at runtime. Any required volume alterations that were identified during the Docker-Compose Translator step are also applied at this time. When done, the output of this step is then rendered into the ADT by the ADT Generator, which invoked DocKubeADT.

#### 4.4. ADT generator internal operation

The ADT Generator performs a sequence of steps in order to convert/compile a complete DMA Tuple consisting of Deployments, Data/Model Assets, Algorithm and Microservices into a MiCADO ADT. The steps to perform are implemented by several tools mentioned in Fig. 5 and can be grouped into 6 main phases as summarised in Fig. 7. In the following, the phases and their internal steps are introduced in detail.

**Phase 1 (preparation).** During the preparation phase, the ADT Generator splits the incoming DMA Tuple into pieces of different Assets and logical components for further processing. For debugging purposes, the tool saves these components into files locally. The next step is to validate the Assets and components syntactically and semantically. The existence of the required parameters as well as the format of the parameters are checked to make sure no incorrect DMA Tuple has arrived.

**Phase 2 (generate deployments).** The entire compilation starts with the compilation of the deployment description(s) as it is independent from any other Assets in the DMA Tuple in terms of conversion. Deployment description is located as the value of “deploymentData” of the DMA Tuple. This parameter should contain a valid definition of either a virtual machine or an edge device that will host/execute Microservices as part of the DMA Tuple. In this phase, after the compilation of deployment description into MiCADO compute node, the result is saved into a deployment file.

**Phase 3 (generate algorithm).** The Algorithm compilation produces a node template for MiCADO ADT where the Microservices and Deployments are listed. For Microservices, the associated resource which will host the container is also defined by referring to a deployment node (generated in phase 2) through the “host” parameter. The result is saved into the corresponding algorithm file.

**Phase 4 (generate microservices).** This is the most complicated compilation phase. It iterates over the Microservice definitions and generates a complete Microservice node definition substituted with its referenced Data and Model Assets, with open parameter definitions and with container definition with all the parameters such as name, command, arguments, environment variables and so on. During the iteration the following main steps are performed: First the referred Data Assets are collected and their referred values are substituted in the

metadata description, then the same is done for the Model Assets. At this point all references to Model and Data Asset values in the metadata description must be resolved. The compilation terminates with error here when an unresolved reference (to Model or Data Assets) is found, otherwise continues with compilation of the Microservice Asset. The result is then saved into a file to be linked by the Algorithm Asset in Phase 3.

**Phase 5 (packing).** At this point all deployments (describing cloud or edge resources), all Microservices (describing the containers) and the Algorithm (describing the mapping between the resources and containers) are generated and stored in TOSCA files. To pack them into a CSAR, first the definition of the MiCADO ADT TOSCA types (referred to by the aforementioned files) must be resolved. Due to MiCADO ADT parsing speed considerations, this resolution is done by copying all MiCADO type definitions into the CSAR archive. After the copy, the CSAR zip file is created by a tool called *puccini-csar* and the result is finally validated by *micado-parser*. Upon successful CSAR creation and validation, the ready-to-run MiCADO ADT is available in the working directory associated with the DMA Tuple being processed.

**Phase 6 (storing).** Storing the generated CSAR can be configured/requested from the ADT Generator. Currently, it supports uploading to S3 buckets predefined in the configuration of the ADT Generator. Both, the CSAR and the log file (generated during the phases of compilation to keep track of the entire workflow) are uploaded to the target bucket with appropriate naming convention and the endpoints are returned as part of the response to the REST API invocation.

#### 4.5. ADT generator deployment

The initial setup and deployment of the tools described above are integrated into the deployment process of the ADT Generator. The source code and installation package are available at [32] which is a snapshot of the ADT Generator GitHub repository. The deployment includes *micado-parser* and *dockubeadt* as Python libraries listed as dependencies, while the *kompose* binary and the MiCADO ADT TOSCA types are fetched by the deployment script during the initialisation of the ADT Generator environment. Finally, the *puccini-csar* tool (a single script) is added as part of the source repository. Overall, the deployment process is fully automated to support the easy installation of the ADT Generator. The hardware requirements are very low, with the ADT Generator service able to run on a 1CPU and 2 GB RAM cloud virtual machine instance.

### 5. Digital twin for punching machine

Within DIGITbrain (Db), more than 25 industry case studies have been implemented where digital twin applications were developed by composing them from their fundamental building blocks of Data, Models, Algorithms and Microservices. In all cases, these building blocks have been published in the DIGITbrain Solution by providing a set of

metadata, as introduced in Section 3.2, composed into DMA Tuples, and then the deployment descriptors were automatically generated by the ADT Generator. In this section, we demonstrate how these concepts and methods were applied to a scenario around a metal punching machine.

In this case study, a new kind of digital twin for a subset of the behaviour of the machine was created, namely a digital twin for the structural behaviour of the stamp centred around a fast GPU implementation of the finite element (FE) method which achieves new levels of runtime performance. The idea/approach behind this scenario is to equip a punching machine with force sensors, send this information over the network via MQTT to a numerical simulation for structural analysis, feed this simulation with the forces, and predict the deformation of the stamp given its simulation model and the real-time data from the shop-floor.

The aim is to achieve new kinds of adapted control of machines in general and for the stamping machine in particular, in order to reduce loads and wear of the machine by taking not just the sensor information into account but also the effects on (parts of) the machine. The scenario is depicted in Fig. 8. The actual simulation component in this scenario runs close to the sensors and the data source on a GPU-equipped edge device. In the next subsection, the numerical solver RISTRA is introduced, followed by describing the setup of the corresponding DMA Tuple and the steps and extensions that were required to facilitate the scenario. Finally, the automated generation of the deployment descriptor and the run-time results of the simulation are demonstrated.

### 5.1. General introduction of RISTRA

The basis for the numerical simulation of the structural behaviour of the punching machine is RISTRA (Rapid Interactive STRuctural Analysis) [37], a software library for fast structural analysis. Structural analysis is one of the most frequently used simulation domains in industry when designing parts of different kinds of products, be it vehicles, buildings, household appliances, etc. Speeding up the simulation and optimisation loop can contribute to finding better solutions in a shorter time, thus reducing time to market.

RISTRA can be integrated into existing CAD/CAE (Computer Aided Design/Computer Aided Engineering) packages to deliver new levels of performance within such applications. Benchmarking RISTRA against CPU-based commercial-off-the-shelf (COTS) alternatives has shown performance benefits of up to a factor of 100 times (two orders of magnitude), while comparisons with GPU-based COTS indicated up to 30 times speedup. These results have been achieved under the constraint of not deviating from the reference solutions by more than 1%. RISTRA's speed is achieved by fully leveraging the massive parallelism of GPUs by implementing the whole finite element method completely on GPUs. In contrast, many other systems only leverage the GPU for some steps of the FE method, and slow memory transfers between CPU and GPU prevent fully exploiting the benefits of GPUs. With sufficient GPU memory, RISTRA can solve problems with more than 10 million degrees of freedom (DoF). RISTRA currently supports linear structural analysis, tetrahedral meshes, linear, quadratic, and cubic shape functions (TET4, TET10, TET20), linear isotropic and anisotropic materials as well as time-dependent and modal simulation.

### 5.2. RISTRA as part of a DMA tuple

To support the scenario illustrated in Fig. 8 and described above, the original implementation of RISTRA needed to be extended to support execution on edge devices. As a starting point, RISTRA was implemented for NVIDIA desktop GPUs using CUDA. This initial version was able to read simulation models as files from local storage. The simulation models consist of tetrahedral meshes, boundary conditions, and loads. To realise the scenario, two extensions needed to be made to RISTRA: a) a possibility to feed it with (sensor) data via an API and b) port it to NVIDIA edge devices with GPUs, resulting in RISTRA@Edge,

a variant of RISTRA for GPU-enabled edge devices, in particular the NVIDIA Jetson platform which is, to the best of our knowledge, the first GPU-accelerated FE simulation software for embedded systems [37]. These two extensions were prerequisites for publishing and executing RISTRA on the DIGITbrain Solution.

The next step was the creation of the simulation model by a domain expert and uploading it to a model repository. The simulation model was created based on the CAD model, simplifying some of the details and defining boundary conditions and load cases. Although initial load cases (directions of force and strength) were set, the actual amount of force is read later from the sensor information and applied to the digital twin. The simulation model then needs to be published in a repository as a single zip file. Models consisting of more than one file always need to be packaged into one zip file. The model repository is a simple file storage where Model files can be stored and downloaded at run time by the associated Microservices. The Db Solution does not require the use of a specific model repository. Any file storage set up by Model providers is appropriate, as long as it can be accessed remotely and the associated Microservice understands the file transfer protocol that the repository uses.

Next, the Model needs to be made known to Db Solution as a Db Asset. In this step, metadata for the Model, consisting of a filename, path, repository URI, along with many other optional metadata entries, needs to be entered to describe and characterise the kind of Model (see left-hand side of Fig. 9).

Unfortunately, RISTRA (as many other existing applications) cannot retrieve Models from repositories by default. Instead of modifying RISTRA, requiring access to the source code, a less intrusive solution is to implement a dedicated Microservice to download the Model file from the model repository. RISTRA reads input from files and to make RISTRA independent from sources, the Model Retriever microservice is introduced as an additional component. Model Retriever collects inputs regardless of the source and provides input files for RISTRA. With this solution, we have the benefit of abstracting the repository/source from the consumer (RISTRA), thus, the consuming service does not need to know about the details of the repository.

Additionally, if solvers shall be fed with data in real time, an API is required. To feed RISTRA with data from the force sensors, we decided to implement the Data Bridge Microservice. Data bridges have the benefit of abstracting the data source from the consumer, thus the consuming service does not need to know the details of the data source. Instead, the Data Bridge serves its API according to a specification.

Therefore, the Algorithm consists of the three Microservices described above: RISTRA, Model Retriever and Data Bridge. All three Microservices need to be published in the Db Solution by providing their related metadata. The publishing process creates IDs for each published Asset (see Fig. 9 centre). These IDs will be used when composing high-level Db Assets (e.g., MA Pairs) and DMA Tuples.

Once the Microservices are published, an Algorithm can be composed out of them by searching the Microservices from the Digital Agora and referencing them by ID. In addition, further metadata (e.g., author, date, name) for the Algorithm is specified while publishing it to DIGITbrain (see right-hand side of Fig. 9).

By now, a Model and an Algorithm have been specified (see left side of Fig. 10) that can be combined to represent and evaluate the behaviour of punching presses that are built around the same stamp—a family of punching presses (or an Industrial Product Family). Like the composition of an Algorithm, the user can find Models and Algorithms via the Digital Agora and reference them when creating an MA Pair. As in the case of Algorithms, additional metadata can also be specified during this process.

The next step is to register/publish the data source (see the middle of Fig. 10). The data in this specific case comes from sensors installed on the machine and delivered via an MQTT broker. Like Models and Microservices, Data sources are published to the Db Solution by providing a set of metadata.

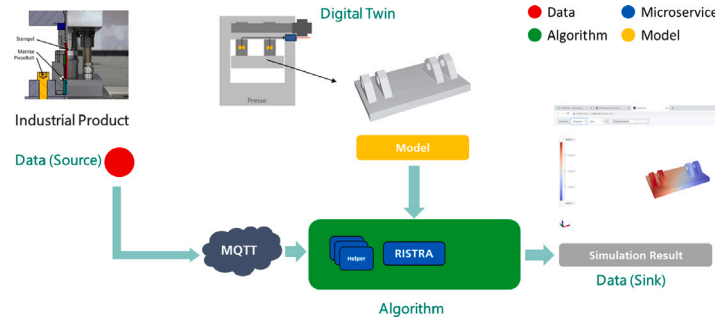


Fig. 8. Digital Twin Scenario.

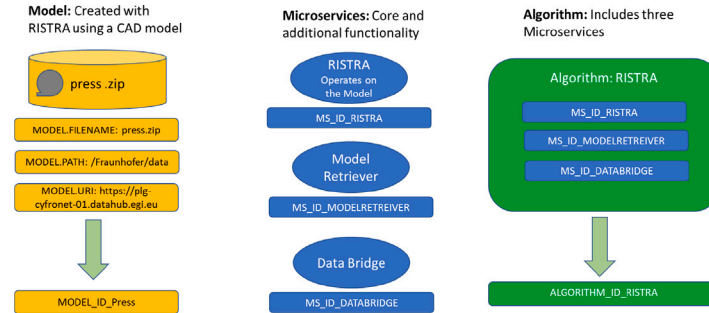


Fig. 9. Publishing Db Assets - Model, Microservice and Algorithm.

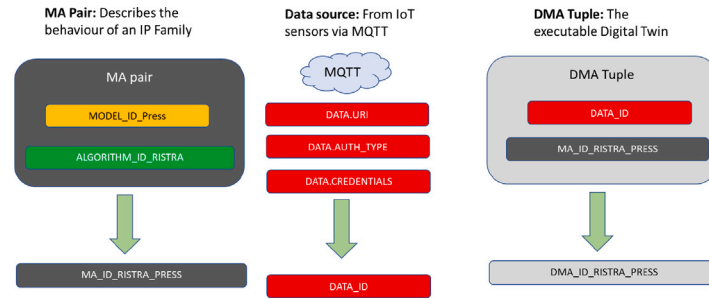


Fig. 10. Creating an MA Pair, publishing a Data resource and composing the DMA Tuple.

The final step in the publication and composition workflow is the creation of a DMA Tuple (see the right side of Fig. 10) out of previously published DIGITbrain Assets. This step is carried out by a person acting in the role of the DMA Composer, typically a person that belongs to or acts on behalf of a manufacturing company, typically because of access restrictions to data sources within the factory. The DMA Tuple is built by referencing a Data source and the MA Pair that uses this Data together with a Model to evaluate/predict the behaviour of the respective manufacturing machine. Thus, DMA Tuples are connected with certain instances of Industrial Products. One or many DMA Tuples represent a Digital Twin for such an instance.

The above case study of a digital twin for a punching machine is one example of the possible scenarios that the proposed approach could be applied to. Nevertheless, the approach is general and the presented modularisation and reusability have been successfully implemented in other case studies, such as production optimisation, defect detection and quality control, optimisation of injection moulding processes, monitoring and optimisation of additive manufacturing processes, co-simulation, or life cycle analysis [38]. The proposed approach was successfully applied to all the above scenarios, which were covering different manufacturing sectors such as textile, automotive, rubber and plastic products, metal products, agriculture, machinery and equipment, food production, among others. This approach is also applicable

beyond the manufacturing industry, since the different building blocks (i.e. Data, Model, Algorithm) could represent multiple industrial solutions, making it generic for any kind of application that brings together software experts (for the Algorithm), domain experts (for the Model) or end users (for the Data).

### 5.3. Deployment

The steps described in Section 5.2, whereby metadata for each of the required Assets is provided, happens in the Digital Agora (see Figs. 11 and 12). Digital Agora is a web-based frontend and management framework for managing users, and their Assets, compilation and execution of the DMA Tuples. This environment interplays with the aforementioned ADT Generator as part of its portfolio in order to provide executables for MiCADO.

Assets like Algorithms (in Fig. 11), which are composed of other Assets (Microservices in this case), are configurable using a search function that can locate previously published Assets.

In the DMA Tuple configuration (see Fig. 12), both Data and the MA Pair (here called Behaviour) can be found using the search, while the Host Configuration details are provided via drop-down menus and text fields. In Fig. 13, components of the application are mapped to specific Assets.



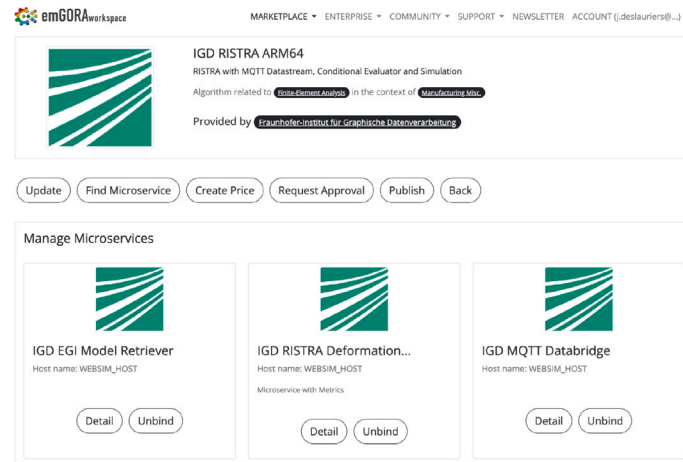


Fig. 11. View of Algorithm publishing in the Digital Agora.

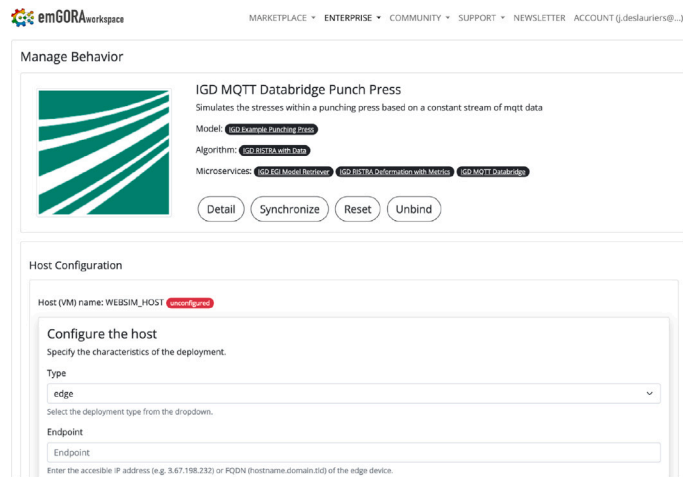


Fig. 12. View of DMA Tuple publishing in the Digital Agora.

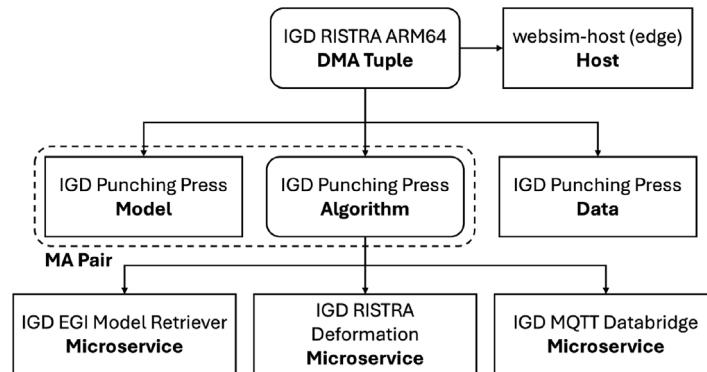


Fig. 13. Assets that make up the RISTRA DMA Tuple.

With all of the metadata for RISTRA provided, a new option becomes available in the DMA Tuple (here called Process) configuration, which will trigger its compilation (see Fig. 14).


Selecting this option will make a POST request to the ADT Generator REST API and the payload will contain the complete metadata of each Asset that is referenced within the DMA Tuple, from the MA Pair down to the Microservice.

The ADT Generator generates the CSAR as described in Section 4.1.2, stores it in an S3 bucket where it will be accessed at execution time. A download link to the CSAR file, as well as the logs


of the generated operation are displayed after successful compilation of the ADT.

Once the ADT has been compiled for a DMA Tuple, it is ready to be executed. A user within the organisational structure of the DMA composer can licence an instance of the DMA Tuple and launch it, as seen in Fig. 15. This gets deployed to EGI resources.

When the DMA Tuple is running, the users can interact with their application, accessing it via the endpoints configured in the Microservice and Host definitions. Advanced users have the option to visit a


emGORAworkspace

MARKETPLACE ▾
ENTERPRISE ▾
COMMUNITY



## IGD DataBridge Process ARM64

Process operated by **Fraunhofer-Institut für Graphische Datenverarbeitung**

Update
Find Equipment
Find Data
Find Behavior
Recompile Process

### Process Compilation v0

Note: If the metadata changed, synchronize or reset the behavior and recompile the process


ADT file (available during development only): **dima\_adt.csar (size: 76.0 KB)**

```

Log of generating CSAR archive based on Process metadata:
ADT generation process ID: 2024-04-01_13-37-19_318828
Storing incoming json starts...
Storing incoming json finished.
Validating incoming json starts...
Validating incoming json finished.
Storing assets in files starts...
Storing process...
Storing behaviour...

```

**Fig. 14.** Compilation of all Assets in a DMA Tuple in the Digital Agora.

emGORAworkspace

MARKETPLACE • ENTERPRISE • COMMUNITY • SUPPORT • NEWSLETTER • ACCOUNT (j.deslauniers@...)

## IGD DataBridge Process ARM64 v0 for Iltoter

[IGD DataBridge Process ARM64 v0 for Iltoter](#)

Licensed at Feb. 23, 2024, 5:19 p.m. by Iltoter and started at Feb. 23, 2024, 5:19 p.m. by Iltoter (last updated at Feb. 23, 2024, 5:29 p.m. by Iltoter)

Current estimated costs: 0.00

Relabel

Back

Status: Executing

Installing

Executing

Start

Update

Abort

Execution

The execution is actively running.

**Fig. 15.** An instance of the RISTRA DMA Tuple during execution.

**kubernetes** (pod out)

default Search + [notifications] [menu]

Cluster > Nodes

### Nodes

Name	Labels	Ready	CPU requests (cores)	CPU limits (cores)	CPU capacity (cores)	Memory requests (bytes)	Memory limits (bytes)	Memory capacity (bytes)	Pods	Created
deployment-websim-host	beta.kubernetes.io/arch: armv4 beta.kubernetes.io/instance-type: t3.xn1 beta.kubernetes.io/os: linux <a href="#">Show all</a>	True	0.00m (0.00%)	0.00m (0.00%)	8.00	0.00 (0.00%)	0.00 (0.00%)	30.28Gi	3 (2.73%)	35 minutes ago
micoad-control-plane	beta.kubernetes.io/arch: amd64 beta.kubernetes.io/instance-type: t3.xn1 beta.kubernetes.io/os: linux <a href="#">Show all</a>	True	200.00m (10.00%)	0.00m (0.00%)	2.00	140.00Mi (3.37%)	170.00Mi (4.34%)	3.88Gi	19 (17.27%)	9h ago

### Workloads > Pods

#### Pods

Name	Images	Labels	Node	Status	Restarts	CPU Usage (cores)	Memory Usage (bytes)	Created
microservice-condition-evaluator-4f9e9v48bc-7kxm	ignaciopeir/digitalbrain-condition-evaluator:latest	app.kubernetes.io/name: MICOAD APP app.kubernetes.io/namespace: micoad app.kubernetes.io/name: microservice-condition-evaluator <a href="#">Show all</a>	micoad-control-plane	Running	0	-	-	2 minutes ago
nista-deformation-service-cpu-549x4cdzfp-zr9qe	micoad/nista-deformation-service-cpu-armv4:latest	io.kompose.network/mgaod-default: true io.kompose.service/nista-service-cpu-armv4: true pod-template-hash: 549x4cdzfp	deployment-websim-host	Running	0	-	-	2 minutes ago
mpt-databridge-57x8d94kdc-fvggl	micoad/mpt-databridge-armv4:latest	io.kompose.network/mgaod-default: true io.kompose.service/mpt-databridge: true pod-template-hash: 57x8d94kdc	deployment-websim-host	Running	0	-	-	2 minutes ago
eg-modelinven-46tuzvdx-znuw	micoad/eg-modelinventor-armv4:latest	io.kompose.network/mgaod-default: true io.kompose.service/eg-modelinventor: true	deployment-websim-host	Running	0	-	-	2 minutes ago

**Fig. 16.** Nodes and pods of RISTRA application deployed on DIGITbrain.

dashboard where they can inspect information about the status of their application.

These users have access to Occopus and Kubernetes platform information, as well as a selection of metrics retrieved by Prometheus and displayed by Grafana. The connected edge node (deployment-websim-host) and running pods of the RISTRA application can be seen in the Kubernetes Dashboard in Fig. 16.

## 6. Conclusion and future work

When different technical experts collaborate to develop complex microservices-based applications for cloud and edge, an approach is needed to support that collaboration and ensure that whatever is described can be deployed. This paper has presented one such approach, that leverages important features within TOSCA to enable modularity and reusability throughout the entire process. The result is the automated generation of an overall application deployment descriptor that draws on individual descriptions of application components that can be authored by different actors at different times. These descriptions can be validated independently and the described components can have flexible relationships with one another when finally composed to the overall descriptor.

Advanced control over the run-time management of the various application components was not explored in this article, but TOSCA policies could be introduced into specific descriptors to give further fine-grained control. This approach remains relevant as TOSCA approaches v2.0, and hopes to leverage future features of the specification to further improve the automated generation of descriptors. Other future work will see this approach elaborated in the Horizon Europe Swarmchestrator project [39], where abstract Host nodes can be fulfilled not only by one specific concrete node, but intelligently matched to a best choice cloud or edge resource.

Additionally, while the current solution focuses on digital twins in the manufacturing sector only, the applicability of the approach has already been considered in other areas, such as transport, logistics, management of critical infrastructures or healthcare. Various funding proposals are currently under preparation to further investigate these possibilities.

Finally, we are considering extending the solution, supporting semantic checking and decision making when composing the digital twins. Utilising advanced AI techniques and already existing and additional metadata, a decision maker or decision support component could advise (or even substitute) the human decision maker when combining data, model and algorithm into a digital twin.

## CRedit authorship contribution statement

**James DesLauriers:** Writing – review & editing, Writing – original draft, Software, Investigation, Conceptualization. **Jozsef Kovacs:** Writing – review & editing, Writing – original draft, Software, Conceptualization. **Tamas Kiss:** Writing – original draft, Project administration, Funding acquisition, Conceptualization. **André Stork:** Writing – original draft, Project administration, Funding acquisition, Conceptualization. **Sebastian Pena Serna:** Writing – review & editing, Software, Project administration, Funding acquisition, Conceptualization. **Amjad Ullah:** Writing – original draft, Software.

## Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## Acknowledgements

This work was funded by the DIGITbrain – Digital twins bringing agility and innovation to manufacturing SMEs, by empowering a network of DIHs with an integrated digital platform that enables Manufacturing-as-a-Service – Project, No. 952071, European Commission, (EU H2020).

## Data availability

Data and code related to the ADTGenerator, MiCADO, and related technologies are open and available under the micado-scale organisation on GitHub.

## References

- [1] Amazon Web Services, Amazon CloudFormation templates, 2024, <https://aws.amazon.com/cloudformation/>. (Accessed: 24 May 2024).
- [2] Cloud Native Computing Foundation, Kubernetes, 2024, <https://kubernetes.io/>. (Accessed: 24 May 2024).
- [3] OASIS, TOSCA: Topology and orchestration specification for cloud applications v1.3, 2020, <https://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.3/TOSCA-Simple-Profile-YAML-v1.3.html>. (Accessed: 24 May 2024).
- [4] A.C. Marosi, M. Emodi, Á. Hajnal, R. Lovas, T. Kiss, V. Poser, J. Antony, S. Bergweiler, H. Hamzeh, J. Deslauriers, et al., Interoperable data analytics reference architectures empowering digital-twin-aided manufacturing, *Future Internet* 14 (4) (2022) 114.
- [5] G. Pierantoni, T. Kiss, G. Terstyanszky, J. DesLauriers, G. Gesmier, H.-V. Dang, Describing and processing topology and quality of service parameters of applications in the cloud, *J. Grid Comput.* 18 (2020) 761–778.
- [6] A. Ullah, H. Dagdeviren, R.C. Ariyattu, J. DesLauriers, T. Kiss, J. Bowden, Micado-edge: Towards an application-level orchestrator for the cloud-to-edge computing continuum, *J. Grid Comput.* 19 (4) (2021) 47.
- [7] O. Kopp, T. Binz, U. Breitenbücher, F. Leymann, Winery—a modeling tool for TOSCA-based cloud applications, in: *Service-Oriented Computing: 11th International Conference, ICSOC 2013, Berlin, Germany, December 2-5, 2013, Proceedings 11*, Springer, 2013, pp. 700–704.
- [8] S. Challita, F. Korte, J. Erbel, F. Zalila, J. Grabowski, P. Merle, Model-based cloud resource management with TOSCA and OCCI, *Softw. Syst. Model.* (2021) 1–23.
- [9] V. Yussupov, J. Soldani, U. Breitenbücher, F. Leymann, Standards-based modeling and deployment of serverless function orchestrations using BPMN and TOSCA, *Softw. - Pract. Exp.* 52 (6) (2022) 1454–1495.
- [10] M. Chinosi, A. Trombetta, BPMN: An introduction to the standard, *Comput. Stand. Interfaces* 34 (1) (2012) 124–134.
- [11] A. Bergmayr, U. Breitenbücher, O. Kopp, M. Wimmer, G. Kappel, F. Leymann, From architecture modeling to application provisioning for the cloud by combining UML and tosca, in: *CLOSER (2)*, 2016, pp. 97–108.
- [12] D. Calcaterra, O. Tomarchio, Automated generation of application management workflows using tosca policies, in: *CLOSER*, 2022, pp. 97–108.
- [13] L. Harzenetter, U. Breitenbücher, F. Leymann, K. Saatkamp, B. Weder, M. Wurster, Automated generation of management workflows for applications based on deployment models, in: *2019 IEEE 23rd International Enterprise Distributed Object Computing Conference, EDOC, IEEE*, 2019, pp. 216–225.
- [14] D. Calcaterra, O. Tomarchio, Policy-based holistic application management with bpmn and Tosca, *SN Comput. Sci.* 4 (3) (2023) 232.
- [15] O. Tomarchio, D. Calcaterra, G. Di Modica, P. Mazzaglia, Torch: a Tosca-based orchestrator of multi-cloud containerised applications, *J. Grid Comput.* 19 (1) (2021) 5.
- [16] Puccini: Cloud topology management and deployment tools, 2024, <https://puccini.cloud/>. (Accessed: 24 May 2024).
- [17] M. Bogo, J. Soldani, D. Neri, A. Brogi, Component-aware orchestration of cloud-based enterprise applications, from TOSCA to docker and kubernetes, *Softw. - Pract. Exp.* 50 (9) (2020) 1793–1821.
- [18] M. Wurster, U. Breitenbücher, L. Harzenetter, F. Leymann, J. Soldani, V. Yussupov, TOSCA light: Bridging the gap between the TOSCA specification and production-ready deployment technologies, in: *CLOSER*, 2020, pp. 216–226.
- [19] M. Wurster, U. Breitenbücher, M. Falkenthal, C. Krieger, F. Leymann, K. Saatkamp, J. Soldani, The essential deployment metamodel: a systematic review of deployment automation technologies, *SICS Softw.-Intens. Cyber-Phys. Syst.* 35 (2020) 63–75.
- [20] S. Kehr, W. Blochinger, AUTOGENIC: Automated generation of self-configuring microservices, in: *CLOSER*, 2018, pp. 35–46.
- [21] P. Štefanič, M. Cigale, A.C. Jones, L. Knight, I. Taylor, C. Istrate, G. Suciu, A. Ulisses, V. Stankovski, S. Taherizadeh, et al., SWITCH workbench: A novel approach for the development and deployment of time-critical microservice-based cloud-native applications, *Future Gener. Comput. Syst.* 99 (2019) 197–212.

- [22] S.J. Taylor, T. Kiss, A. Anagnostou, G. Terstyanszky, P. Kacsuk, J. Costes, N. Fantini, The cloudsme simulation platform and its applications: A generic multi-cloud platform for developing and executing commercial cloud-based simulations, *Future Gener. Comput. Syst.* 88 (2018) 524–539, <http://dx.doi.org/10.1016/j.future.2018.06.006>.
- [23] D. Weber, A. Stork, C. Stahl, A. Collado, D. T., Computational cloud services and workflows for agile engineering, in: *European Project Space on Information and Communication Systems - EPS Barcelona*, SciTEPress, 2014, pp. 71–88, <http://dx.doi.org/10.5220/0006183300710088>.
- [24] S.J. Taylor, A. Anagnostou, T. Kiss, G. Terstyanszky, P. Kacsuk, N. Fantini, D. Lakehal, J. Costes, Enabling cloud-based computational fluid dynamics with a platform as a service solution, *IEEE Trans. Ind. Inform.* 15 (2019) 85–94, <http://dx.doi.org/10.1109/TII.2018.2849558>.
- [25] T. Kiss, A cloud/HPC platform and marketplace for manufacturing SMEs, in: *11th International Workshop on Science Gateways, IWSG 2019*, Ljubljana, Slovenia 12–14 Jun 2019, 2019, <https://westminsterresearch.westminster.ac.uk/item/qv2xy/a-cloud-hpc-platform-and-marketplace-for-manufacturing-smes>.
- [26] DIGITbrain Project, Digitbrain project documentation, 2024, <https://digitbrain.github.io/>. (Accessed: 24 May 2024).
- [27] A. Stork, et al., DIGITbrain 2nd open call - short technical description, 2024, <https://digitbrain.eu/open-calls/#overview-2OC>. (Accessed: 24 May 2024).
- [28] V. Zambrano, J. Mueller-Roemer, M. Sandberg, P. Talasila, D. Zanin, P.G. Larsen, E. Loeschner, W. Thronicke, D. Pietrarola, G. Landolfi, et al., Industrial digitalization in the industry 4.0 era: Classification, reuse and authoring of digital models on digital twin platforms, *Array* 14 (2022) 100176.
- [29] T. Kiss, P. Kacsuk, J. Kovacs, B. Rakoczi, A. Hajnal, A. Farkas, G. Gesmier, G. Terstyanszky, Micado—microservice-based cloud application-level dynamic orchestrator, *Future Gener. Comput. Syst.* 94 (2019) 937–946.
- [30] J. Kovacs, P. Kacsuk, Occopus: a multi-cloud orchestrator to deploy and manage complex scientific infrastructures, *J. Grid Comput.* 16 (2018) 19–37.
- [31] D.F. Leymann, CSAR - cloud service archive V0.1, 2012, [https://groups.oasis-open.org/higherlogic/ws/public/document?document\\_id=46057](https://groups.oasis-open.org/higherlogic/ws/public/document?document_id=46057). (Accessed: 24 May 2024).
- [32] J. Kovacs, J. DesLauriers, D. Kagiialis, R. Arjun, H. Hamzeh, ADTGenerator, 2024, <http://dx.doi.org/10.5281/zenodo.11295968>, Zenodo.
- [33] J. DesLauriers, R. Arjun, DockKubeADT, 2024, <http://dx.doi.org/10.5281/zenodo.11295944>, Zenodo.
- [34] The MiCADO Parser Authors, Micado parser in github, 2024, <https://github.com/micado-scale/micado-parser/>. (Accessed: 03 June 2024).
- [35] The TOSCA Parser Authors, TOSCA parser in github, 2024, <https://github.com/openstack/tosca-parser/>. (Accessed: 03 June 2024).
- [36] The Kubernetes Authors, Kompose: Convert your docker compose file to kubernetes or OpenShift, 2024, <https://kompose.io/>. (Accessed: 24 May 2024).
- [37] N. Piatkowski, J.S. Mueller-Roemer, P. Hasse, A. Bachorek, T. Werner, P. Birnstill, A. Morgenstern, L. Stobbe, Generative machine learning for resource-aware 5G and IoT systems, in: *2021 IEEE International Conference on Communications Workshops, ICC Workshops, IEEE, 2021*, pp. 1–6.
- [38] The DIGITbrain consortium, DIGITbrain experiments webpage, 2023, <https://www.digitbrain.eu/experiments/>. (Accessed: 30 September 2024).
- [39] T. Kiss, A. Ullah, G. Terstyanszky, O. Kao, S. Becker, Y. Verginadis, A. Michalas, V. Stankovski, A. Kertesz, E. Ricci, et al., Swarmchestrator: Towards a fully decentralised framework for orchestrating applications in the cloud-to-edge continuum, in: *International Conference on Advanced Information Networking and Applications*, Springer, 2024, pp. 89–100.



**James DesLauriers** is a Research Fellow with the Research Centre for Parallel Computing at the University of Westminster, and a Teaching Fellow and AI Futurist in Education at Imperial College London. His research interests are focused on applying cloud and edge orchestration solutions in various fields, including securing cloud-enabled applications in healthcare, deploying digital twins to enable Manufacturing-as-a-Service, and novel work in applications of swarm computing.



**Jozsef Kovacs** is a Senior Research Fellow at the Laboratory of Parallel and Distributed Systems (LPDS) at the Institute for Computer Science and Control (SZTAKI), Hungarian Research Network (HUN-REN) and part-time Senior Research Fellow at Centre for Parallel Computing (CPC) at the University of Westminster (UOW). He got his B.Sc. (1997), M.Sc. (2001) and Ph.D. (2008) in the field of parallel computing. His early research topics were parallel debugging and checkpointing, clusters, grids and desktop grid systems, web portals. Recently, he has been focusing on



**Tamas Kiss** is a Professor of Distributed Computing at the School of Computer Science and Engineering, Director of the Research Centre for Parallel Computing and Director of Research and Knowledge Exchange at the School of Computer Science and Engineering. He holds a Ph.D. in Distributed Computing, and M.Sc. Degrees in Mathematics and Computer Science, and Electrical Engineering. He has been leading national and European research projects related to cloud to edge orchestration and enterprise applications of cloud computing technologies, especially in the areas of manufacturing, healthcare and scientific applications. He has been involved in more than 20 European and UK funded research projects as principal investigator or co-investigator.



**André Stork** is Head of Division at Fraunhofer Institute for Computer Graphics Research, Darmstadt, 64283, Darmstadt, Germany, and an honorary professor with Technical University Darmstadt. His major research interests include geometry modelling and shape processing, 2-D/3-D interaction techniques, simulation, digital twins and scientific visualisation. He is a member of IEEE, ACM SIGGRAPH, Eurographics, Gesellschaft fur Informatik, and VDI. From January 2023 until May 2024, he was Editor-in-Chief (EIC) of IEEE Computer Graphics and Applications and now he is member of the Advisory Board. Contact him at [andre.stork@igd.fraunhofer.de](mailto:andre.stork@igd.fraunhofer.de)



**Sebastian Pena Serna** founded clesgo GmbH in 2016, a startup facilitating the democratisation of ICT technology for the manufacturing sector. Sebastian is a mechanical engineer, finalising his Ph.D. at the TU Darmstadt. He worked with ESI Group in the position of Domain Lead Geometry and he was the deputy head of the Competence Center "Interactive Engineering Technologies" at Fraunhofer IGD.



**Amjad Ullah** is a Lecturer at the School of Computing, Engineering and the Built Environment at Edinburgh Napier University, Scotland, UK. He is also a research associate at the Centre for Parallel Computing (CPC) at the University of Westminster (UOW). He received his M.Sc. in Advanced Distributed Systems degree from the University of Leicester, UK, in 2011 and his Ph.D. in Computer Science from the University of Stirling, UK in 2017. His research interests include Orchestration and run-time management of applications in the cloud and the Cloud-Edge compute continuum, Resource provisioning, management and optimisation; Auto-scaling (Horizontal and vertical elasticity); Performance-based scaling policies; Deadline-based scaling policies to support batch-based applications in the cloud environment; and Computational offloading in edge computing.