OpenDT: A Reference Framework for Service Publication and Discovery using Remote Programmable Digital Twins

Md Rakib Shahriar
University of Florida
Gainesville, USA
shahriar.mdrakib@ufl.edu

Xiaoqing "Frank" Liu
Southern Illinois University
Carbondale, USA
xiaoqing.liu@siu.edu

Md Mahfuzer Rahman
University of Arkansas
Fayetteville, USA
mmr014@uark.edu

S M Nahian Al Sunny
University of Arkansas
Fayetteville, USA
smsunny@uark.edu

***Abstract*— Traditional service registries or catalogs publish and describe individual services of different entities. In this paper, we propose a new approach of service publication and discovery based on the philosophy of "product catalogs". In this new approach, entities are equivalent to products in typical products catalogs. We conceptualize an entity registry where each entry constitutes to its collection of remote services. For logical abstraction of entities, we utilize the concept of Digital Twin (DT). To support our objective, we present a reference DT architecture that virtualizes entities, exposes all its functionalities as services, and offers a remote programmable instance to invoke the services directly from application code. To publish and discover DTs, we propose a novel framework, OpenDT. This framework enables entity owners to publish DTs and allow users to discover them for creating mashups and applications using DT services. It also allows developers to create composite DTs that consists of other DTs for large and complex entities. To evaluate OpenDT, we implement a cyber-manufacturing testbed comprising of multiple machining tools and their DTs. Case validations from testbed show excellent efficiency of DT-driven entity publication and discovery.**

Keywords- Digital Twin; Services; Publication; Discovery; Registry; Catalogs; OpenDT

I. INTRODUCTION

Traditional service registries or catalogs act as central sources of digital services where organizations publish individual functionalities of their entities as remotely accessible services and manage them in a way so that users can search and discover them [1]. UDDI ((Universal Description Discovery and Integration) based service publication is one of the traditional approaches that describes individual services of entities and creates registries in service repositories. However, publishing each service separately overcrowds service registries, which potentially impedes service discovery performance [2, 3]. Hence, in this paper, we investigate a more integral approach of service publication. We envision an approach akin to the philosophy of *"product catalogs"*, where entities are equivalent to products and they are published with their collection of functionalities. Hence, the key difference between this new approach and the UDDI-based service publication methods is to publish services collectively in lieu of individually.

Publishing Application Programming Interface (API) is another approach of service publication. ProgrammableWeb (www.programmableweb.com) is one of such platforms where different organizations can publish their services as APIs to enable public access to their entities. This platform stores and indexes each of the services and APIs in service registry with different search tags. For example, search key '*Mercedes-Benz*' returned a list of 11 different APIs, which consist of many web services for different functionalities such as remote vehicular diagnostics support, fuel status monitoring, automotive configurations etc. To create a monitoring and diagnostics application for specific car models, one has to discover and subscribe web services across these 11 APIs. As these web services are distributed across all these APIs, application developer has to access these APIs for discovery and access. Our argument is publishing collection of services for each car models will improve discovery performance in such application contexts.

Based on the above contexts and reasonings, we propose a novel service publication approach that creates common virtual representations of heterogeneous entities. Each of these virtual representations creates an opportunity to unite all essential services of an entity into one package. Thus, a user can subscribe all services of an entity from a single entry of traditional service registry. To create such abstractions of entities, we recognize three fundamental requirements. First, virtual representations need to be uniform for heterogeneous entities. Second, these abstractions must have service-oriented architectures (SoA) to accommodate basic features of service computing. Third, they must offer resemblances to the structures of original entities.

Considering these requirements, we identify Digital Twin (DT) as a primary candidate to create homogeneous virtual representations of heterogeneous entities. DT is a rapidly growing concept in the domain of smart manufacturing, which creates virtual *as-is* copies of entities and offer entity functionalities as services [4]. We utilize this concept of DT and re-design it for a more general purpose. We present a new service-oriented architecture to satisfy the three above requirements that creates virtual representations of entities. Proposed DT design documents all remote services of an entity into a combined UDDI-based service descriptor. This service descriptor functions as a service collection. Proposed DTs represent instances of both physical and logical entities. They also capable of abstracting original structures of corresponding entities. These DTs are designed to be accessible through remote programmable instances in order to enable their easy programmability. Towards the objectives of this paper, we propose a novel service-oriented DT

2474-2473/20/$31.00 ©2020 IEEE
DOI 10.1109/SCC49832.2020.00024

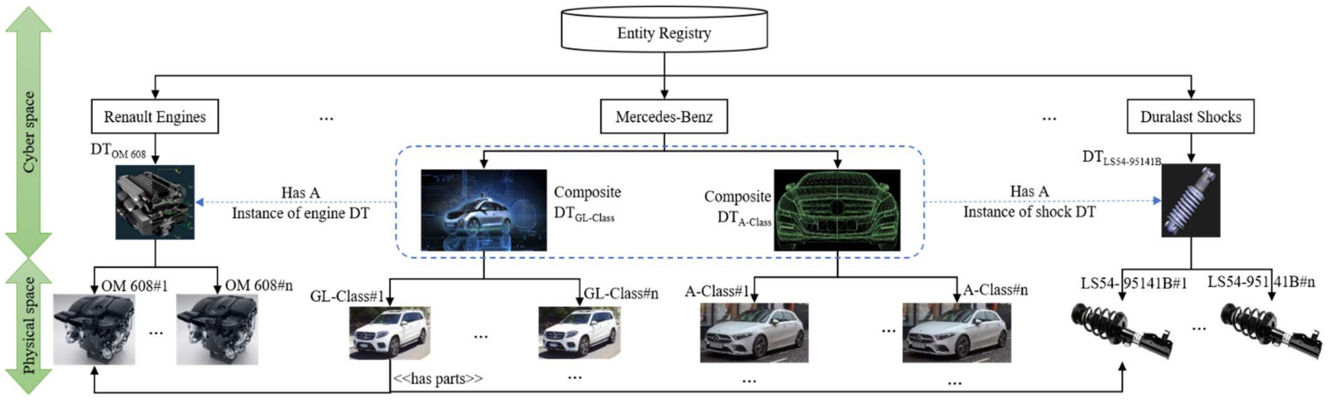

Fig. 1: A motivating scenario to describe the new publication approach based on the idea of product catalogs.

publication and discovery framework, namely OpenDT. Entity owners publish their DTs in OpenDT. Users can subscribe and access published DTs in their mashups and applications. In reality, a product can be combined of multiple sub-products. This resemblance is achievable in this proposed framework. As the published DTs are remotely programmable and their services are collectively accessible from instances, one can easily create a new DT by compositing multiple other DTs.

The followings are our research contributions of this paper. (i) We develop a new service publication approach based on the concept of *"product catalogs"*. In this new approach, entity functionalities are published as a service bundle. (ii) We identify and choose DT as the candidate tool to package collection of entity services and represent them in virtual space. To underpin this new role, we present a SoA of DT. Proposed architecture offers remotely programmable DT instances. (iii) We design an open framework, OpenDT, where entity owners can publish their DT and users can discover them.

II. MOTIVATION

We begin with presenting a motivating scenario to contextualize the proposed publication and discovery approach. Based on the motivating scenario, we describe the necessity of the proposed approach. Finally, we address the conceptual and technical requirements of the solution framework.

A. Motivating Scenario

Fig. 1 depicts a motivating scenario to conceptualize the new publication approach that is described by the following.

Auto manufacturer Mercedes-Benz manufactures different models of cars. Each car comes with common components such as engine, brakes, absorber shocks, struts etc. Mercedes-Benz does not manufacture all of these parts. Some of these come from different other manufacturers. For example, Renault S.A. produces engines (model OM 608) for different models of Mercedes Benz car models such as A-Class, GL-Class, and others. Similarly, automotive manufacturer Duralast produces shocks (model LS54-95141B) for these cars. These automotive parts are equipped with IoT sensors and their status data can be collected by readings from Controller Area Network (CAN) busses of cars. We assume for this case that these sensor data readings are limited to the Mercedes-Benz CAN busses because they come from different organizations. Mercedes-Benz publishes several APIs in a service registry for collecting vehicular data, so that third party application developers can develop mashups for different purposes such as monitoring, diagnosis, sales, etc. Renault and Duralast publishes APIs for their products, which offer more information than the CAN busses of cars.

We now live in an era where almost every business entity is going through digital transformations using web services. Emergence of IoT and CPS excelled the digitalization through service-oriented monitoring and control of physical things. In addition, intense digitalization of physical processes producing a technical universe of interconnected things or entities. Demonstrated scenario of Fig. 1 is a practical example of automotive entities, where a product (GL-Class) consists of many other sub-products (OM 608 & *LS54-95141B*) from different organizations. To create a mashup application for remote automotive monitoring and diagnosis, developers need to subscribe services from Mercedes-Benz, Renault, Duralast and others. In traditional UDDI-based service registries, these remote services are generally published individually. In proposed OpenDT framework, each of these product models are published as multi-tenant DTs and offer their collection of services. Users access the DT instances, which comes with DT descriptors. Users discover suitable services from these descriptors from their application code.

B. Need for a novel product-oriented publication approach

We consider the case where the aforementioned remote monitoring and diagnosis application is developed using traditional UDDI-based service publication approaches. We identify two issues in this case.

First, we point out that there is a mismatch between involved real entities and their logical representations. If a car (GL-Class) in Fig. 1 is observed from object-oriented design (OOD) approach, we can see that it consists of many other parts such as Renault OM 608, Duralast LS54-95141B, etc.

Therefore, an OOD of GL-Class model should consist of instances of those part models. Such an OOD is absent if the application is developed using traditional service registries. This issue is particularly problematic in the cases of IoT and CPS-based entities where resemblances of logical and physical representations are a priority.

Second, we identify a difficulty in runtime discovery of remote services from other entities. When a diagnosis application measures car health, it needs to access remote services of different parts. These services are likely to come from Original Equipment Manufacturers (OEM). Hence, the application needs to discover these OEM provided services, which warrants additional remote service calls for discovery and can impede application runtime performance.

These issues can be resolved using the proposed publication approach of Fig. 1. First, Renault will create a multi-tenant DT for all engines of model OM 608, denoted by DT_{OM608}. They will publish the created DT in a service registry, where other companies also publish their DTs. Respectively, Duralast will establish $DT_{LS54-95141B}$. Mercedes Benz will create two multi-tenant DTs namely DT_A and DT_{GL} for all car models of A and GL respectively. As car models are products with sub-products like engines and shocks, both DT_A and DT_{GL} will have personalized instances of DT_{OM608} and $DT_{LS54-95141B}$.

If the monitoring and diagnosis application is developed with this new approach of DT publication, one can access the remote programmable DTs of DT_A and DT_{GL}. These DTs resemble original ontology structures of the corresponding car models. In addition, instances of DT_A and DT_{GL} internally refer to instances of DT_{OM608} and $DT_{LS54-95141B}$. Therefore, by accessing remote programmable instances of DT_A and DT_{GL}, one can discover all available services of large and complex entities locally from application code.

At this point, it is also necessary to answer the following question - *why publishing entities as DTs?* There are many different ways that entities can be logically abstracted in virtual spaces. One of the widely popular approaches is REST protocol, which takes an object-oriented approach to transform entities as logical resources [5]. But statelessness of RESTful representation is not appropriate in many cases of CPS and IoT enabled physical entities [1, 2]. In comparison, concept of DTs can provide a better logical abstraction for wider ranges of heterogeneous entities. Another study in [6] inspires us to envision a DT-driven service publication where researchers have recognized homogeneity as a key property of DTs for heterogeneous machining entities. We also take ideas from researches in [4, 7, 8] where concept of DTs were efficiently applied for software-defined representations of heterogeneous entities.

C. Basic requirements of the proposed approach

We address a set of fundamental requirements towards developing the theoretical approach of DT-driven service publication and discovery. These requirements are set with a motivation to expand on top of the existing approaches. We divide the requirements into following two prongs.

1) Requirements of OpenDT

Service registration using DTs: Our approach aims to reduce the number of entries in service registries by publishing entities as DTs. While publishing DTs can be performed using existing trivial schema designs as proposed in [9], it is necessary to specify other DT related information that needs to be incorporated in the registry.

Service discovery using DTs: Proposed approach warrants service discovery in multiple steps through DTs. Hence, it is necessary to design and develop a DT service discovery mechanism that enables a user to discover both entities and their functionalities.

Query model of DTs: In order to discover DT services, it is necessary to design a query data model. Objective query structure needs to incorporate information for searching specific entities based on functional and non-functional requirements. It also needs to incorporate information to find suitable entity functionalities through published DTs.

Composition mechanism of DTs: To support existing requirements of traditional service composition and mashup methods, it is necessary to underline a composition mechanism for DTs and their services. Objective mechanism needs to address different scenarios of composition operations. One particular scenario is composition of multiple DT services in applications and mashups. Another scenario is composition of multiple DTs into a new DT.

2) Requirements of DTs

SoA of DTs: Many existing architectures of DTs focus to virtually represent physical entities for different purposes such as remote monitoring, visualizations, simulations, controls, predictive analytics, etc. But these architectures do not satisfy the needs of our proposed approach. Hence, it is necessary to design a more general SoA of DTs, so that it can represent entities beyond the domains of CPS and IoT.

Instantiation of multi-tenant DTs: Clearly our proposed approach welcomes to publish a plethora of DTs. As users will access instances of these DTs, it is necessary to design DTs as a multi-tenant virtual entity that offers parallelly accessible instances. It is also necessary to specify about lifecycle of DT instances.

Remote programmability of DTs: According to the proposed approach, a DT can be consisted of multiple sub-DTs. Hence, DTs will require to offer remotely programmable instances so that one can programmatically access and combine them into composite DTs.

Collective DT service descriptor design: Proposed approach takes a collective approach to publish entity functionalities as DT-services. Hence, it is necessary to design a DT descriptor model that describes both functional and non-functional information of heterogeneous entities. In addition, many CPS and IoT enabled entities will involve chronologically dynamic properties, which also should be accommodated in the descriptor design.

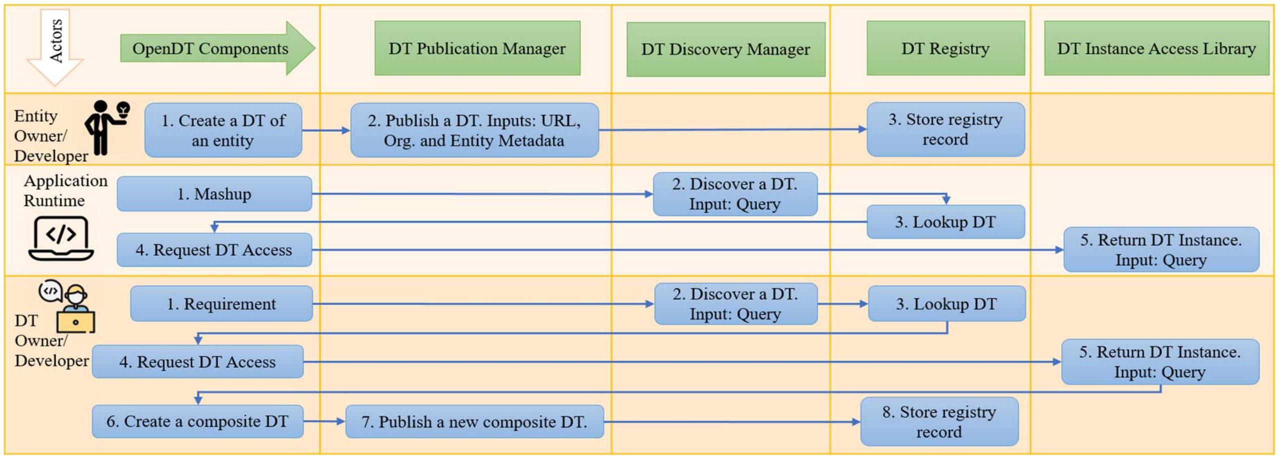

Fig. 2: A high-level view on OpenDT framework. This diagram presents basic operations of OpenDT such as DT publication, registry, and discovery.

Interoperability of DTs: Interoperability is one of the key properties of service computing. While accessing remote programmable DT instances create easier programmability, a concern raises about loss of service interoperability. To mitigate such concern, DTs will need to offer options to access services, both through DT instances and traditional service calls through APIs.

III. Reference Framework of OpenDT

Towards the research goals presented earlier, we design a framework, namely OpenDT, which establishes a publication and discovery platform based on the idea of *product catalogs*. Fig. 2 depicts an overall picture of the framework by showing interactions among different actors and components. Rest of the section organizes as follows. First, the general approach of the OpenDT framework is detailed. To support the framework, a service-oriented DT architecture is discussed and formalized. This section also discusses a DT descriptor model that provides the foundation of service bundles within a DT. Last but not the least, a query architecture for DT service discovery is presented.

A. Framework design

OpenDT diagram in Fig. 2 depicts a general approach and an overall workflow for publication, discovery and remote programmability of service-oriented DTs. Following sub-sections discuss the approach by describing different actors, components and their basic operations in the framework.

1) Actors

We consider three types of actors in OpenDT. Entity Owners create DTs for their entities and publish them using different components of the framework. Published DTs are discoverable by other users for use. Third party developers can discover and access DT services from code and create applications and mashups for general users. During runtime, application code can dynamically discover and access the published DTs. Developers from different organizations can discover DTs, which are parts of their products. They can composite discovered DTs to create new composite DTs.

2) Components

DT Publication Manager (DTPM) component offers services where entity owners can publish their DTs. There are existing service publication platforms, such as ProgrammableWeb. This DTPM module functions similar to that platform. However, the fundamental difference is DTPM publishes DTs instead of web services and APIs.

DT Registry (DTR) component stores registry information produced by DTPM component. This component contains a database for the registry entries. It also keeps necessary descriptive and metadata of DTs.

DT Discovery Manager (DTDM) component allows users to search for suitable DTs and their services. When searching DTs in runtime, this module discovers list of suitable DTs and shows them to the requesting mashup or application.

DT Instance Access Library (DTIAL) component offers a plugin that is a pre-compiled library for mashup and application developers. This precompiled library channels a communication between OpenDT framework and applications. Upon request for a specific DT, this plugin component retrieves the reusable DT instance, which is programmable in the application code.

3) Basic operations

Among many basic operations of service computing, we consider three, which are fundamental to the research contributions of the proposed reference framework. These also help to describe the general approach of OpenDT.

DT Publication: In order to publish, entity owners submit the endpoint of a DT with descriptive information and metadata. Upon submitting, DTPM component probes the DTs and automatically generate entry for the DTR component. Finally, DTR stores the publication information into its data store.

Runtime discovery of DTs: Runtime discovery cases generally occur from mashups and applications. Mashups and applications configure a query and send it to the DTDM component. DTDM analyzes the query information and preprocess it for DTR to discover the suitable DTs. DTR component returns URLs of the DTs so that they can be requested for access to DTIAL component.

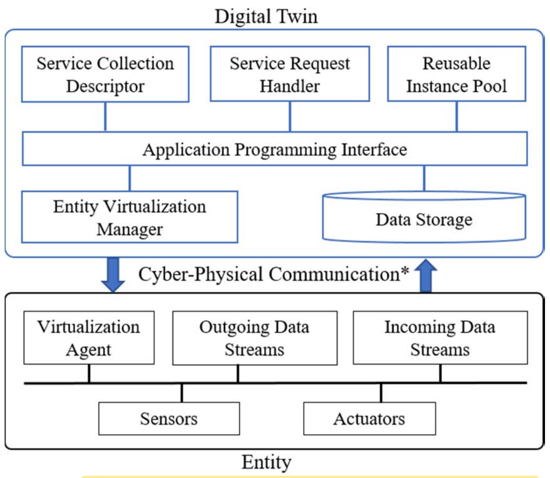

Fig. 3: A service-oriented architecture of DT. Cyber-Physical Communication* is optional and particularly applicable for CPS and IoT.

Compositing a new DT: In previous sections, we described the approach of creating composite DTs of entities, which are consisted of several other DTs of sub-entities. Hence, we describe a unique composition operation in OpenDT. Developers from different organizations can discover the sub-entities, which construct their products and get necessary DT instances upon request to DTIAL. These DT instances are programmed into a new composite DTs for new entities and published into DTR. This way OpenDT establishes a network of interconnected DTs.

B. Service-oriented architecture of DTs

Fig. 3 presents a service-oriented architecture of DTs for OpenDT. We divide the diagram into two parts such as the entity, and its representational DT in virtual space. An entity can be entirely logical where the corresponding DT performs simulation of it. On the other hand, an entity can be physical, particularly in the contexts of CPS and IoT enabled entities, where they integrate computational and networking capabilities and establish cyber-physical concurrent synchronizations. In such contexts, they exchange real-time data back and forth with the DT [8]. Data exchange from entity to DT corresponds to real-time monitoring data collection of physical processes. Inversely, DT to entity data exchange resembles remote service-based execution of entity functionalities. The following sections formalizes a general concept of entities and their corresponding DTs.

The proposed architectural system, denoted by S_{DT}, consists of the following subsystems and components such as set of entities EN, their DTs DT, Entity Virtualization Manager V, Service Collection Descriptor Svc, Reusable Instance Pool Lib. The following expression formalizes the architectural system.

$$S_{DT} = \{EN, DT, V, Svc, Lib\} \qquad (1)$$

1) Entity

Physical components of an entity can comprise of other entities, which consist of one or multiple devices. The devices have Sensors and Actuators. All of these elements are virtualized using the Virtualization Agent component. For simulation-based DTs, input data can be collected offline. In the contexts of DTs with multi-physics property, two components of the proposed architecture take responsibilities of data transmission. Outgoing Data Streams component gathers status information from the physical components and transmit to DT. Inversely, Incoming Data Streams component accepts and routes requests for entity functionality execution.

Here, a formal presentation of CPS and IoT enabled entities is constructed. We adopt the CPS based design of physical objects in our design [7]. We consider that one entity can consist of one or more smaller entities. An entity, denoted by en $\in EN$, is comprised of the following eight elements such as sensors S_e, actuators A_e, Incoming Data Streams C_e, set of events E_e, Outgoing Data Streams D_e, and virtualization agent V_e. The following expression formalizes

$$en = \{S_e, A_e, C_e, E_e, D_e, V_e\} \qquad (2)$$

Considering the formation using things, we formalize entity by the following equation.

$$EN = \{en_i, where\ i = 1 \ldots |EN|\} \qquad (3)$$

2) DT

Entity Virtualization Manager component defines the business logics, endpoints of functionalities and semantic structures of virtualized entities. The entire virtual representation of an entity relies on this component. It organizes and stores data in the Data Storage component. We consider the Data Storage component as a single multi-tenant database, which facilitates DT multitenancy. For simplicity of design, we assume that it is scalable like the traditional big data storages. To construct a DT, entity owner needs to publish its functionalities as services, which can be done using traditional approach of APIs. Thus, we place API as a component within the DT part. API uses Entity Virtualization Manager and Data Storage components to virtualize entity functionalities and characteristics as services. Service Collection Descriptor module stores an UDDI-based document that describes the collection of services hosted in the API component. Keeping the API component also enhances interoperability of DTs.

Service Request Handler component receives incoming service requests from applications and mashups. Afterwards it validates and forwards them to the API component. While routing incoming requests, we consider a design that accepts dot notation based unique service identifiers. The dotted service identifiers reflect semantic hierarchies of entities.

Reusable Instance Pool component enables remote programmability of DTs. This component creates a representational instance of DTs upon request from DTIAL component of OpenDT framework. For simplicity of design, we assume an infinite lifecycle of DT instances. Fig. 4 (a) depicts a programming interface of DT instances that application and mashups can access from code.

In this paper, we consider two structural patterns of DTs. First, a DT represents an entire entity such as Renault engine DTs from the motivating scenario from Section II. Second, a

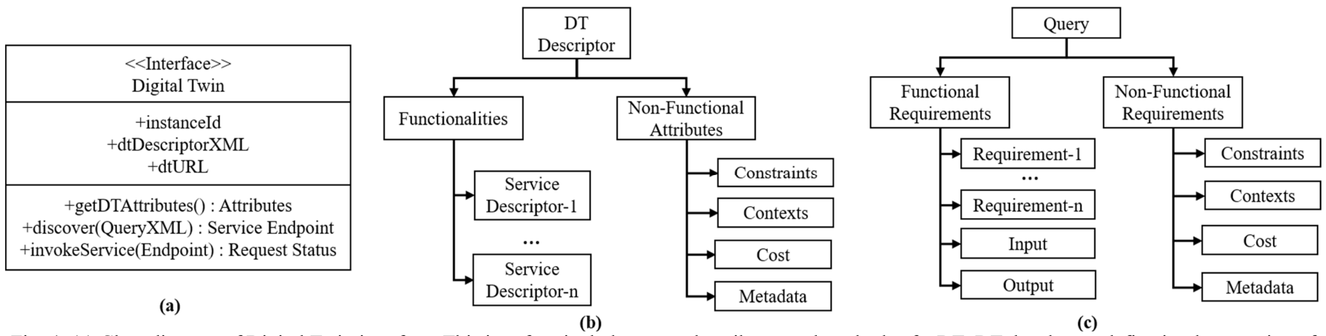

Fig. 4: (a) Class diagram of Digital Twin interface. This interface includes general attributes and methods of a DT. DT developers define implementation of this interface for use of applications and mashups. (b) Information model of DT descriptor. (c) Query data hierarchy in OpenDT.

DT comprises of multiple sub-product DTs such as Mercedes-Benz cars. Hence, in our formalization, we consider a DT, denoted by DT, consists of one or more sub-ordinate DTs, dt, such that $dt \in DT$. The following equation defines the construction of DT.

$$DT = \{dt_i \; where \; i = 1 \ldots |DT|\} \qquad (4)$$

C. Service Collection Descriptor model

In this section, we discuss Service Collection Descriptor component and present its information model. We construct an information hierarchy for the proposed Service Descriptor module, denoted by SD_{dt}. A descriptor document SD_{dt} comprises of a collection of remote services. According to our proposed design, each DT service corresponds to a functionality of an entity. Fig. 4 (b) outline an information model for DT descriptors.

We formalize the outlined DT descriptor model. We consider remote services, denoted by s, consist of three basic information such as input parameters description (in), functional description ($desc$), and output description (out). Hence, $s = \{in, desc, out\}$. Hence, collection of services in a DT can be expressed as $Svc = \{s_i \, , where \; i = 1 \ldots |svc|\}$.

DT developers can combine these remote services to create mashups and composite DTs. A mashup service, denoted by ms and transitions between s_i and s_{i+1} are denoted by δ_i, combines multiple svc, can be expressed as $ms = \left\{\prod_{i\geq 1}^{i\leq |svc|} s_i\right\} \cup \{\delta_i, where \; 1 \leq i \leq |ms| - 1\}$. Note that, δ_i is only applicable to ms when output of s_i is sequentially fed to any $s_{j\neq i}$. All mashup services in a composite DT can be expressed by $MS = \{ms_i, where \; i = 1 \ldots |MS|\}$. Therefore, we can further formalize Svc of a composite DT as $Svc_{composite} = \{Svc, MS\}$.

D. DT Query model

Query data model is a widely researched topic in service computing. For example, [10] presents a query data model for IoT enabled entities where they incorporate different contexts of service discovery. In this section, we take insights from the existing query models and redesign for OpenDT. Fig. 4 (c) depicts our proposed query model for OpenDT.

A query has two types of requirements such as functional requirements and non-functional requirements. Functional requirements consist of a requirement collection. These requirements specify both DT level requirements and service level requirements. Because in OpenDT, the general approach is to discover DTs of entities and then discover suitable services later. Non-functional requirements incorporate essential characteristics of entities such as constraints, contexts, operational costs, and other metadata about DTs. DT Discovery Manager component of OpenDT crossmatches these functional and non-functional requirements with Service Collection Descriptors of DTs for service recommendations.

IV. Implementation

A. Experimental setup

We map the motivating scenario of creating composite DTs for Mercedez-Benz cars to an alternative scenario of constructing composite DTs for Virtual Production Lines [11] consist of multiple DTs of machining tools. We have evaluated OpenDT using a testbed presented in Fig. 5. Its infrastructure includes five manufacturing machines such as two 3D printer (Ultimaker2 and Bukito), two robotic arms (Uarm), and one CNC cutting machine (XCarve). Each machine is virtualized to their corresponding DTs through their virtualization Agents. Virtualization method of this testbed was adopted from a previous research presented in [8]. All the implemented DTs provide hypermedia web services for near real-time 3D visualization. They also expose web services for remote monitoring and execution of manufacturing operations. Each of them offers XML-based DT Descriptors. One can call a web service in Remote Instance Pool component to access DT instances. A similar and comparable testbed was implemented in a prior work [12] using traditional UDDI-based service registry approaches by the same group of authors, namely the CPMC testbed. In this section, we contrast the differences of implementation between this new OpenDT testbed and the old CPMC testbed. To validate and discuss different cases of the testbed, we have developed multiple Java-based Single Page Applications (SPA), namely Ultimaker2, Bukito, XCarve, Composite-1, and Composite-2. In order to contrast results, each of these SPAs are separately implemented and experimented in both testbeds. In CPMC testbed, web services of individual functionalities of machine tools are recorded in a UDDI-

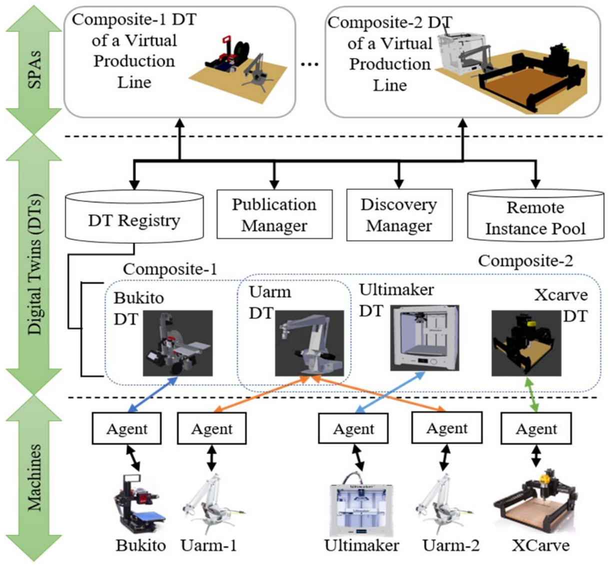

Fig. 5: Testbed structure for case validations. This testbed re-uses the machining tools from the CPMC testbed presented in *[8, 12]*.

based registry for monitoring and operations. For Ultimaker2 machine, there were 15 web services published in service registry. Similarly, there were 15, 13, and 6 web services published in service registry for Bukito, XCarve, and Uarm machines respectively. SPAs in CPMC testbed discovers individual manufacturing services from the registry whenever they needed to perform specific functionalities. For example, to print a 3D object model, first Ultimaker2 SPA will invoke the UDDI-based service registry module with search tags to return the web service for 3D printing operation. In the context of runtime discovery, it will invoke the web service for further machine execution.

In OpenDT testbed, SPAs use programmable DT instances from the implemented OpenDT framework. The first three SPAs offer remote machine monitoring by creating mashup of DT services accessed through DT instances. Composite-1 SPA accesses instances of Bukito DT and UARM DT and configures a new composite manufacturing entity that establishes an event-driven collaboration, where Bukito prints an object and requests Uarm-1 to move it to a storage. Similarly, Composite-2 SPA creates an event-driven collaborative manufacturing entity, where Ultimaker2 prints an object, requests Uarm-2 to ship it to XCarve for further drilling. SPAs of this testbed discover DT instances in runtime from DT Registry. DT instances provide service descriptor information as presented in Fig. 4 (b). This way the SAPs can use the DT instances to locally discover suitable services. This approach of implementation leads to avoid making repetitive calls for service discovery during runtime.

B. Initial evaluations

Evaluation metrics: Proposed framework is at the very early stage of development. Thus, we identify three evaluation metrics to perform initial assessment of OpenDT. We measured number of service registries in both of the testbed implementations to evaluate the publication approach. To evaluate easier programmability, we recorded lines of application code in the SPAs. Last but not the least, we logged number of remote service calls initiated in both of the testbed implementations to contrast improvement on runtime service discovery. Based on these metrics, we performed the following case validations.

Improvements in publication approach: We compared the number of registries created for SPAs in CPMC and OpenDT testbeds. In both of these testbeds, service registry database schema remains mostly identical. However, due to the publication approach, number of registries were reduced significantly in OpenDT testbed. For example, total number of publishable services in machines of Composite-2 DT sums to 34 where Ultimaker2, Bukito and XCarve contributes 15, 13, and 6 respectively. All of these services were needed to be published in CPMC testbed. Contrarily, in OpenDT framework, we needed to publish only 4 DTs for Composite-2, 1 for the composite DT and 3 for the collaborating DTs.

Improvements of programmability: We also drew a comparison of Line of Codes (LOC) among the SPAs developed in CPMC and OpenDT testbed. The SPAs did not need to import jQuery library to invoke DT services. Instead, they invoked services by the DT instances. Our observation shows that building SPAs by using the proposed DTs reduces 38%-53% LOC. Through pair programming by multiple peers, we identified easier programming with DT instances.

Reduction of discovery service traffic: In the context of the considered testbeds, we observed significant reduction of discovery service calls in OpenDT. This significant reduction on service operations was achieved because the DT instances already contained the service collection descriptor, which leads DT service discovery within the application codebase.

C. Comparisons with state-of-the-arts

Existing service publication and discovery approaches: There is no uniform service registration approach that satisfies all of the contexts of service computing. A survey in [13] presented different trade-offs in different architectural patterns and approaches of service registries. Our proposed approach falls into the service registry pattern where users or applications discover and access services through published virtual entities from a catalog of DTs. However, primary focus of this paper is to initiate a scientific research towards a new publication and discovery approach using DTs.

Researchers have conducted intense research on dynamic service registration methods. Baresi et al. [14] presented Distributed Registry by Example (DREAM) platform where heterogeneous entity owners can publish individual services. Publishing services individually has been a norm in state-of-the-arts, while researchers put significant efforts to improve service discoverability using dynamic service descriptions [15]. Another direction for improving service availability and discovery is through replications [16]. All of these state-of-the-arts provide formidable solutions for improved service registry models, but none of them touches the base of publishing collection of services as it is in our proposed approach. In the contexts of CPS and IoT enabled entities, researchers have proposed different publication and discovery frameworks based on numerous UDDI-based service registries. The study in [10] formalizes a context-aware approach for publication and discovery of IoT enabled entities. Contextual information is also accounted in [9] to discover services for mobile environments. However, none of these state-of-the-arts deviate from the typical individualistic publication approach. By contrast, we use DTs to logically abstract entities, which are capable of offering collection of functionalities as services and incorporate dynamic contextual information during discovery operations. Also, our proposed approach does not limit to domains of IoT and CPS. It can support a wider range of heterogeneous entities.

Service-oriented architectures of DTs: In 2002, Grieves introduced the concept of DT in the domain of manufacturing for production lifecycle management [4]. Since then, researchers have presented numerous definitions of DTs and different architectures for different domains. In this paper, we only compare with a few state-of-the-art service-oriented architectures. A SoA of DT was presented in [6] that focused on creating physical processes in smart manufacturing plants through a mediator module. Design of the mediator module addresses interoperability of DTs for the domain of manufacturing. In comparison, our DT architecture is tailored for a wider group of entities. In addition, our DT architecture addresses the requirements for a publication and discovery framework, which was not addressed in any of the state-of-the-arts to our knowledge. There are few prior SoA of DTs proposed in [7, 8]. Neither of these architectures proposed the conceptual approach of developing homogeneous DTs for heterogeneous entities. These two studies also do not address the general requirements of service computing.

V. Conclusion & Future Work

Service computing has been one of the most influential tools to integrate technology with different types of entities across domains. It is also instrumental in the process of digitizing physical objects and processes. Typically, an entity (physical or virtual) publishes its functionalities as remote services in service registries where each individual service creates an entry in the registry database. Therefore, typical service registries get overcrowded because of the ever-increasing number of individually published services. This issue potentially impedes discovery performance. In this paper, we take a new publication approach to reduce number of entries in service registries and improve discovery performance. We investigate a method where one entry in registry can publish all services of an entire entity. Towards this objective, we apply the concept of DT, which can virtually represent entities and offer a collection of services from one entry of registry. To support the proposed approach, we design a novel DT publication and discovery framework, namely OpenDT. We detail the framework and its essential components including a service-oriented architecture of DTs for heterogeneous entities. We validate the OpenDT reference framework using a prototype implementation. We experimented the testbed for multiple case validations, which clearly show feasibility and early potentials of the framework.

This research is at the very beginning stage and focuses on fundamental concept development and more scientific validation is needed for clearer results in future.

References

[1] W. J. Obidallah and B. Raahemi, "A survey on web service discovery approaches," in Proceedings of the Second International Conference on Internet of things, Data and Cloud Computing, 2017.

[2] H. Zorgati, R. B. Djemaa and I. A. B. Amor, "Service discovery techniques in Internet of Things: a survey," in 2019 IEEE International Conference on Systems, Man and Cybernetics (SMC), 2019.

[3] G. Shinde and H. Olesen, "A Survey on Service Discovery Mechanism," in International Conference on Intelligent Computing and Communication ICICC - 2017, 2018.

[4] M. Grieves and J. Vickers, "Digital twin: Mitigating unpredictable, undesirable emergent behavior in complex systems," in Transdisciplinary perspectives on complex systems, Springer, 2017, pp. 85-113.

[5] R. T. Fielding and R. N. Taylor, "Principled design of the modern Web architecture," ACM Transactions on Internet Technology, vol. 2, no. 2, pp. 115-150, 2002.

[6] T. Catarci, D. Firmani, F. Leotta, F. Mandreoli, M. Mecella and F. Sapio, "A Conceptual Architecture and Model for Smart Manufacturing Relying on Service-Based Digital Twins," in 2019 IEEE International Conference on Web Services (ICWS), 2019.

[7] K. M. Alam and A. El Saddik, "C2PS: A digital twin architecture reference model for the cloud-based cyber-physical systems," IEEE access, vol. 5, pp. 2050-2062, 2017.

[8] M. R. Shahriar, S. M. N. Al Sunny, X. Liu, M. C. Leu, L. Hu and N.-T. Nguyen, "MTComm based virtualization and integration of physical machine operations with digital-twins in cyber-physical manufacturing cloud," in 2018 5th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/2018 4th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom), 2018.

[9] R. Verma and A. Srivastava, "A dynamic web service registry framework for mobile environments," Peer-to-peer Networking and Applications, vol. 11, no. 3, pp. 409-430, 2018.

[10] N. Ibrahim, "Publication and Discovery of Things in the Internet of Things," EAI Endorsed Transactions on Internet of Things, vol. 4, no. 14, p. 156082, 2018.

[11] Y. Tang, M. Zhou and R. Qiu, "Virtual production lines design for back-end semiconductor manufacturing systems," IEEE Transactions on Semiconductor Manufacturing, vol. 16, no. 3, pp. 543-550, 2003.

[12] X. F. Liu, R. Shahriar, S. M. N. A. Sunny, M. C. Leu and L. Hu, "Cyber-physical manufacturing cloud: Architecture, virtualization, communication, and testbed," Journal of Manufacturing Systems, vol. 43, pp. 352-364, 2017.

[13] T. Aihkisalo and E. Ovaska, "Identifying Architectural Design Trade-Offs in Service Registry Features," in 2017 IEEE Symposium on Service-Oriented System Engineering (SOSE), 2017.

[14] L. Baresi, M. Miraz and P. Plebani, "A distributed architecture for efficient Web service discovery," service oriented computing and applications, vol. 10, no. 1, pp. 1-17, 2016.

[15] L. Braubach, K. Jander and A. Pokahr, "A novel distributed registry approach for efficient and resilient service discovery in megascale distributed systems," Computer Science and Information Systems, vol. 15, no. 3, pp. 751-774, 2018.

[16] A. Usman, P. Zhang and O. Theel, "A Highly Available Replicated Service Registry for Service Discovery in a Highly Dynamic Deployment Infrastructure," in 2018 IEEE International Conference on Services Computing (SCC), 2018.