Towards a Methodology for RAMI4.0 Service Design

Jorge Buenabad-Chávez*, Gabor Kecskemeti†, Vasilios Tountopoulos‡, Evangelia Kavakli*§, Rizos Sakellariou*
*School of Computer Science, The University of Manchester, Manchester, UK
†School of Computer Science, Liverpool John Moores University, Liverpool, UK
‡Athens Technology Center S.A., Chalandri, Greece
§Department of Cultural Technology and Communication, University of the Aegean, Mitilini, Greece
{jorge,evangelia.kavakli-2,rizos}@manchester.ac.uk,g.kecskemeti@ljmu.ac.uk,v.tountopoulos@atc.gr

***Abstract*—The Reference Architecture Model Industry 4.0 (RAMI4.0) proposes two models to guide the design of virtual representations of assets. A cubic model represents assets in the form of layers and allows them to be tracked over their lifetime and assigned to technical or organizational hierarchies. The I4.0 component model suggests how to organise virtual representations, i.e., asset properties (data and functions). A service hierarchy has also been proposed as the mechanism to gain access to asset properties. This paper presents an analysis of the RAMI4.0 service hierarchy, compared to traditional service oriented architecture, towards a methodology for the design of RAMI4.0 services. We envision a methodology based on object-oriented analysis and design principles. The rationale behind our approach is the similarity that holds between objects and assets as I4.0 components. Data and functions within objects are accessible through *method* invocations; data and functions of assets are accessible through *service* invocations. The paper outlines the use of this approach in defining RAMI4.0 services for a software system being designed to support decision making under events that disrupt manufacturing operations.**

***Index Terms*—Industry 4.0, RAMI4.0, Internet-of-Things, Smart Factories, Reference Architectures, Service-Oriented Architectures**

I. Introduction

The Internet of Things (IoT) is all about making our environments smarter and friendlier through ubiquitous computing. Cyber-physical systems (CPSs) are a key technology for IoT and are already being deployed in various settings — *smart homes* already allow appliances such as air conditioners to be controlled remotely through the Internet. A CPS consists of an embedded system (hardware and software), sensors, actuators, and a network access device [1]. Although embedded systems have been used since the advent of the microprocessor in the 1970s, their use until recently has mostly involved specific tasks to control individual devices, machines and processes [2]. CPSs can interact with their environment and with other CPSs to accomplish dynamically determined cooperative tasks, including collective self-adaptivity.

The vision of IoT in manufacturing, known as Industry 4.0 (I4.0), is automated vertical and horizontal integration including product lifecycle management (PLM) [3]. Most required technologies are now available [3]–[6]. The main issue is their integration through software [3]. Various reference architectures have been proposed to guide this integration and the transition into Industry 4.0 [7]–[11]. A reference architecture is a blueprint to guide the design of concrete software architectures in a particular domain. It has the twofold purpose of reducing design effort and ensuring interoperability between applications in the domain. A reference architecture's design is based on standards and complemented with supporting documents related to use cases to exemplify its applicability, best practices for development, and current technologies and products that may facilitate implementations. The Industrial Internet Reference Architecture [7], by the Industrial Internet Consortium[1], provides guidance for the entire development process of industrial IoT systems (in Energy, Healthcare and Manufacturing, among other domains): from the evaluation of business benefits and use cases analysis up to design and deployment of a *generic* software infrastructure.

The Reference Architecture Model Industry 4.0 [8], or RAMI4.0, by the German initiative Platform Industry 4.0[2], proposes a cubic layer model and the I4.0 Component model to guide the design of virtual/digital representations of assets. The I4.0 component model "constitutes a specific case of a cyber-physical system" [12]. An I4.0 Component comprises an asset and a (software) wrapper, called Administration Shell, that enables remote access to the properties (data and functions) of the asset. Interaction with, and between, I4.0 components is to be based on a service hierarchy [13], [14] that comprises application, information and communication service layers.

This paper presents an analysis of the RAMI4.0 service hierarchy, compared to the traditional Service Oriented Architecture (SOA), towards a methodology for the design of RAMI4.0 services. Their main difference is that RAMI4.0 services are logically attached to assets [13]. Thus, in general, assets must be identified first; or if a service is identified first, an asset to which to attach the service should eventually be designed. For example, a service to configure a production line should be attached to an asset such as *productionLineConfiguration*. An approach is presented towards the design of RAMI4.0 services that seeks to identify assets to be managed as I4.0 components and is based on Object-Oriented Analysis and Design (OOA&D) principles. The rationale behind the use

[1] www.iiconsortium.org/
[2] www.plattform-i40.de/I40/Navigation/EN/Home/home.html

of OOA&D is the similarity that holds between objects and assets as I4.0 components. Data and functions within objects are accessible through *method* invocations; data and functions of assets are accessible through *service* invocations (managed by an Administration Shell). Although the similarity of objects and I4.0 components is apparent and SOA and OOA&D are well known, how to combine these technologies for the design of I4.0-Component systems in a systematic manner is not obvious. The paper discusses relevant relationships between those technologies and an approach based on OOA&D to combine them.

Section II presents background to SOA and RAMI4.0 models and service hierarchy. Section III presents design issues for RAMI4.0 services and Section IV the main aspects of our approach. Section V outlines the use of our approach in identifying assets and services in the system being designed within the EU project DISRUPT to support decision making under events that disrupt manufacturing operations. Section VI presents related work and Section VII concludes the paper.

II. Background to SOA and RAMI4.0

A. SOA

SOA is a style to architecture design where services, as building blocks, are designed to comprise business processes. The main benefits of SOA are agility and flexibility to create new business processes quickly and efficiently from existing services, or mostly from them. Technically, a service is a wrapper to functionality available in enterprise software systems or through other services, or combinations thereof. Services are organised into a hierarchy to facilitate separation of concerns, loose coupling and service reuse. Business services at the top layer correspond to business tasks within business processes. Integration/functional services in the layer(s) below serve to compose services in the layer above; infrastructure systems at the bottom layer provide the required (service) functionality.

1) SOA based on Business Modelling: Creating new business processes quickly and efficiently from existing services requires that services be aligned with business goals and objectives. A business model should: (a) represent the business resources and processes required to meet enterprise operational, tactical, and strategic business goals; and, (b) specify enterprise goals, enterprise outcomes (objectives) to meet the goals, the processes to achieve the outcomes, the capabilities to implement the processes, and the services to expose the capabilities [15].

2) Service Design (Identification): How agile and flexible a SOA is can be determined by the level of service reuse. Service design should aim to reuse and extend existing services first, and to create a new service as a last resort [15]. Thus, the initial identification of services should aim to identify services that have wide applicability and whose functionality does not overlap, or is minimum. Efficient service identification requires a searchable catalogue (e.g., a registry) that lists the functions and data provided by existing services in order to check that functionality is not replicated. Catalogues also provide access to service descriptions, software services, policy, documentation, and other assets essential to the operation of a business.

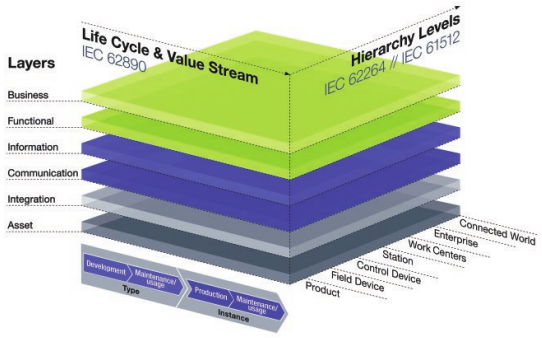

Fig. 1. RAMI4.0 cubic model [12]. Copyright Platform Industrie 4.0, 2015.

3) Semantic Interoperability: Reusability involves interoperability too. For services to be reusable within different business processes at different levels/layers, services must use a common enterprise (or inter-enterprise) semantic data model. Service interface design is based on the semantic model. Access to data in domain systems involves a mapping from the semantic model to *internal* data models of each domain system and vice versa. The creation of the semantic model should consider industry and cross-industry semantic dictionaries, if available, in order to ensure interoperability.

4) Service Types: Business services are offered to external consumers, are coarse grained and support high-level functionality. Integration services facilitate the integration between, and access to, existing applications and services in layers below. They can be classified according to the service classification by ISO/IEC 18384-2 Reference Architecture for SOA (originally developed by the Open Group) [16]. Access services integrate legacy applications and functions into a SOA solution. Information services provide access to the persistent data of a business. Lifecycle services support the management of the lifecycle of SOA solutions.

B. RAMI4.0: Reference Architecture Model Industry 4.0

RAMI4.0 proposes two models to guide the design of virtual/digital representations of assets, an asset being anything of value to an organisation including physical objects, such as a product, a machine, a production line, a factory, but also intangible objects, such as ideas, designs, software, and machine configurations.

1) RAMI4.0 Cubic Model: Shown in Fig. 1, this model "shows technical objects (assets) in the form of layers, and allows them to be described, tracked over their entire lifetime and assigned to technical and/or organizational hierarchies" [8]. An asset is represented at the bottom layer (vertical axis) in the figure; the layers above represent the various types of functions of an asset. The *Layers* view represents a functional decomposition to guide software design.

The life cycle of an asset is represented by the *Life Cycle & Value Stream* dimension (left-hand horizontal axis in that figure), both as a product type and as a product instance.

TABLE I
PROPOSED SERVICES FOR RAMI4.0 SERVICE ARCHITECTURE [13], [22]

Communication services (used by Inf. services)	Information services (used by App. services)	Platform services (admin. services)
+ DiscoverHost() + Transmit() + Connect() + Disconnect() + NegotiateQoS() + Encrypt()	+ Read() + Write() + Publish() + Subscribe() + Create() + Delete() + Browse() + MethodCall()	+ Register() + Discover() + GetId() + GetVersion() + Authenticate() + Connect() + NegotiateQoS() + GetCurrentQoS()

TABLE II
SIGNATURE OF INFORMATION SERVICE CALLS [13]

Signature (o = optional)	Description
Operation: Read, Write, etc.	what is being done to the entity at the given target address
Target address	target of the service operation, e.g: attribute to set
Information payload	attribute values to assign
RSVP flag (o)	indicates whether a response/reply is expected
Context QoS (o)	QoS to provide for this service call
Context security (o)	expectation on the confidentiality of the end-to-end communication
Context (implicit)	context of the service call: application, user, the local I4.0 Network, a global (discovery) scope

As a product type, a design is produced and refined; as a product instance (of a design), a product is manufactured, used, maintained, and disposed of (*cf.* PLM [3]). This dimension is based on IEC 62890, "Life-cycle management for systems and products used in industrial-process measurement, control and automation" [17]. A product in use is assigned to a functional (operational) hierarchy level within a factory (right-hand horizontal axis). The Hierarchy Levels dimension is based on IEC 62264, "Enterprise-control system integration" [18], and IEC 61512, Batch Control [19]. The RAMI4.0 cubic model is based on the cubic model of the Smart Grid Architecture Model Framework (SGAMF) [20].

2) The I4.0 Component Model: An I4.0 component is made up of an asset and an Administration Shell (AdminShell). The AdminShell is a software wrapper structured into a Manifest and a Component Manager. The Manifest holds the properties (data and functions) of an asset; the Component Manager provides access to them. The structure of the AdminShell is based on "IEC 62832 - Digital factory framework" [21], but was extended for RAMI4.0 to include the specification of functions an asset can perform. An AdminShell does not have to be physically close to its asset; it can reside on a software server. I4.0 components can be nested, making up composite components that may be only temporarily active if needed.

3) Service-based Interoperability: A service architecture [13], [14] is being defined to provide access to the properties (data and functions) of assets. The service architecture is organised into layers, *communication, information* and *application* service layers, which fairly correspond to the software decomposition layers in Fig. 1 — the application service layer embraces both the functional layer and the business layer in Fig. 1. Thus, software decomposition layers *implement* asset functionality; service layers *expose* asset functionality.

RAMI4.0 has proposed the *base* services shown in Table I in order to facilitate interoperability. Communication and information services are 'non-technology specific'. They have been defined at an abstract/conceptual level to avoid vendor lock-in and to efficiently cope with technology evolution. Hence, they must be mapped to services in the communication technology used, e.g., OPC-UA Services [23]. Communication services are used by information services, as described below. Information services must be used/refined by platform and application services. Platform services are *functional* services (*cf.* functional layer in Fig. 1) that are 'domain-agnostic' as their sole purpose is administering assets virtual representations.

Application services (*cf.* functional and business layers in Fig. 1) are *not* defined by RAMI4.0. They may be domain-specific and would refer to some higher-level, often asset-specific functionality (e.g., 'close valve', 'calibrate', 'drill hole'). As observed in [22], "no standards for application-specific services or respective service catalogues are known".

4) The Link between Software and Service Layers: The AdminShell "itself is implemented on the information layer" [13, p. 7], "making available data and functionality of the adjoining layers" [idem.]. That is, the Manifest (in an AdminShell) comprises the information layer (in Fig. 1), holding all the (properties) data and functions of an asset. The Component Manager (in an AdminShell) receives *information* service requests (operations), those in the middle column of Table I, to access the data and functions of an asset. These functions are exposed, by design, by higher level application services [13, pp 17,19], and are invoked by a Component Manager through the service operation `MethodCall()` (see Table I).

So far, only the signature of information service calls has been proposed, see Table II. This table shows the parameters to be received by a Component Manager in a service request message. The parameter *Operation* (one of those in the middle column of Table I) identifies the internal method a Component Manager will call with parameters *Target address* and *Information payload*. The optional parameters, if specified, will be processed by a Component Manager through calling the relevant communication service operation (left column of Table I): `Transmit()` (a response) if the *RSVP flag* is specified; `NegotiateQoS()` if *Context QoS* is specified; and `Encript()` if *Context security* is specified [13, p. 22].

III. DESIGN ISSUES FOR RAMI4.0 SERVICES

A. Overview

SOA and the RAMI4.0 service Architecture (RM-SA) share a few design concepts. They both propose a service hierarchy with business services at the top. The AdminShell is similar to the Service Container described in ISO/IEC 18384-2 Reference Architecture for SOA [16].

The main difference between both is that services in RM-SA are attached to assets, which are inherently stateful in

contrast to traditional SoA. Assets are considered first in RAMI4.0, or should so be considered; services are the means for accessing asset properties. This is a key concept defined at the model-design level. It stems from the fundamental purpose of Industry 4.0: "to facilitate cooperation and collaboration between technical objects [assets], which means they have to be virtually represented and connected" [8, p. 7]. By attaching services to assets at model-design level, communication and interoperability capabilities are endowed to an asset the moment its virtual representation is specified.

Communication requires agreement on APIs, while interoperability requires agreement on the meaning of data to be exchanged. Accordingly, RM-SA has also proposed signatures (APIs) for services in the information layer, which is the point of entry to all other services. For interoperability, RAMI4.0 has proposed the use of standards for modelling asset properties (*cf.* semantic information model), accessible through AdminShells (see Fig. 8 in [13, p. 20]: "Examples of existing standards to be integrated into the [AdminShell]"). Thus, RAMI4.0 developers can focus on designing application services atop Communication and Information (CI) services. Further, CI services are not technology-specific, and hence are inherently flexible — different assets can use different communication technologies, or these technologies can change, without affecting design based on CI services.

In traditional SOA, *asset* and *infrastructure* services [16] among others could be considered as asset-attached services (section II-A4). All services in a SOA can be considered as attached to the assets to which they provide access (e.g., a database system). Also, it is possible to develop functional capabilities for "cooperation and collaboration between technical objects" following traditional SOA best practices. Some example applications in this direction are already available, e.g., home appliances controlled through the internet, and automatic software updates in our computers [24].

However, there is a fundamental difference in the design approach between SOA and RAMI4.0 and its service architecture, a difference that can be regarded as analogous to the difference between *procedural programming* and *object oriented programming* (see Booch *et al.* [25] for an in-depth comparison of both programming paradigms). SOA offers a strong overhaul to remote procedure call techniques where services are expected to be developed on a stateless manner (just like procedures implemented in the functional programming paradigm). In contrast, the focus of RAMI4.0 are assets that exist in the physical world, thus the SOA's stateless handling of assets would become cumbersome (e.g., these assets often require certain protocols to follow during the manufacturing process), and in fact these assets resemble objects more.

As design practice tends to align with the principal design concept, SOA and procedural programming have similar design practices in that main 'activities' are identified first and data is defined as needed. In SOA, business resources and processes to achieve enterprise goals are first identified (*cf.* business model), then business services, integration services, etc. The enterprise semantic data model is defined as needed with global scope. (Note that business resources correspond to assets, but design is focused on services.) In procedural programming, the major procedures (modules) are first identified and then utility procedures; data structures are defined as needed, with local or global scope depending on the problem.

B. Traditional Objects vs I4.0 Components

It is our view that RAMI4.0 models suggest a development based on OOA&D. It is indeed the case that most service-oriented software (e.g., web systems especially stateful services based on RESTful [26] and WSRF concepts [27]) has been developed using OOA&D — but perhaps not as cohesively as, in our view, RAMI4.0 suggests, *throughout* those entities whose properties need to be reached remotely: "Even a factory can be an asset that has an administration shell and can be addressed using its ID" [8, p. 36].

In this section, traditional OO objects and RAMI4.0 objects, i.e., I4.0 Components, are compared. In Section IV we discuss the use of OOA&D in the design of I4.0 Component services. The terms *asset*, *object* and *technical object* are used interchangeably from now on.

1) Objects and Remote Objects: Traditional objects consist of methods and data, data is typically encapsulated and is accessed through method invocations. The method invocation mechanism is a given in that we simply refer to the relevant object method, e.g., `myObject.add(1)`. Invoking remote object methods is similar, though it involves more setting-up. E.g., Java remote method invocation (RMI) uses local and remote stub methods in clients and servers that hide the communication aspects of remote method invocations.

2) I4.0 Components: Objects+Internet: In an I4.0 Component, asset functions and data (virtual representation of the real life properties of the asset) are specified in the asset AdminShell, in the Manifest, and are exposed through services. Since communication and information (CI) services in RM-SA have been proposed, and application services atop must be based on CI services, the service invocation mechanism of application services is somewhat a given as well. RAMI4.0 developers can thus focus on designing application services that involve interacting with I4.0 components. E.g., knowing the ID of an I4.0 component, its functions and data can be obtained (queried) through its CI services. The functions exposed by application services in an AdminShell can correspond to, or can be a subset or a superset of, the actual functions supported by the relevant asset; it is only a design decision.

Accessing the *actual* functions of an asset involves a series of service invocations starting at an application service, going down through the specified service hierarchy, until the actual functions are reached. For a software asset, such as a DBMS, access to its actual functions will take place in the same way it does in traditional SOA: the last service (in that series) will be an *access service* that invokes queries or updates on the DBMS using the DBMS *internal* interface.

Recall that *application services* are not defined in/by RAMI4.0. To the best of our knowledge, we know of *no*

proposed definition for any domain. Also, application services comprise functional and business services. This is only a logical division. It suggests that, as in traditional SOA, application *business services* should be at the top and correspond to business tasks, and that other application services be defined as deemed necessary, possibly organised into a sub hierarchy (between business and CI services) in order to promote loose coupling and service reuse.

3) Accessing Physical Asset Properties: In manufacturing, some business processes correspond to manufacturing processes such as producing a particular car model. Hence, business services comprising such processes would correspond to manufacturing tasks, such as welding, assembling, painting, etc., which are performed by manufacturing physical assets such as machines and production lines.

Access to manufacturing assets' functions and data differs from access to software assets' only in the last step. A series of service invocations will go down the service hierarchy up to an *access service* that invokes methods, relevant to the task on hand, on the software that controls a machine, e.g., a Manufacturing Execution System (MES). Finally, the software that controls the machine can send/write commands to the machine's embedded system to actually affect the machine operation or read status data.

Note that manufacturing assets have temporary constraints. Their physical functions may be long-term compared to functions of software assets, and thus may affect non-functional requirements with regard to quality of service. Services that provide information on the time granularity of assets physical functions are needed for planning.

IV. I4.0 Component Design based on OOA&D

The similarity between traditional objects and I4.0 components suggests considering the use of OOA&D for the design of I4.0 components. The benefits of OOA&D include: abstraction, encapsulation and modularity, among others [25].

In this section we outline a generic design of I4.0 components and their services based on OOA&D, discussing relevant issues as they arise. We follow a top-down approach, identifying assets that should be managed as I4.0 components (i.e., through an AdminShell), and suggesting how to group, and what *application services* to attach to, I4.0 components. Recall that application services comprise business and functional services and must be based on RM-SA communication and information (CI) services.

A. The Ultimate I4_Object Class

There is a class in various object-oriented programming languages that is the ultimate superclass of any class. In Java, every class has *Object* as its superclass; all subclasses can use or override the *base methods* of this class, e.g., clone() to create a copy of an object, equals(otherObject) to compare two objects, among others. Similarly, every I4.0 component can have an ultimate superclass that we call the *I4_Object*.

The *base services* (cf. methods) of the *I4_Object* class can include RM-SA CI and platform (admin.) services, see Table I. Every I4.0 component, once specified, can be endowed with those service capabilities. These services must be overridden by each subclass derived from *I4_Object*, so that the actual capabilities of each asset are used.

In addition to those services, generic *functional application services* should be defined in the class I4_Object so that every I4.0 component inherits them. As I4.0 components are Internet objects *per se*, the service *monitor(what, everyWhen)* should probably be such a generic service. It would be implemented using information services to validate whether the property *what* belongs to the I4.0 component being accessed, and to read and send its values *everyWhen*.

B. The I4_Factory Class

An *I4_Factory* object comprises the I4.0 components of all assets within a factory which, by design, need to be digitally reached. It provides lookup services (atop the information services) to search for available assets and services within a factory, so they can be accessed. In essence, this factory object offers a scope-limited service registry (i.e., a searchable catalogue of associated I4.0 components and their services).

The assets can include manufacturing assets, hardware assets, software assets, information assets, etc. The *I4_Factory* object can contain *references* to the I4.0 components (objects of *I4_Object* subclasses) of those assets, thus holding an *aggregation* relationship. Notice that the factory itself is not to be subclassed, but during instantiation of an *I4_Factory* object it is expected that the managed I4.0 components are identified or designed, along their services, in a similar way traditional services are identified for a SOA solution. Business resources and business processes should be identified first based on business goals.

For the design of I4.0 components and their services, it must be considered that: (a) business resources are assets to be represented as I4.0 components of some class (more on this shortly); and (b) services must be specified as attached to an asset. Hence, a catalogue for searching either for (business) resources or services, or both, is more convenient. This catalogue can serve to identify assets to be managed as I4.0 components and their application services (atop CI services).

The catalogue can be initially loaded with all the *existing* resources in a factory (manufacturing assets, software assets, etc.). Then, for each resource/asset, its functions/services will be added to the catalogue. For example, for manufacturing assets such functions may include: welding, assembling, painting, etc. These services were mentioned above as business services; they should be labelled so in the catalogue. If lower-level (functional) services that comprise business services are already implemented, they should be included in the catalogue.

Higher-level, composite AdminShells could represent complex business and/or manufacturing processes (e.g., described with BPMN/BPEL [28]). Cataloguing these services is essential to minimize the needs to extend our *I4_Factory* class (as otherwise the need for subclassing *I4_Factory* would rise with every new process to prepare for its specialities). By controlling the availability and access (i.e., authorization checks) to

these high-level services a factory object essentially validates and coordinates the use of asset services. By utilising a stateful behaviour as that of BPEL it also allows the coordination of the outputs and inputs between services, or between processes consisting or various services. Section V presents an example of an *I4_Factory* object and its responsibilities.

C. Other I4_ Classes

1) Manufacturing I4_ Classes: For manufacturing assets the class *I4_ManufacturingAsset* is defined as the root class. Subclasses of this class could be defined based on IEC 62264 Enterprise-control system integration [18]: *I4_MnfAsset_BatchControl*, *I4_MnfAsset_ContinuousControl* and *I4_MnfAsset_DiscreteControl*.

Sub-sub-classes within each subclass could be defined, and so on, the overall purpose being to facilitate organisation and management. Within the class hierarchy thus defined, only the class at the bottom will serve to instantiate I4.0 components of actual assets (resources) initially identified and entered in the catalogue. For a machine entered in the catalogue, its instantiated I4.0 component object will hold services corresponding to manufacturing tasks such as welding, painting, etc.

The I4.0 component object of a machine can hold maintenance services. A maintenance service may or may not involve human interaction while the service runs. A maintenance service that involves no human interaction (other than launching the service) will be updating the software driver that controls the operation of the machine. This service will involve CI services to fetch (read) the status of the machine, stop it if needed, change (write) its status to *software update mode*, check (read) its status has changed accordingly, send the new software driver to the machine, re-boot the machine, etc.

Production processes can also be regarded as manufacturing assets, and hence managed as I4.0 components derived from an *I4_ManufacturingProcess* class. They can comprise invocations to the services of I4.0 components of actual manufacturing assets. They can include services for configuring the various machines involved in the process they represent, through invoking relevant CI services in the I4.0 components of the machines to be used in the process.

2) Software I4_ Classes: Software assets such as databases or Enterprise Information Systems offer data services and operational services (e.g. MES). As I4.0 components, software assets offer application services to access specific data, update data, or control a machine (*cf.* MES). E.g., application services for a database will offer read and write operations of specific data records through a library, such as JDBC or ODBC. Application services for maintenance purposes can be offered to update software assets as described above for updating machine driver software, based on CI services.

V. Application Example

This section shows how to use our approach to I4.0 component design. The example we use is based on work carried out in the EU funded project DISRUPT[3] [29]. We briefly describe the DISRUPT system and then its configuration as I4.0 components. Even though the DISRUPT system is not designed as an I4-component system, it can offer a basis to highlight the use of the proposed approach.

[3]http://www.disrupt-project.eu/

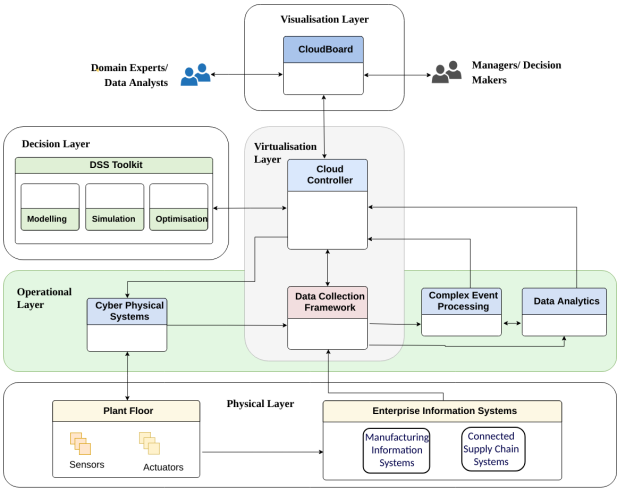

Fig. 2. DISRUPT conceptual architecture atop its target environment.

A design based on I4.0 components can be considered for two reasons. First, such a design involves mostly the design of wrappers (AdminShells), which are also needed for SOA-based design. A design based on the I4.0 Component model has the benefit of using OOA&D as outlined in the last section. Thus, AdminShells and software assets can both be designed using OOA&D. Second, the I4.0 Component model (i.e., RAMI4.0+RM-SA) is a specific yet flexible approach to interoperability. Since the structure to store asset properties (i.e., the Manifest within the AdminShell of each asset) and communication and information services to reach the properties have been proposed, the development effort can mostly focus on application services. Also, solutions that partly involve I4.0 component design and implementation may prove useful through the transition into Industry 4.0 paradigms.

A. The DISRUPT System

The purpose of the DISRUPT system is to support, in close to real-time, data-driven decision making and enactment of decisions on manufacturing operations under events that disrupt enterprise operations, e.g., delays in the supply chain or failure in a production line [29], [30]. Fig. 2 shows the DISRUPT software architecture atop its target environment. The target environment, labelled *physical layer* in the figure, comprises existing manufacturing equipment and Enterprise Information Systems (EISs).

The DISRUPT system is to be added on to such an environment to: (a) collect data from EISs and directly from the plant floor and the supply chain in real time; (b) process this data to identify events that disrupt enterprise operations; and (c) generate alternatives to circumvent a disruption through modelling, optimisation and simulation. A visualisation dashboard (CloudBoard in figure) will display the state of enterprise

operations (continuously) and alternatives to disruptions on identifying disruptions. Such alternatives can either be interpreted by a human, who will then manually enact relevant actions, or carried out by a machine driven by the Cyber-Physical System (CPS) module.

The Cloud Controller enacts user requests (received through the CloudBoard) and facilitates the integration and interaction between all other modules. User requests include setting-up monitoring of equipment and enacting alternative actions to disruptions through the CPS module, among others.

B. The DISRUPT System as I4.0 Components

The DISRUPT system will be deployed as a web system using the Model-View-Controller (MVC) pattern. The CloudBoard will implement the *View* which corresponds to data visualisation required by users. The Cloud Controller will implement the *Controller* which corresponds to processing users input requests and interacting with the *View* and the *Model* as required. All other DISRUPT modules will implement the *Model* which corresponds to the business logic and data.

1) The I4_Factory Component of DISRUPT: Recall that an *I4_Factory* component (object), specified as an instance of the *I4_Factory* class, corresponds to the virtual representation of a factory. Such a component comprises the I4.0 components (assets and services) within a factory which, by design, must be exposed and digitally reached. It is, thus, the point of access to available assets and services.

The *I4_Factory* component of DISRUPT (and of any enterprise for that matter) corresponds to both the *Controller* and the *Model* (*cf.* MVC) in its web system. The *Controller* corresponds to the AdminShell, and thus to the DISRUPT Cloud Controller; the *Model* corresponds to the factory as a set of assets and services, i.e., to all other DISRUPT modules virtual representations (I4.0 components). The *I4_Factory* component of DISRUPT will expose high-level application services composed of the specific application services provided by other DISRUPT modules (except the CloudBoard).

2) I4_Software Components: DISRUPT modules (except the CloudBoard and the Cloud Controller) can be I4.0 components instantiated from the *I4_Software* class. The application services of each module, to be specified within its AdminShell, will be derived from the overall functionality of each module. E.g., the CPS module is to continuously collect and process data from the plant floor and the supply chain. Thus CPS application services will include some form of data *monitor* and *filter* services. The Data Collection Framework (DCF) will collect, process and store data from EISs, and store processed data from CPS. Thus DCF application services will include *query* and *store* services. The Data Analytics (DA) module will seek to identify (from data in DCF) abnormal trends that may lead to actual disruptions. DA application services will include analysis services such as *regression* analysis.

3) Manufacturing Assets: Within the project DISRUPT, lower-level assets such as manufacturing equipment are not managed as I4.0 components, i.e., AdminShells were not defined for them. It can be done, but the design choice was for the CPS module to manage those assets collectively or individually according to decisions taken above. Partly this decision was taken because DISRUPT functional requirements involve only remote control of manufacturing assets, where assets role can be seen as *passive*. Autonomous cooperation between assets would require an *active* role from assets based on complex application services atop CI services; in this case an AdminShell for each active asset would be necessary.

4) Implementation and Deployment Issues: Compared to a traditional SOA web system implementation, the implementation of DISRUPT as I4.0 components as outlined above must include the following. Each I4.0 component (AdminShell) must expose the CI services proposed for RM-SA (Table I) so that assets data and application services can be reached; and application services should be based on CI services. E.g., the Cloud Controller (MVC Controller, *I4_Factory* AdminShell) can offer a lookup service to be tuned according to some criteria, say, data services or analysis services. This lookup service can use CI-information services to recover the relevant services. If a data service is chosen, e.g. monitored data of a particular kind of equipment, the Cloud Controller can invoke the relevant application service of the Data Collection Framework to fulfil the service request for monitoring data.

The aspect of services being attached to assets should not affect implementation much. The code of a service can be a method within a Java program whose *main()* procedure will act as the AdminShell, or it can be a PHP script in the same directory where the PHP AdminShell resides. Overall we think the concept of asset-attached services helps not only design but also deployment. For instance, the web server directory can be organised according to types and subtypes of assets to facilitate access to relevant code and data.

VI. Related Work

Our approach outlined above aims to guide the identification of assets to manage as I4.0 components and the design of application services to access through AdminShells. Tantik and Anderl [31], [32] propose a structure for AdminShells with "separated data modules and functionality applications for a flexible adaptation", based on the entire functionality AdminShells should support as described in [33]. The structure is generic; in principle it should be used for any (asset) I4.0 component. Such a structure could help select/specify the application services an AdminShell should expose on account of the actual functionality of the corresponding asset.

The identification and design of assets and services should be based on the analysis of requirements and use cases to identify the capabilities that are needed to fulfil business goals. Usländer describes a systematic agile service engineering for Industry 4.0 software applications [34] that can be considered in defining our methodology.

The need for semantic interoperability (semantic data information model) was discussed in the context of traditional SOA. In Industry 4.0, semantic interoperability is multi-faceted due to the different types of assets and standards and vocabularies to describe them. González *et al.* [35], [36] propose to employ

the Resource Description Framework (RDF) as "the lingua franca to represent and integrate information in Industry 4.0 contexts", as a "middle layer" within AdminShells to support interoperability. The approach can manage specifications in English and German. It is not described how the middle layer can be integrated within AdminShells. We believe different types of RDF translations could be configured as application services.

VII. Conclusion

This paper has analysed the RAMI4.0 models and service architecture, comparing them to traditional SOA and OOA&D. I4.0 components resemble objects; their communication capabilities can be considered as a given. We have suggested to model I4.0 components through OOA&D and outlined some *I4_* classes to do so. We also presented an application example of I4.0 components and highlighted their relationship to web system technologies.

As for future work, our approach to identify assets and application services must be refined through the design of classes and application services for I4.0 components based on application scenarios. What asset hierarchies, service hierarchies, or both, are more suitable under what type of application scenarios? For example, a single I4.0 component CPS module for the DISRUPT system is enough, as opposed to multiple I4.0 components for each production machine, because requirements involve a passive role for such equipment.

Acknowledgement

This work has been supported by the EU H2020-FOF-11-2016 project DISRUPT (grant no 723541).

References

[1] R. Anderl, "Industrie 4.0 – advanced engineering of smart products and smart production," in *Proceedings 19th International Seminar on High Technology, Technological Innovations in the Product Development*, (Piracicaba, Brazil), 2014.

[2] E. A. Lee and S. A. Seshia, *Introduction to Embedded Systems - A Cyber-Physical Systems Approach*. LeeSeshia.org, 2011.

[3] L. D. Xu, E. L. Xu, and L. Li, "Industry 4.0: state of the art and future trends," *International Journal of Production Research*, vol. 56, no. 8, pp. 2941–2962, 2018.

[4] L. Da Xu, W. He, and S. Li, "Internet of things in industries: A survey," *IEEE Transactions on industrial informatics*, vol. 10, no. 4, pp. 2233–2243, 2014.

[5] S. Li, L. Da Xu, and S. Zhao, "5G Internet of Things: A survey," *Journal of Industrial Information Integration*, vol. 10, pp. 1 – 9, 2018.

[6] J. Cheng, W. Chen, F. Tao, and C.-L. Lin, "Industrial IoT in 5G environment towards smart manufacturing," *Journal of Industrial Information Integration*, vol. 10, pp. 10–19, 2018.

[7] IIC, "The Industrial Internet of Things Reference Architecture," technical report, Industrial Internet Consortium, Jan. 2017.

[8] "IEC PAS 63088:2017 Smart manufacturing - Reference architecture model industry 4.0 (RAMI4.0), Publicly Available Specification (PAS) Pre-Standard," Mar. 2017.

[9] IoT-A, "Internet of Things - Architecture," final report, EU Project 257521, 2013.

[10] IoT-RA, "Internet of Things Reference Architecture," standard in progress, ISO/IEC CD 30141:2016(E), 2016.

[11] WoTA, "Web of Things (WoT) Architecture," Visite Jul. 2017.

[12] VDI/VDE and ZVEI, "Reference Architecture Model Industrie 4.0 (RAMI4.0)," status report, VDI/VDE Society Measurement and Automatic Control (GMA), ZVEI German Electrical and Electronic Manufacturers Association Automation Division, July 2015.

[13] VDI/VDE, "Industrie 4.0 Service Architecture — Basic Concepts for Interoperability," status report, VDI/VDE Society Measurement and Automatic Control (GMA), Nov. 2016.

[14] "DIN SPEC 16593:2017-04. RM-SA - Reference Model for Industrie 4.0 Service architectures - Basic concepts of an interaction-based architecture," Apr. 2017.

[15] M. Rosen, B. Lublinsky, K. T. Smith, and M. J. Balcer, *Applied SOA: Service-Oriented Architecture and Design Strategies*. Wiley Publishing, Inc., 2008.

[16] ISO/IEC 18384-2:2016 , "Reference Architecture for Service Oriented Architecture (SOA RA)."

[17] "IEC 62890: Life-cycle management for systems and products used in industrial-process measurement, control and automation (Working Document)," 2013.

[18] "IEC 62264: Enterprise-control system integration (Intl. Stand.)," 2013.

[19] "IEC 61512-1: Batch control (Intl. Stand.)," 1997.

[20] "SGRA: Smart Grid Reference Architecture. CEN-CENELEC-ETSI Smart Grid Coordination Group," 2012.

[21] "IEC 62832: Industrial-process measurement, control and automation – Digital factory framework (Technical specification)," Dec. 2016.

[22] Platform Industrie 4.0 (PI4) and ZVEI, "Industrie 4.0 Plug-and-Produce for Adaptable Factories: Example Use Case Definition, Models, and Implementation," Working paper, PI4 and German Electrical and Electronic Manufacturers Association Automation Division, June 2017.

[23] "IEC 62541-4:2015 OPC unified architecture - Part 4: Services.," 2015.

[24] BOSCH, "The Bosch IoT Suite." https://www.bosch-si.com/media/en/bosch_si/iot_platform/bosch-iot-suite_product-brochure.pdf, 2016.

[25] G. Booch, R. Maksimchuk, M. Engle, B. Young, J. Conallen, and K. Houston, *Object-oriented Analysis and Design with Applications, Third Edition*. Addison-Wesley Professional, third ed., 2007.

[26] L. Richardson and S. Ruby, *RESTful web services*. " O'Reilly Media, Inc.", 2008.

[27] OASIS WSRF Technical Committee, "Web Services Resource Framework (WSRF) standard version 1.2," 2006.

[28] M. B. Juric and K. Pant, *Business Process Driven SOA using BPMN and BPEL*. Packt Publishing, 2008.

[29] P. Eirinakis, J. Buenabad-Chávez, R. Fornasiero, H. Gokmen, J.-E. Mascolo, I. Mourtos, S. Spieckermann, V. Tountopoulos, F. Werner, and R. Woitsch, "A proposal of decentralised architecture for optimised operations in manufacturing ecosystem collaboration," in *Working Conference on Virtual Enterprises*, pp. 128–137, Springer, 2017.

[30] E. Kavakli, J. Buenabad-Chávez, V. Tountopoulos, P. Loucopoulos, and R. Sakellariou, "An architecture for disruption management in smart manufacturing," in *SMARTCOMP 2018: 4th IEEE International Conference on Smart Computing*, pp. 279–281, 2018.

[31] E. Tantik and R. Anderl, "Potentials of the Asset Administration Shell of Industrie 4.0 for Service-Oriented Business Models," *Procedia CIRP*, vol. 64, pp. 363 – 368, 2017. 9th CIRP IPSS Conference: Circular Perspectives on PSS.

[32] E. Tantik and R. Anderl, "Integrated Data Model and Structure for the Asset Administration Shell in Industrie 4.0," *Procedia CIRP*, vol. 60, pp. 86 – 91, 2017. Complex Systems Engineering and Development, Proceedings of the 27th CIRP Design Conference, Cranfield University, UK, 10th - 12th May 2017.

[33] Platform Industrie 4.0, "Structure of the Administration Shell: Continuation of the Development of the Reference Model for the Industrie 4.0 Component," working paper, Platform Industrie 4.0, Apr. 2016.

[34] T. Usländer, "Agile service-oriented analysis and design of industrial internet applications," *Procedia CIRP*, vol. 57, pp. 219 – 223, 2016. Factories of the Future in the digital environment – Proceedings of the 49th CIRP Conference on Manufacturing Systems.

[35] I. Grangel-González, L. Halilaj, G. Coskun, S. Auer, D. Collarana, and M. Hoffmeister, "Towards a semantic administrative shell for industry 4.0 components," in *2016 IEEE Tenth International Conference on Semantic Computing (ICSC)*, pp. 230–237, Feb 2016.

[36] I. Grangel-González, L. Halilaj, S. Auer, S. Lohmann, C. Lange, and D. Collarana, "An rdf-based approach for implementing industry 4.0 components with administration shells," in *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*, pp. 1–8, Sept 2016.