Journal of Software Engineering and Applications, 2017, 10, 734-753
http://www.scirp.org/journal/jsea
ISSN Online: 1945-3124
ISSN Print: 1945-3116

From Mechatronic Components to Industrial Automation Things: An IoT Model for Cyber-Physical Manufacturing Systems

Theodoros Foradis, Kleanthis Thramboulidis

Electrical and Computer Engineering, University of Patras, Patras, Greece
Email: thrambo@ece.upatras.gr

How to cite this paper: Foradis, T. and Thramboulidis, K. (2017) From Mechatronic Components to Industrial Automation Things: An IoT Model for Cyber-Physical Manufacturing Systems. *Journal of Software Engineering and Applications*, **10**, 734-753.
https://doi.org/10.4236/jsea.2017.108040

Received: May 4, 2017
Accepted: July 9, 2017
Published: July 12, 2017

Abstract

IoT is considered as one of the key enabling technologies for the fourth industrial revolution that is known as Industry 4.0. In this paper, we consider the mechatronic component as the lowest level in the system composition hierarchy that tightly integrates mechanics with the electronics and software required to convert the mechanics to intelligent (smart) object offering well defined services to its environment. For this mechatronic component to be integrated in the IoT-based industrial automation environment, a software layer is required on top of it to convert its conventional interface to an IoT compliant one. This layer, which we call IoT wrapper, transforms the conventional mechatronic component to an Industrial Automation Thing (IAT). The IAT is the key element of an IoT model specifically developed in the context of this work for the manufacturing domain. The model is compared to existing IoT models and its main differences are discussed. A model-to-model transformer is presented to automatically transform the legacy mechatronic component to an IAT ready to be integrated in the IoT-based industrial automation environment. The UML4IoT profile is used in the form of a Domain Specific Modelling Language to automate this transformation. A prototype implementation of an Industrial Automation Thing using C and the Contiki operating system demonstrates the effectiveness of the proposed approach.

Keywords

Mechatronics, Cyber-Physical Systems, Internet of Things (IoT), Contiki, UML4IoT Profile

1. Introduction

Based on one of the most commonly used definitions, the term Mechatronics emphasizes on the synergistic integration of the three discipline areas, *i.e.*, mechanical engineering, electronics and intelligent computer control, in the design and manufacture of products and processes [1], *i.e.*, it emphasizes on synergy. What is not clear by this definition is the level at which this integration should be performed, *i.e.*, at the system level, which is the traditional approach, or at the subsystem or even at the mechanical unit (component) level. The latter is proposed in Model Integrated Mechatronics [2] and refined with the 3+1SysML-view model [3] [4]. This approach defines the Mechatronic component as the main building block that abstracts the mechanical object to the software level, and transforms it to a smart object by adding additional functionality to the one offered by the mechanical part. The so constructed mechatronic components are integrated with cyber components and humans to construct the industrial automation system. This approach slightly finds its road to production in the context of Industry 4.0, e.g., [5], since it greatly reduces the coupling between the system components compared to the traditional one, which considers the integration of the three disciplines at the system integration level.

A substantial number of communication mechanisms and middleware are used for the integration of the constituent components of industrial automation systems (IAS). DPWS [6], one of the most recent, is the result of extensive research based on the service-oriented architecture (SOA). It is based on SOAP which was the dominating technology used in SOA implementations. However, the REST paradigm is gaining more attention in manufacturing the past few years [7] [8]. A comparison of REST with SOAP in the context of the manufacturing domain can be found in [7]. Cloud computing has also attracted the interest of researchers. Cloud manufacturing defines a new service-oriented manufacturing model where cloud computing and IoT deeply influence the development process of manufacturing systems [9]. IoT technologies greatly reduce the time for decision-making that is very critical in modern manufacturing environments [10]. Both, cloud computing and IoT, have been moving from buzzwords and hypes to tangible practical technologies that subtly influence and change our world [11]. A state-of-the-art survey in cloud manufacturing is given in [9], which considers IoT as a technology that deeply influences the development of cloud manufacturing.

IoT is aligned well with the architecture of a manufacturing enterprise and it is able to provide "vital solutions to planning, scheduling, and controlling of manufacturing systems at all levels" [12]. Several approaches consider IoT as a technology that can be utilized as an integration mechanism to be used down to the sensor and actuator level of the industrial automation system. Others consider IoT as the new logical transition from the automation and connectivity concepts that exist in the IAS domain for many years. Bradley *et al.* [13], in an article with title "The Internet of Things—The future or the end of mechatronics", argue that many of the smart components associated with the IoT will be essentially mechatronic in nature, and will be constructed as far as it regards their interaction with the physical world on the conventional hierarchical model.

This model considers the controller of the mechatronic component in the loop with the controlled physical unit through sensors and actuators.

For the conventional mechatronic component to be integrated in the IoT-based industrial environment a software layer is required on top of it to convert its conventional interface to an IoT-compliant one. Thus, the adoption of the IoT as integration technology for the system, transforms the conventional mechatronic component to an Industrial Automation Thing (IAT). This transformation is more likely, as authors also argue in [13], to bring significant changes to the way mechatronic, and related, systems are designed and configured. There is already an increasing complexity in the job of the industrial engineer in the task of transferring the functionality of the physical world in the software world in the level of the IAT. To this, the complexity of adding an extra layer to transform the conventional mechatronic component to an IAT is added. New protocols, languages, environments and architectural paradigms should be used and successfully integrated with the already used conventional architectures and this complicates the job of the industrial engineer. To this direction, authors in [14] present UML4IoT with focus on the modeling of the IAT. UML4IoT is a UML-based approach that realizes the model driven engineering paradigm to exploit IoT in the manufacturing domain.

In this paper, 1) we extend the model of the IoT introduced in [14], and 2) define a model-to-model transformer to automate the construction of IATs based on the Contiki OS [15] and the C language. In the extended IoT model, the IAT is still the key artifact for the adoption of the IoT infrastructure in the manufacturing domain. Based on this model the manufacturing system is considered as a composition of cyber-physical and cyber components along with humans [16]. All these components are considered as Things, either permanent or on demand, that collaborate exploiting an IoT communication infrastructure to realize a higher level of behavior, *i.e.*, the one of the system level.

The presented approach is discussed in comparison with other approaches and mainly the IoT-A reference architecture that has been adopted by the Papyrus for IoT project [17], which is building a platform for the design of IoT systems in general. This project has many similarities with our project; both projects use UML and SysML as modeling languages, and Papyrus as tool to provide a modeling solution for IoT. Our approach focuses on manufacturing systems. More specifically, it focuses on the case that a high-level design specification for the mechatronic component is not available. Specific annotations were defined to annotate the C source code specification of the mechatronic component so as to automatically transform the mechatronic component to an IAT. The approach is presented using as case study the Liqueur production laboratory system. The LWM2M IoT application protocol [18] running on top of CoAP [19] is used as the IoT protocol stack.

The remainder of this paper is structured as follows. Section 2 positions this research against related work. In Section 3, the proposed extension to the IoT model which is used in the UML4IoT approach is presented and discussed in comparison with existing IoT models. In Section 4, the Contiki based Industrial Automation Thing is presented along with the case study. The model-to-model transformer for the C language and the Contiki operating system is presented in section 5 and the paper is concluded in the last section.

2. Related Work

IoT offers new levels of connectivity in the industrial domain that may lead to higher efficiency, flexibility, and interoperability among industries [20]. However, not only many definitions exist for the IoT but also several models. These models, e.g., ETSI, IETF, SENSEI, have been developed to capture the key concepts of the domain and provide the infrastructure to develop frameworks and architectures for the systems based on IoT. Fei *et al.* [21] claim that even though IoT has been used in various application domains there is still no clear and uniform definition and architecture about it. Therefore, the IoT domain suffers from an inconsistent usage and understanding of the meaning of several key terms, as also claimed in [22]. The definition of an IoT domain model is a prerequisite for defining IoT Architectures. A detailed discussion on the IoT models can be found in [23], where the IoT-A reference model for IoT is presented and validated. Authors in [24] present and discuss applications of IoT in Manufacturing and describe a five-layer architecture for manufacturing based on IoT.

Authors in [25] focus on the device nature of Thing and consider sensors, actuators and controllers as IoT devices, *i.e.*, things. They focus on the data field structures and evaluate the benefit of using an IP smart gateway as the decentralized peripheral to integrated sensors, actuators and the controller and claim that this may improve the performance of IoT devices. We do not agree with the use of sensors and actuators as first-class model elements in the high-level design specification of the system. Sensors and actuators are just technology artefacts used to integrate the physical with the cyber world so they have no place in the high-level design spec of the system. In our approach, the IAT, which encapsulates sensors, actuators and the controller, plays the role of the Thing and represents the key construct in the IoT manufacturing environment.

MDD becomes more and more popular in the development of embedded software systems and various reports refer efficiency gains, from up to 50%, for example, in the development in the car industry [26], with high error reductions and a rapid increase of the maturity level of developed products. MDD is considered as a promising solution to address the complexity of software development in IoT [27] and improve quality characteristics of the produced software. Malavolta and Muccini [28] argue that MDD is the right tools to address the complexity of wireless Sensor Networks development exploiting abstraction, reuse, separation of concerns and automation. They present a framework to systematically study, classify and compare existing MDA approaches in this area [28]. Several works publish results that exploit the MDD paradigm in IoT based systems to improve their quality characteristics but also the ones of their development process.

Thang *et al.* [27] present FRASAD, a framework based on MDA to manage the complexity of IoT applications. They present a rule based model and a domain specific language to describe the application, using the sensor node as key concept. The primary objective is to model the sensor node software. Contiki is also supported among other OSs by this framework. Authors assume that the application logic of the sensor node program is captured in a Platform Independent Model (PIM), which is constructed using a set of rules they have defined to describe behavior of the sensor node programs. This PIM is next mapped through a Domain Specific Language to the specific platform where it is indented to be executed. However, the approach focuses more on the message dissemination compared to the processing which is considered as an optional part of a sensor node. Our approach focuses on the interface of the Industrial Automation Thing; the behaviour which is very complicated compared to one of a sensor node is defined using another DSML we have defined for structuring the cyber-physical component.

Authors in [29] present the software architecture of a platform developed to address issues, among which the lack of development toolkits, that limit the diffusion of IoT within industrial environments. They also describe an innovative, IoT oriented, model-driven development toolkit that focuses on the seamless integration of heterogeneous industrial devices and sensors, into existing legacy systems by transforming them into web services. The proposed toolkit allows inexperienced developers to discover and compose distributed devices and services into mashups using a modelling tool. Thus, the use of the MDD approach is mainly on the generation of the mashups and does not focus on the modelling of a mechatronic component as is the case of our approach. Furthermore, authors do not refer or describe the domain modelling language that they use in their MDD approach.

A very early approach to model complex IoT systems with UML and then generate RESTful interfaces from these models is presented by Prehofer [30]. Authors do not define any DSML but they construct class diagrams and state charts using only primitive UML model elements. Authors in [31] describe an approach to define a visual DSML for the IoT based on UML. They model the Thing, which they consider as key construct for building an IoT system, using the UML component construct and its interface using provided and required interfaces. Authors do not address the mapping of the conventional object-oriented (OO) interfaces of the Thing with the ones of the REST paradigm.

Yingfeng Zhang *et al.* [32] present a real-time information capturing and integration architecture of the internet of manufacturing things (IoMT) to provide a new paradigm by extending the techniques of IoT to the manufacturing field. They use the term manufacturing thing but they do not focus on its structure and its development process. Instead, they focus on the overall architecture for a manufacturing system and describe a framework with focus on real-time tracking and tracing for the dynamic monitoring of the manufacturing process.

To the best of our knowledge there is no other work that focuses on the automation of the transformation process of the conventional mechatronic component to an IoT compliant one, *i.e.*, to an Industrial Automation Thing ready to be integrated into the IoT-based industrial environment.

3. Towards an IoT Model for Manufacturing

3.1. The IoT-A Reference Model

De *et al.* [33] describe the key concepts of the IoT-A reference architecture that is a result of an EU funded IoT project. These concepts and their interrelations are depicted in Figure 1. Based on this, device is attached to entity, which is associated with resource that is accessed through service. In more detail, authors consider the entity as the "thing" in the IoT, *i.e.*, the focus of interactions by humans and/or software agents. The device represents the hardware component that is either attached to an entity or it exists in its environment and monitors it. The resource is the actual software component that provides information on the entity or enables the controlling of the device. A service exposes the functionality of a device by accessing its hosted resources.

3.2. The Proposed IoT Model

Figure 2 captures the high-level key concepts of our IoT model. Based on this, the IoT is defined as a composition of Things and a processing and communication (IPV6-based) infrastructure. Any artifact that is able to communicate with other Things using the processing and IPV6-based communication infrastructure (*Proc&ComnInfr*) is considered as Thing. Things collaborate to achieve

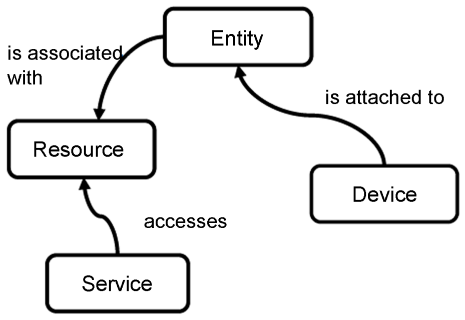

Figure 1. Example key concepts and interaction in the IoT-A model (De *et al.* 2011).

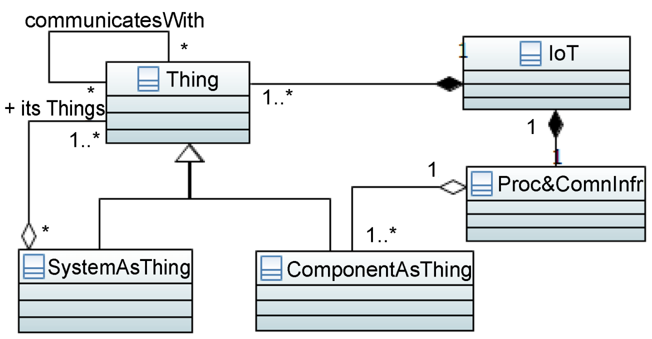

Figure 2. High level key concepts in IoT.

higher level of behavior compared to the one offered by each one of the collaborating Things. Collaborating Things form a new *Thing* of type *SystemAsThing* that represents a system of Things.

A Thing may be either a system of Things (SystemAsThing) or a component (ComponentAsThing). A ComponentAsThing is a Thing that does not utilize IoT for the integration of its constituent components. This type of Thing is used to represent in the IoT world conventional systems or components that have been transformed to Things (ComponentAsThing) by adding on top of their conventional interface an IoT-compliant one. The smallest Thing of this type is a sensor or actuator. The Proc&ComnInfr is a composition of: 1) processing nodes and 2) communication devices, *i.e.*, gateways, bridges, switches, routers, etc. Proc&ComnInfr is modelled as a composition of Things (ComponentAsThing) assuming that these devices are IoT enabled; this will be the case in the near future. The Cloud is considered part of the Proc&ComnInfr.

3.3. The Model of Thing in Manufacturing

Figure 3 presents the proposed model for the Thing. Based on this, a Thing is either real, cyber or virtual. A *RealThing* is either permanent or on demand Thing. A permanent Thing is a composition of a cyber-physical object (*CpObject*) and a cyber IoT enabler (*CyberIotEnabler*). An *OnDemandThing* is an aggregation of a *PhysicalObject* and a cyber-physical IoT enabler (*CpIotEnabler*). As cyber-physical IoT enabler we model any device such as laptop, tablet, mobile phone, wearable, RFID reader, that provides an IoT like interface and is able to interact with a physical object. As physical object we consider a human, an inanimate object or even animal with an embedded or attached tag. A human interacts with an app, *i.e.*, application specific IoT wrapper and is temporarily transformed to a Thing. Physical objects of type animals or inanimate objects

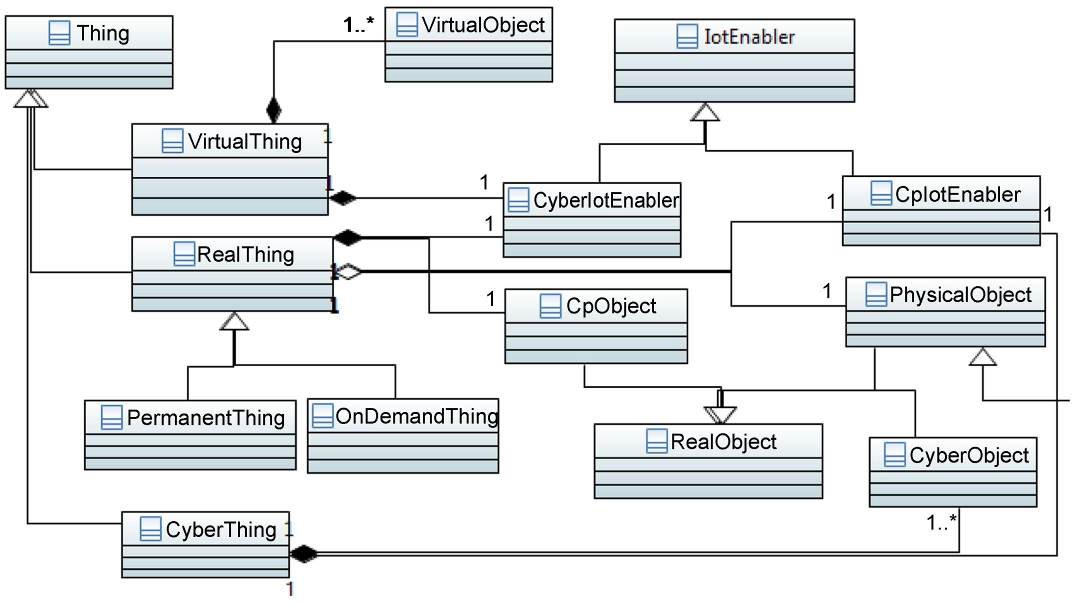

Figure 3. The model of thing in IoT.

with an embedded or attached tag interact with an RFID reader with IoT like interface for the same reason. A cyber-physical object is any physical object that: 1) implements a kind of functionality, *i.e.*, material and/or energy transformations and 2) has been transformed to a smart object by appending on it information processing functionality. A mechatronic component is an example of cyber-physical object. A cyber IoT enabler integrated with virtual objects and deployed on a processing unit constructs a virtual Thing (*virtualThing*). The IoTwrapper of the UML4IoT approach is an example of cyber IoT enabler. A *cyberThing* is defined as a composition of one *CpIotEnabler* and one-to-many *cyberObjects*. This allows the developer to optionally group cyber objects of the system design model and map these to one *cyberThing*.

A Thing may expose to its environment: 1) part of its structure in terms of properties and/or 2) part of its processing or storage functionalities. These functionalities are exposed as services. Services are discriminated to: a) IoT infrastructure services (IotService), b) application domain services and c) application specific services. All types of services may be managed (activated, configured, updated, etc.) through the IoT communication infrastructure. Communication infrastructure services may be considered similar to the Industrial Automation Thing services with the remark that Industrial Automation Things perform material, energy and information processing while communication infrastructure Things perform only energy and information processing. For example, device management services, defined by the OMA LWM2M [18] are *IotServices*. Native services are services that would be defined for a specific application domain, e.g., home automation, manufacturing, or system specific services, such as the generate LiqueurTypeA service of the Liqueur Plant laboratory production system [14]. Device management is implemented by the management interface of the LWM2M. On the other side, Thing management is domain or application specific and should be implemented by specific cyber components on top of the device management and service interface, e.g., the one of LWM2M.

The IoT processing and communication infrastructure (*Proc&ComnInfr*) should provide an environment for a service-based collaboration of Things. Each *Thing* implements functionalities offered to the environment as services with negotiated QoS that should be discovered and exploited by other Things. It may also utilize services of other Things to realize its behavior. The cyber components of the manufacturing system model are deployed during the deployment time on Things of type *ComponentAsThing* (see **Figure 2**). UML/SysML design models of the system are marked with the IoT model elements using the UML4IoT profile to automatically transform the system to an IoT-compliant one.

3.4. Discussion on the Proposed IoT Model

Our definition for the Thing is different from a widely accepted one, described by De *et al.* [33], which defines the thing and its relations to devices, resources and services. Device, Resource, Service and Thing are also model elements in the

IoT model presented by Stephan [34]. In our approach, the *ComponentAsThing* encapsulates and hides how sensing of its physical part as well as actuation are realized, since this is an implementation issue. Based on this, we do not consider sensors and actuators first class model elements in our model and we do not capture these artifacts in the design model of the system. Thus, actors of the OO approach or terminators of the SA approach are modeled either as Industrial Automation Things or as human Things.

This definition of Thing satisfies the requirement set by Zhuming *et al.* [12] according to which all interactions among the system constituent components, which may be humans, machines and products should be performed under the same umbrella. This allows the developers to focus on the system's functionality and not worry about interactions, thus increasing the productivity.

In the design model of the system we do not capture resources. A resource is a technology artifact used: 1) to represent the exposed properties and services of a Thing, and 2) to access these through a well-defined set of operations to achieve low coupling between collaborating Things. The LWM2M defines a set of such operations, *i.e.*, READ, EXECUTE, WRITE, etc., implemented on top of the http operations.

The RESTful as well as the SOAP paradigms can be utilized for accessing the services offered by Things. Thus, we adopt a different meaning for service from the widely accepted and described, e.g., [18], where access to resources from the outside world finally happens through *services*. IoT-A [23] which is a result of the IoT-A EU project, consider the service as an entity that accesses a Resource which is associated with an Entity that has attached a Device. It should also be noted that while a resource is defined in IoT-A as the core software component that represents an entity in the digital world, a Device is attached to a Resource. We do not adopt this model because it is technology driven. Our model focuses on the system modeling level and its objective is to offer a platform independent modeling of the target system. In our model, a Thing has structural (attributes) and behavioral properties (functions/methods). Those properties that are accessible from its environment are represented as resources. The RESTful paradigm is adopted for accessing the resources. In this context and to exploit the benefits of IoT, the networking entities of the IoT *Proc&ComnInfr* are also considered as Things (IoT-Thing *NetworkingThing*) that provide their own set of information processing services required to establish the communication infrastructure of the IoT.

Plant processes as well as other functionalities of the plant are assigned to cyber objects of the system's design model. These cyber objects may be marked as *cyberThings*. Alternatively, cyber objects may be deployed on other *cyberThings* or on Things of the *Proc&ComunInfr*. In both cases the corresponding services are mapped to resources of the corresponding Thing.

Our approach differs from the one of the ebb its platform [20] that identifies the following four layers which are considered required to bind the physical world with software services:

1) Physical-world layer, where devices, sensors and physical-objects are captured,

2) The IoT layer,

3) The internet-of-services layer and,

4) The business system mediation and product life cycle layer.

Pramudianto *et al.* [35] capture sensors and actuators as first-class model elements in their IoT meta model and use the term virtual object to refer to the software entity that acts as a proxy of the real-world object. In our model, the meaning of the virtual Thing is completely different. We use the term software representative (SR) to refer to what authors in [35] call virtual object.

4. A Contiki Based Industrial Automation Thing

4.1. The Liqueur Production Laboratory System

The my Liqueur production mechatronic system, used as case study in this work, is composed of the following mechatronic components: smartSilo1, smartSilo2, smartSilo3, smartSilo4 and smartPipe (see **Figure 4**). The system is based on the case study initially used by Basile *et al.* [36] and then extended by Thramboulidis [16] to be compliant with the mechatronic component concept. The smartSilo mechatronic components are reserved in couples to produce specific types of liqueurs. SmartSilos 1 and 4 form one couple; smartSilos 2 and 3 form the other couple. A mechatronic component has a well-defined interface through which exposes its behavior to be used by the liqueur production processes. This interface exposes the functionalities offered by the silo such as fill, empty, mix and heat. Using the common pipe at the same time for liquid transfer among the silos is not allowed. Moreover, mixing the liquid in two silos at the same time is not permitted due to a constraint in power consumption. Implementation issues regarding the physical silo are encapsulated and hidden from the mechatronic component's environment.

Our intention is to integrate the components of this conventional mechatronic system using IoT and gain from the low coupling that this technology introduces among the interacting components. The use of the IoT will also enable the system to exploit the benefits of this technology regarding the user interaction by allowing end users to produce custom types of liqueur. The end user would be able to define, through an app (*myLiqueurApp*), the production parameters of the desired type of liqueur, as shown in **Figure 4**.

4.2. The Cyber-Physical Component

The legacy smartSilo Mechatronic/cyber-physical component is composed of the physical silo (physical part), a processing, storage and communication unit and the low-level control software (cyber part) required for the smartSilo to provide a higher level of abstraction functionality compared to the one provided by the physical silo. As shown in **Figure 5**, which presents the high-level architecture of the cyber-physical component, the software part is composed of two main parts. The first part is the software representation of the physical object, *i.e.*, the

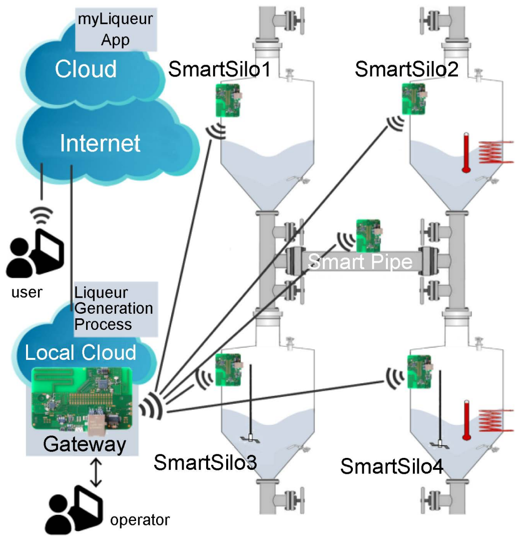

Figure 4. The Liqueur production system used as case study.

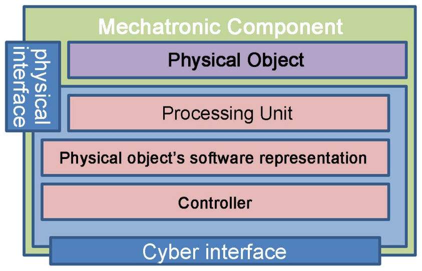

Figure 5. The architecture of the cyber-physical component.

mechanical unit, into the software domain. This part does not add any extra functionality; it only encapsulates the details of the integration of the physical world with the cyber world. On top of this, another part, the controller in Figure 5, transforms the physical object to a smart one adding extra functionality. This part encapsulates the low-level control of the physical object required to transform the physical world object into a smart cyber-physical component that provides its functionality through a well-defined interface.

We use the Interface construct of UML to specify the cyber interface of a cyber-physical component. The Interface is used in UML to declare a set of public features and obligations that together constitute a coherent service [37]. In this sense, an Interface specifies a contract that any instance of the mechatronic component shall fulfil. The UML class diagram of Figure 6 presents the interface of the smartSilo Mechatronic component in terms of provided and required interfaces. The SmartSiloUsageIf represents the provided interface while the SmartSiloUserIf represents the required one. In the required interface, we show how to model the interaction between SmartSilo and its client with the Signal and Reception constructs of UML in order to represent the possibly asynchronous nature of this interaction. Thus, the heatingCompleted and mixingCompleted

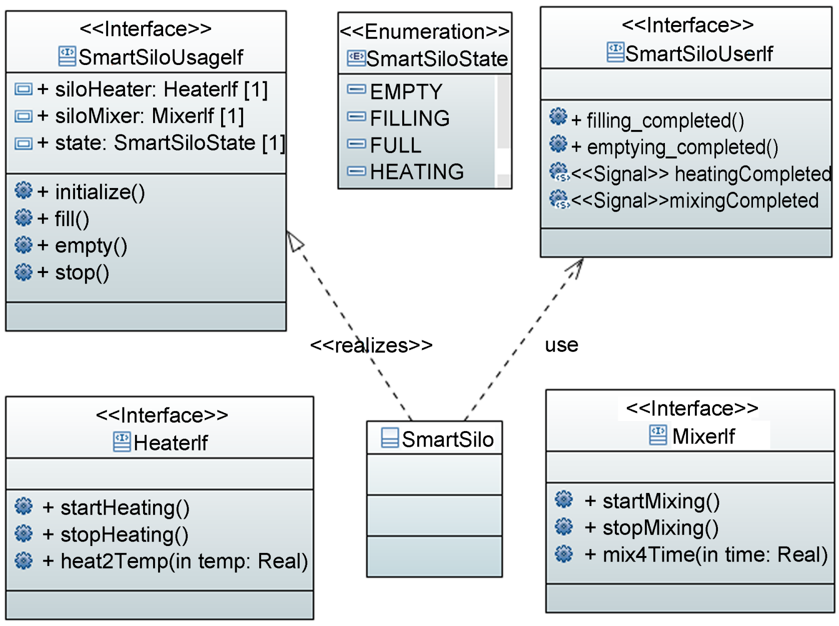

Figure 6. The cyber interface of the SmartSilo mechatronic component.

signals sent by the SmartSilo will trigger an asynchronous without a reply reaction to the SmartSilo client, e.g., the type A liqueur generation process, through the corresponding Receptions captured in the SmartSiloUserIf.

4.3. Towards a Contiki-Based Industrial Automation Thing

The 6LoWPAN IoT gateway of Weptech electronic Gmbh running the Contiki operating system is used to host the controller of the smartSilo mechatronic component. The 6LoWPAN IoT gateway, which is based on an ARM®Cortex®–M3 SoC with 512 kB Flash and 32 kB RAM, functions as a border router in a 6LoWPAN network. It connects a wireless IPv6 network, over an 802.15.4 compliant radio interface in the 2.4 GHz band, to the Internet via a 10BASE-T Ethernet interface.

Contiki is a lightweight operating system ported to various microcontroller architectures on resource constraint devices. It was selected mainly for its event-driven kernel that guaranties fast response times to events and to its ability for dynamic loading and replacement of individual programs and services that leads to very flexible Mechatronic components whose behaviour may be modified during run-time. The interfacing of the cyber part with the physical one, has been developed using the event-driven handling mechanism of Contiki to get a better response time compared to the traditional scan cycle approach mainly used in industry. Sensor signals generate interrupts which are handled by Contiki and transformed to asynchronous software events. These Events are broadcasted and captured by the corresponding event handling routines. Thus, the high-level sensor signal is transformed to the highLevelReached asynchronous event. This event is handled by the corresponding event handling routine, which is responsible to implement the sensor data handling algorithm. The data handling algorithm sends among others a close signal to the inValve and activates the sending of a fillingCompleted event to the client of the mechatronic component, *i.e.*, the liqueur generation process.

The response of the system from the time that the sensor generates the signal to the time that the signal arrives to the inValve actuator has an average value of 39.20 μs. Listing 1 presents a part of the object-based C implementation of the smart silo cyber part that is related with the interface of the component with its environment. This implementation is for the case that the required interface will be modelled by call back functions instead of signals and receptions. It is evident that both alternatives, *i.e.*, signals or call back functions, imply a tight coupling among the smartSilo and the components that use its behaviour, in the sense that these interfaces should be known in advance for the development of the component's clients.

A model-to-model transformer is required to automate the process of generating the IAT. This transformer will use as input the properly annotated with the DSML conventional mechatronic component. One approach is to mark the UML design specification of the mechatronic component with the stereotypes defined by the UML4IoT profile. If a UML design is not available then the source code of the cyber part of the mechatronic component is properly annotated with specific annotations that have been defined based on the UML profile. An example of annotated code with Java-like annotations is given in Listing 2,

```
struct silo{
  enum silo_state state;
  struct level_sensor *high_level_sensor,*low_level_sensor;
  struct valve *in_valve,*out_valve;
  ...
  int32_t target_temperature;
  // provided If
  void (*initialize)(void);
  void (*fill)(void);
  void (*empty)(void);
  void  (*stop)(void);
  void (*heat)(void);
  void (*mix)(void);
  // required If
  void (*filling_completed)(void);
  void (*emptying_completed)(void);
```

Listing 1. Part of the C object-based implementation of the cyber part of the mechatronic component.

```
struct valve//@ObjectType (name="valve",id=1664,instanceType=multiple,mandatory=true)
{
  int state;//@ResourceDef (id=5850,name="state",operations=R,type=string,instanceType=single,mandatory=true)
  int io;
  void (*valve_open)(void);//@ResourceDef (id=5851,name="valve_open",operations=E,type=void,instanceType=single,mandatory=true)
  void (*valve_close)(void);//@ResourceDef (id=5852,name="valve_close",operations=E,type=void,instanceType=single,mandatory=true)
};

struct level_sensor//@ObjectType (name="LevelSensor",id=1665,instanceType=multiple,mandatory=true)
{
  int state;//@ResourceDef (id=5550,name="state",operations=R,type=void,instanceType=single,mandatory=true)
};

struct silo{//@ObjectType (name="SmartSilo",id=1663,instanceType=single,mandatory=true)
  enum silo_enum state;//@ResourceDef (id=0,name="state",operations=R,type=string,instanceType=single,mandatory=true)
  int (*fill_op)(void);//@ResourceDef (id=1,name="fill",operations=E,type=void,instanceType=single,mandatory=false)
  int (*empty_op)(void);//@ResourceDef (id=2,name="empty",operations=E,type=void,instanceType=single,mandatory=false)
  int (*initialize_op)(void);//@ResourceDef (id=4,name="initialize",operations=E,type=void,instanceType=single,mandatory=false)
  struct _level_sensor high_level_sensor,low_level_sensor;
  struct _valve in_valve,out_valve;
  int filling_completed;//@ResourceDef (id=7,name="filling_completed",operations=R,type=boolean,instanceType=sinle,mandatory=false)
  int emptying_completed;//@ResourceDef (id=8,name="emptying_completed",operations=R,type=boolean,instanceType=single,mandatory=false)
};
```

Listing 2. Part of the C object-based implementation of the cyber part of the mechatronic component annotated with the UML4IoT java-like annotations.

where objects and object types of the cyber-physical component as well as their properties that should be exported to the IoT environment are properly annotated.

In Figure 7, the Contiki-based silo industrial automation thing developed with the proposed approach is shown. In the current implementation, the Weptech embedded board is used as processing unit while a hardware simulator is used for the silo. The Raspberry Pi and the XDK of Robert Bosch are alternative supported platforms.

4.3. The Interfaces of the Industrial Automation Thing

Adopting the OMA LWM2M application protocol, the interface of the Industrial Automation Thing is well defined and independent of the behavior that is implemented by the component. This interface is defined using UML provided and required interfaces as shown in Figure 8. Based on this figure the IAT has three provided interfaces and three required that are independent of the nature of the component. This feature combined with the ability of Contiki to dynamically load and replace individual services, results to a completely flexible component

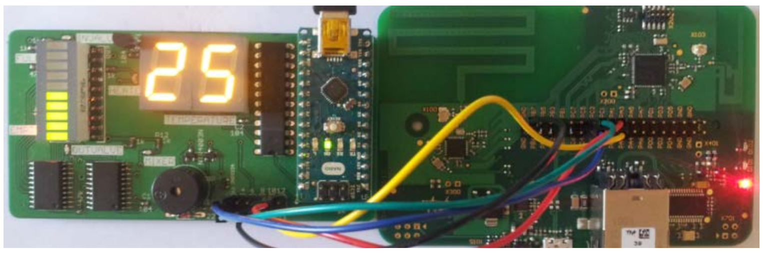

Figure 7. The silo industrial automation thing based on a hardware silo simulator.

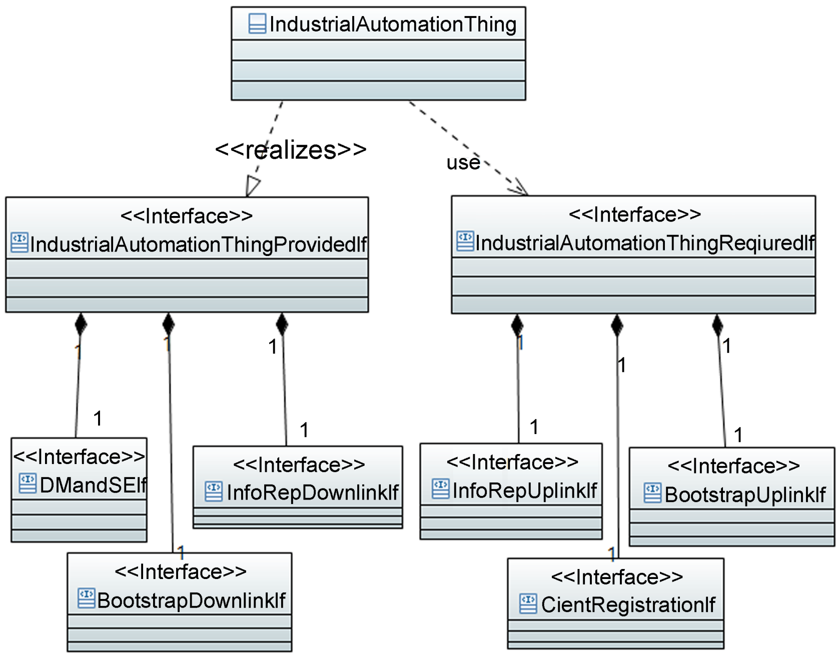

Figure 8. Industrial automation thing interfaces (provided and required).

regarding its behavior. New or replaced behavior can be activated by the same well-defined REST interface of the Industrial Automation Thing.

A comparison with Figure 6 that captures the conventional mechatronic component interface points out the flexibility of the AIT compared to the conventional one. Through this REST interface, resources may be created and used on demand based on requirements assuming that the physical part supports the requested new behavior. Resources, Resource Instances, Objects, Object Instances which are exposed by the IAT as well as their attributes, are accessed by the clients of the IAT through the device management and service enablement interface (*DM&SE If*). IPSO smart objects [38] have been adopted in this work to satisfy the requirement for interoperability.

5. Automating the Generation Process of IAT

For the automation of the generation process of the IAT a model-to-model transformer has been implemented. This transformer accepts as input the annotated legacy source code and applies a set of transformation rules properly defined to get the source code of the IAT.

5.1. The Model-to-Model Transformer

A prototype model-to-model transformer was developed based on Autogen and GNU Guile. Autogen (https://www.gnu.org/software/autogen/) is a GNU tool that supports creation and maintenance of source code. As shown in Figure 9, Autogen takes as input the annotated legacy code of the smart object, a template file and the definitions file. The template file implements the set of transformation rules. It actually defines the structure of the textual output of Autogen, *i.e.*, the structure of the IAT source code, making use of autogen's macro-type format and embedded scheme code. The definitions file provides the information required to instantiate the template file. More specifically, it includes the LWM2M object/resource/instance properties and other source-level information. The definitions file can be generated by the appropriate scheme procedures

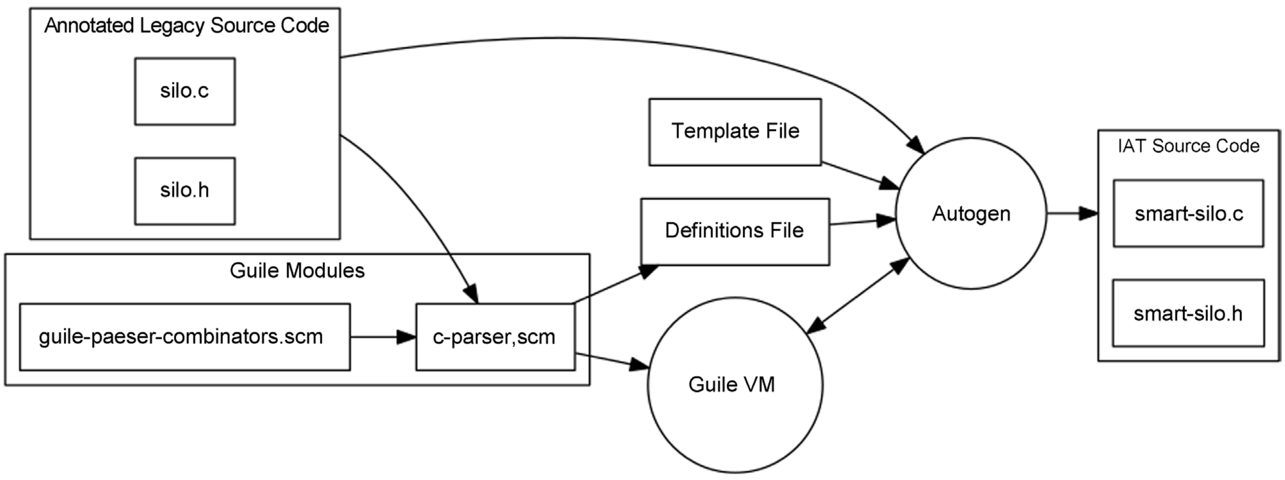

Figure 9. Transformation process of the IAT.

of the c-parser Guile module that uses as input the annotated legacy source code, as shown in Figure 9. The Guile module that implements the c-parser scheme procedures was developed as a high-level recursive descent parser, based on the guile-parser-combinators module (https://git.dthompson.us/guile-parser-combinators.git). GNU Guile is an implementation of the Scheme programming language (https://www.gnu.org/software/guile/). Autogen collaborates with the Guile VM through guile procedure calls embedded in the template file. The output of the transformation process is the source code of the IAT in a format ready to be compiled with the Contiki operating system and then deployed.

5.2. Transformation Rules

The following rules that apply for each object of the lwm2m client have been defined.

Rule 1:Create wrapper functions for annotated behaviors.

For each function with the BehaviorResource annotation create a wrapper function with input parameters:

lwm2m_context_t *ctx, const uint8_t *arg, size_targsize, uint8_t *outbuf, size_t outsize

E.g., Source:static int fill(void);

Target: static intfill(lwm2m_context_t *ctx, const uint8_t *arg, size_targsize, uint8_t *outbuf, size_t outsize);

Rule 2:Create getters/setters functions for each property annotated with the *PrimitiveRes* annotation.

For each attribute annotated with the PrimitiveRes annotation create the corresponding read and write function depending on the applied operations on the attribute defined in the annotation.

E.g., for the silo_state property

staticintget_silo_state(lwm2m_context_t *ctx, uint8_t *outbuf, size_t outsize) {

char *value;

value = get_silo_state_inString(silo->state);

returnctx->writer->write_string(ctx, outbuf, outsize, value, strlen(value));}

Rule 3:Construct the Resource model.

For each annotated attribute or function create an entry using the LWM2M_RESOURCE_CALLBACK macro of the lwm2m implementation which is integrated into the Contiki OS. E.g., for the silo_state

LWM2M_RESOURCE_CALLBACK(0,{get_silo_state, NULL, NULL}),

Append this entry to a list of resources, *i.e.*, *silo_resources*, using the LWM2M_RESOURCES macro.

Create the corresponding object instance using the LWM2M_INSTANCE macro and register the resources. E.g., LWM2M_INSTANCE(0, silo_resources)

Append it to the list of silo instances using the LWM2M_INSTANCES.

E.g.,LWM2M_INSTANCES(silo_instances, ….);

Create the Object and registers its instances using the LWM2M_OBJECT macro. E.g.,

LWM2M_OBJECT(silo_obj, 1663, silo_instances);

Rule 4:Modify setter functions.

For each attribute annotated with the ObservableResource annotation, modify its setter function (set_<attribute name>())by appending a call to the lwm2m_object_notify_observersfunction.

Assumption: For each observable attribute a setter function exists. E..g.,

```
Source: void set_filling_completed(){
  silo->filling_completed = 1;}
Target: void set_filling_completed(){
  silo->filling_completed = 1;
lwm2m_object_notify_observers(&silo_obj, "/0/7");    }
```

Rule 5:Generate and handle the initialize function for the object.

*5.1 Generate an initialize function to initialize the legacy object and register it to lwm2m by a call to the*lwm2m_ engine_register_object *function.*

The legacy initialize function of the object should be properly annotated. E.g.,

```
void ipso_ silo_init(void) {
  silo_init(); // legacy object initialization function
  lwm2m_engine_register_object(&silo_obj); }
```

*5.2 Append the initialize function prototype to the*ipso-objects.h*file. E*.g.

```
void ipso_silo_init(void);
```

*5.3 Append a call statement to the initialize function of each object to the*ipso_objects_init()*function body of the*ipso-objects.c*file.* E.g.,

```
void ipso_objects_init(void) {
ipso_silo_init();
```

6. Conclusion

In this paper, we consider the tight integration of the physical world with the cyber one at the mechatronic component level. A mechatronic component offers its functionality through well-defined mechanical, electrical and software interfaces. In this sense the industrial automation system is a composition of mechatronic components along with cyber components and humans. IoT is adopted for the integration of these components to exploit the benefit of this technology and UML4IoT is utilized to automatically transform the conventional mechatronic component into an IoT compliant cyber-physical one, *i.e.*, to an Industrial Automation Thing. The IoT model used in the UML4IoT approach is extended towards a complete IoT model for the manufacturing domain. The transformation rules required for the development of the model-to-model transformer have been developed and validated through a prototype implementation of the liqueur production laboratory system. The prototype implementation of the silo industrial automation thing based on a Contiki enabled embedded board and the C language is used to demonstrate the applicability and the effectiveness of the proposed approach.

References

[1] UNESCO Chair (2016) Home Page. On Mechatronics and Mechatronics Research and Application Center. http://mecha.ee.boun.edu.tr/

[2] Thramboulidis, K. (2005) Model Integrated Mechatronics—Towards a new Paradigm in the Development of Manufacturing Systems. *IEEE Transactions on Industrial Informatics*, **1**, 54-61. https://doi.org/10.1109/TII.2005.844427

[3] Thramboulidis, K. and Scholz, S. (2010) Integrating the 3+1 SysML View Model with Safety Engineering. 2010 *IEEE Conference on Emerging Technologies and Factory Automation* (*ETFA*), Bilbao, Spain, 13-16 September 2010, 1-8. https://doi.org/10.1109/etfa.2010.5641353

[4] Thramboulidis, K. (2012) Overcoming Mechatronic Design Challenges: The 3+1 SysML-View Model. *The Computing Science and Technology International Journal*, **2**, 6-14.

[5] FESTO (2016) Industry 4.0: Efficient Engineering Processes with "OPAK". https://www.festo.com/net/en_corp/SupportPortal/MobilePressDetails.aspx?documentId=368146&q=

[6] OASIS (2016) Devices Profile for Web Services (DPWS) Specification. http://docs.oasis-open.org/ws-dd/ns/dpws/2009/01

[7] Angulo, P., Guzmán, C.C., Jiménez, G. and Romero, D. (2016) A Service-Oriented Architecture and Its ICT-Infrastructure to Support Eco-Efficiency Performance Monitoring in Manufacturing Enterprises. *International Journal of Computer Integrated Manufacturing*, **30**, 202-214. https://doi.org/10.1080/0951192X.2016.1145810

[8] Morgan, J. and O'Donnell, G.E. (2015) Enabling a Ubiquitous and Cloud Manufacturing Foundation with Field-Level Service-Oriented Architecture. *International Journal of Computer Integrated Manufacturing*, **30**, 442-458.

[9] He, W. and Xu, L. (2015) A State-of-the-Art Survey of Cloud Manufacturing. *International Journal of Computer Integrated Manufacturing*, **28**, 239-250. https://doi.org/10.1080/0951192X.2013.874595

[10] Ghimire, S., Luis-Ferreira, F., Nodehi, T. and Jardim-Goncalves, R. (2016) IoT Based Situational Awareness Framework for Real-Time Project Management. *International Journal of Computer Integrated Manufacturing*, **30**, 74-83. https://doi.org/10.1080/0951192X.2015.1130242

[11] Ren, L., Zhang, L., Wang, L., Tao, F. and Chai, X. (2014) Cloud Manufacturing: Key Characteristics and Applications. *International Journal of Computer Integrated Manufacturing*, **30**, 501-515. https://doi.org/10.1080/0951192X.2014.902105

[12] Bi, Z., Xu, L.D. and Wang, C. (2014) Internet of Things for Enterprise Systems of Modern Manufacturing. *IEEE Transactions on Industrial Informatics*, **10**, 1537-1546. https://doi.org/10.1109/TII.2014.2300338

[13] Bradley, D., Russell, D., Ferguson, I., Isaacs, J., MacLeod, A. and White, R. (2015) The Internet of Things—The Future or the End of Mechatronics. *Mechatronics*, **27**, 57-74.

[14] Thramboulidis, K. and Christoulakis, F. (2016) UML4IoT—A UML Profile to Exploit IoT in Cyber-Physical Manufacturing Systems. *Computers in Industry*, **82**, 259-272.

[15] Dunkels, A., Gronvall, B. and Voigt, T. (2004) Contiki—A Lightweight and Flexible Operating System for Tiny Networked Sensors. 29*th Annual IEEE International Conference on Local Computer Networks*, 16-18 November 2004, 455-462. https://doi.org/10.1109/lcn.2004.38

[16] Thramboulidis, K. (2015) A Cyber-Physical System-Based Approach for Industrial Automation Systems. *Computers in Industry*, **72**, 92-102.

[17] Eclipse (2016) Papyrus for IoT—A Modeling Solution for IoT. https://www.eclipse.org/community/eclipse_newsletter/2016/april/article3.php

[18] Open Mobile Alliance (OMA) (2015) Lightweight Machine to Machine Technical Specification. OMA-TS-LightweightM2M-V1_0-20151214-C, Candidate Version 1.0. 14 December 2015.

[19] Kovatsch, M., Duquennoy, S. and Dunkels, A. (2011) A Low-Power CoAP for Contiki. *8th IEEE International Conference on Mobile Ad-Hoc and Sensor Systems*, Valencia, Spain, 17-21 October 2011, 855-860. https://doi.org/10.1109/MASS.2011.100

[20] Khaleel, H., Conzon, D., Kasinathan, P., Brizzi, P., Pastrone, C., *et al.* (2015) Heterogeneous Applications, Tools, and Methodologies in the Car Manufacturing Industry through an IoT Approach. *IEEE Systems Journal*, **PP**, 1. https://doi.org/10.1109/JSYST.2015.2469681

[21] Tao, F., Cheng, Y., Xu, L.D., Zhang, L. and Li, B.H. (2014) CCIoT-CMfg: Cloud Computing and Internet of Things-Based Cloud Manufacturing Service System. *IEEE Transactions on Industrial Informatics*, **10**, 1435-1442. https://doi.org/10.1109/TII.2014.2306383

[22] Bauer, M., Bui, N., De Loof, J., Magerkurth, C., Nettstrter, A., Stefa, J. and Walewski, J. (2013) Iot Reference Model. In: Bassi, A., *et al.*, Eds., *Enabling Things to Talk*, Springer, Berlin, Heidelberg, 113-162. https://doi.org/10.1007/978-3-642-40403-0_7

[23] IoT-A (2016) Final Architectural Reference Model for the IoT. http://www.meet-iot.eu/deliverables-IOTA/D1_5.pdf

[24] Tao, F., Zuo, Y., Xu, L.D. and Zhang, L. (2014) IoT-Based Intelligent Perception and Access of Manufacturing Resource toward Cloud Manufacturing. *IEEE Transactions on Industrial Informatics*, **10**, 1547-1557. https://doi.org/10.1109/TII.2014.2306397

[25] Diaz-Cacho, M., Delgado, E., Falcon, P. and Barreiro, A. (2015) IoT Integration on Industrial Environments. 2015 *IEEE World Conference on Factory Communication Systems* (*WFCS*), Palma de Mallorca, 27-29 May 2015, 1-7. https://doi.org/10.1109/wfcs.2015.7160553

[26] Broy, M., Kirstan, S., Krcmar, H., Schätz, B. and Zimmermann, J. (2013) What Is the Benefit of a Model-Based Design of Embedded Software Systems in the Car Industry? In: *Software Design and Development*: *Concepts*, *Methodologies*, *Tools*, *and Applications*, IGI Global, 310-334.

[27] Nguyen, X.T., Tran, T., Baraki, H. and Geihs, K. (2015) FRASAD: A Framework for Model-Driven IoT Application Development. 2015 *IEEE 2nd World Forum on Internet of Things* (*WF-IoT*), Milan, 14-16 December 2015, 387-392. https://doi.org/10.1109/WF-IoT.2015.7389085

[28] Malavolta, I. and Muccini, H. (2014) A Study on MDE Approaches for Engineering Wireless Sensor Networks. *40th EUROMICRO Conference on Software Engineering and Advanced Applications* (*SEAA*), Verona, 27-29 August 2014, 149-157. https://doi.org/10.1109/seaa.2014.61

[29] Conzon, D., Brizzi, P., Kasinathan, P., Pastrone, C., Pramudianto, F. and Cultrona, P. (2015) Industrial Application Development Exploiting IoT Vision and Model Driven Programming. 2015 *18th International Conference on Intelligence in Next Generation Networks* (*ICIN*), Paris, 17-19 February 2015, 168-175. https://doi.org/10.1109/icin.2015.7073828

[30] Prehofer, C. (2015) Models at REST or Modelling RESTful Interfaces for the Internet of Things. *IEEE 2nd World Forum on Internet of Things* (*WF-IoT*), Milan, 14-16 December 2015, 251-255.

[31] Eterovic, T., Kaljic, E., Donko, D., Salihbegovic, A. and Ribic, S. (2015) An Internet of Things Visual Domain Specific Modeling Language Based on UML. 2015 *XXV International Conference on Information*, *Communication and Automation Technologies* (*ICAT*), Sarajevo, 29-31 October 2015, 1-5. https://doi.org/10.1109/icat.2015.7340537

[32] Zhang, Y., Zhang, G., Wang, J., Sun, S., Si, S. and Yang, T. (2015) Real-Time Information Capturing and Integration Framework of the Internet of Manufacturing Things. *International Journal of Computer Integrated Manufacturing*, **28**, 811-822. https://doi.org/10.1080/0951192X.2014.900874

[33] De, S., Barnaghi, P., Bauer, M. and Meissner, S. (2011) Service Modelling for the Internet of Things. 2011 *Federated Conference on Computer Science and Information Systems* (*FedCSIS*), Szczecin, 18-21 September 2011, 949-955.

[34] Haller, S. (2010) The Things in the Internet of Things. *Internet of Things Conference* 2010, Tokyo, 29 November-1 December 2010.

[35] Ferry, P., *et al.* (2014) IoTLink: An Internet of Things Prototyping Toolkit. *IEEE International Conference on Ubiquitous Intelligence and Computing*, Bali, 9-12 December 2014, 1-9.

[36] Basile, F., Chiacchio, P. and Gerbasio, D. (2013) On the Implementation of Industrial Automation Systems Based on PLC. *IEEE Transactions on Automation Science and Engineering*, **10**, 990-1003. https://doi.org/10.1109/TASE.2012.2226578

[37] OMG (2015) Object Management Group, OMG Unified Modeling Language (OMG UML), Version 2.5, OMG Document Number Formal/2015-03-01. http://www.omg.org/spec/UML/2.5

[38] Internet Protocol for Smart Objects (IPSO) Alliance (2014) IPSO Smart Object Guideline. IPSO Smart Object Committee, 21 September. https://www.ipso-alliance.org/ipso-community/resources/smart-objects-interoperability/