INTEROPERABILITY BY MEANS OF CONFIGURABLE CONNECTORS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

MUHAMMED ÇAĞRI KAYA

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF DOCTOR OF PHILOSOPHY
IN
COMPUTER ENGINEERING

MAY 2020

Approval of the thesis:

INTEROPERABILITY BY MEANS OF CONFIGURABLE CONNECTORS

submitted by **MUHAMMED ÇAĞRI KAYA** in partial fulfillment of the requirements for the degree of **Doctor of Philosophy in Computer Engineering Department, Middle East Technical University** by,

Prof. Dr. Halil Kalıpçılar
Dean, Graduate School of **Natural and Applied Sciences** _____

Prof. Dr. Halit Oğuztüzün
Head of Department, **Computer Engineering** _____

Prof. Dr. Ali H. Doğru
Supervisor, **Computer Engineering, METU** _____

Examining Committee Members:

Prof. Dr. Halit Oğuztüzün
Computer Engineering, METU _____

Prof. Dr. Ali H. Doğru
Computer Engineering, METU _____

Prof. Dr. Bedir Tekinerdoğan
Information Technology, Wageningen University _____

Assist. Prof. Dr. Pelin Angın
Computer Engineering, METU _____

Assist. Prof. Dr. Gül Tokdemir
Computer Engineering, Çankaya University _____

Date:

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Surname: Muhammed Çağrı Kaya

Signature :

ABSTRACT

INTEROPERABILITY BY MEANS OF CONFIGURABLE CONNECTORS

Kaya, Muhammed Çağrı
Ph.D., Department of Computer Engineering
Supervisor: Prof. Dr. Ali H. Doğru

May 2020, 109 pages

A configurable connector-based software development methodology for component-based approaches is presented. This method involves the incorporation of variability modeling capabilities into component modeling environments. The focus of this research is on supporting technologies for the combination of parts that are not directly compatible. In the scope of this research, firstly, proposals for the configurable connector paradigm are put forth, that are, achieving interoperability among system components by using existing connectors to increase reuse and customizing them through simple user interfaces. This methodology is applied to the Live-Virtual-Constructive simulation systems domain as a configurable gateway application between Data Distribution Service for Real-Time Systems (DDS) and High-Level Architecture (HLA) standards. Finally, interoperability of different parties are investigated for Metrology and the calibration industry, and an Industrial Internet of Things-based architecture is established. Academic and industrial case studies have been conducted for proof of concept. They show the practicality of the proposed approaches.

Keywords: component, connector, interoperability, software architecture, variability

ÖZ

YAPILANDIRILABİLİR BAĞLAYICILAR ARACILIĞIYLA BİRLİKTE ÇALIŞABİLİRLİK

Kaya, Muhammed Çağrı
Doktora, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi: Prof. Dr. Ali H. Doğru

Mayıs 2020 , 109 sayfa

Bu tezde bileşen tabanlı yazılım geliştirme yaklaşımları için yapılandırılabilir bağlayıcı tabanlı bir yöntem sunulmaktadır. Araştırmanın odak noktası, doğrudan uyumlu olmayan parçaların tümleştirilmesi için destekleyici teknolojilerin irdelenmesidir. Bu kapsamda, sistem bileşenleri arasında birlikte çalışabilirlik sağlanırken var olan bağlayıcıları kullanmayı önceleyen ve yapılandırılmalarını basit kullanıcı arayüzleri yoluyla yapmayı kapsayan bir yaklaşım önerilmektedir. Bu yöntemle Canlı-Sanal-Yapısal benzetim sistemleri alanında Gerçek Zamanlı Sistemler için Veri Dağıtım Hizmeti (DDS) ve Yüksek Seviye Mimari (HLA) standartları arasında yapılandırılabilir bir ağ geçidi geliştirilmiştir. Son olarak, Metroloji ve kalibrasyon alanında farklı paydaşların birlikte çalışabilirliğini sağlamak için araştırmalar yapılmış ve Nesnelerin Endüstriyel İnterneti (IIoT) tabanlı bir mimari oluşturulmuştur. Yapılan akademik ve endüstriyel vaka çalışmaları önerilen yöntemlerin uygulanabilirliğini göstermektedir.

Anahtar Kelimeler: bağlayıcı, bileşen, birlikte çalışabilirlik, yazılım mimarisi

To Didar Duru

ACKNOWLEDGMENTS

I would like to thank my professors in my Thesis Monitoring Committee. I thank my advisor Prof. Ali H. Doğru for his support, guidance, and friendship throughout my graduate studies. I am thankful to Prof. Halit Oğuztüzün for his guidance, help, and collaboration. I also thank Prof. Bedir Tekinerdoğan for his constant support and guidance despite his busy schedule. Being exceptional characters and professionals, they are my role models in my academic life.

I thank Assist. Prof. Pelin Angın and Assist. Prof. Gül Tokdemir for their useful comments on my thesis work. I am also thankful to all academic and departmental staff of Computer Engineering Department for their friendship.

I am thankful to Dr. Selma Süloğlu and other members of our research group for their collaboration and help. I would like to thank Mahdi Saeedi Nikoo for his friendship and our research collaboration. I also thank Alperen Eroğlu, Özcan Dülger, Anıl Çetinkaya, Alper Karamanlıoğlu, Gökhan Özsarı, Alperen Dalkıran, and Tuğberk İşyapar for their friendship and support in many ways for years.

Last but not least, I want to express my gratitude to my family for their patience and endless support.

The work described in this thesis is partially supported by The Scientific and Technological Research Council of Turkey (TÜBİTAK) under Grant No. 3151162 entitled "DDS-Based Hybrid Avionic Emulator System Development (HAVESIS)".

TABLE OF CONTENTS

APPENDICES

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

LIST OF ABBREVIATIONS

ABBREVIATIONS

AB	Accreditation Body
AmI	Ambient Intelligence
API	Application Programming Interface
AWS	Amazon Web Services
BPEL	Business Process Execution Language
BPMN	Business Process Modelling Notation
CBSE	Component-Based Software Engineering
CL	Calibration Laboratory
CMC	Calibration and Measurement Capability
COSE	Component-Oriented Software Engineering
COSEML	Component-Oriented Software Engineering Modeling Language
DDS	Data Distribution Service for Real-Time Systems
DIS	Distributed Interactive Simulation
DUT	Device Under Test
FOM	Federation Object Model
GCL	Gateway Configuration Language
GML	Gateway Mapping Language
GPIB	General Purpose Interface Bus
GUI	Graphical User Interface
HLA	High-Level Architecture
IaaS	Infrastructure as a Service
IDL	Interface Description Language
IEC	International Electrotechnical Commission

IIC	Industrial Internet Consortium
IIoT	Industrial Internet of Things
IoMT	Internet of Measurement Things
IoT	Internet of Things
ISO	International Organization for Standardization
LVC	Live-Virtual-Constructive
LVCAR	Live-Virtual-Constructive Architecture Roadmap
MathML	Mathematical Markup Language
MDD	Model-Driven Development
MII	Metrology Information Infrastructure
MVC	Model-View-Controller
MVVM	Model-View-ViewModel
NCSLI	National Conference of Standards Laboratories - International
OPC-UA	Open Plant Communication Universal Architecture
OVM	Orthogonal Variability Model
PaaS	Platform as a Service
PC	Personel Computer
QoS	Quality of Service
RTI	Runtime Infrastructure
SaaS	Software as a Service
SISO	Simulation Interoperability Standards Organization
SOA	Service Oriented Architecture
SoA	Scope of Accreditation
SOM	Simulation Object Model
SPL	Software Product Line
SQL	Structured Query Language
TES	Tactical Environment Simulation

UI	User Interface
UML	Unified Modeling Language
UoM	Units of Measure
UUT	Unit Under Test
VS	Visual Studio
WPF	Windows Presentation Foundation
XML	Extensible Markup Language

CHAPTER 1

INTRODUCTION

Already existing software components or even systems can be utilized in the composition of a new application. Such composition can even be carried out over the Internet. This is becoming a necessity for large-scale systems where developing them from scratch is costly and time-consuming. Moreover, not necessarily only for large-scale software, it is desirable to offer software development means for a targeted domain based on drag-and-drop activities to decrease costs and speed up time-to-market [1]. However, once a domain getting mature with pre-created software, it is inevitable to have components that are created for different purposes of usage, with different technologies of implementation and communication styles. The cost of this kind of development methodology is the burden of interoperability among system components.

The heterogeneity and the need for interoperability are noticeable problems for heterogeneous and distributed systems [2], either for a large system that is composed of different subsystems belonging to different areas of application, or a smaller system with components from the same domain. Considering the Internet of Things (IoT) as an example, the variety in the included units can easily be observed [3]. There is no limit to the types and capabilities of units and their communication styles that can be connected. Proposed solutions for this kind of problems include gateway technologies focusing on different levels of communication, such as *application* or *transport* [4, 5].

This thesis aims to provide interoperability solutions for component-based systems by managing the variability of architectural connectors and evolve them into gateways when possible. Besides, in a broader sense, other architectural solutions are also

investigated, focusing on the Industrial IoT (IIoT) domain.

The rest of this chapter clarifies the problem statement that the thesis aims to solve. Then, the contributions of the thesis are explained. The chapter is concluded with the outline of the thesis.

1.1 Motivation and Problem Definition

Managing interactions among components of a complex system is a challenging problem. The burden of interoperability is considerable especially when the system components are heterogeneous, such as in a smart city application [6]. This heterogeneity may be caused by using different subsystems in the composition of a large-scale system: using components developed through different technologies, system components belonging to different disciplines, etc. The well-known *separation of concerns* principle should be considered to overcome heterogeneity and reduce the complexity of developing software systems: All components or subsystems of the system should focus on their functionality and carry minimum concern if not none, about communicating with the rest of the system.

Developing all of the components of a domain, e.g., IoT, in a way that all of them have a common means of communication, such as a generic communication protocol, is not realistic and feasible. Thus, devices or components in a domain may have various implementation technologies, characteristics, power and computation constraints, and purposes of usage. Moreover, most of the systems in different application domains that we are dealing with do not have a static but a dynamic nature. For example, new components may be added and removed from the system at different stages.

Components should adapt their communication channels with interacting parties. This brings the duty of multi-interaction management to the component. This situation increases complexity and decreases the reusability of both components and connectors as the interaction logic is incorporated inside the communicating components. Following the separation of concerns principle, components should carry out their core functionality, and connectors should satisfy interaction needs. Thus a requirement of a highly reusable and dynamic infrastructure arises.

The idea of handling communication issues by architectural connectors comes to play to ease software development by ensuring the separation of concerns. Moreover, this is an established idea in the literature with more than a two-decade-old history; an early example of this kind of research is [7]. Most of the literature, however, focus on the development of connectors [8, 9] rather than choosing one of the existing and configuring it for customization.

1.2 Contributions and Novelties

The main contribution of this dissertation is related with the incorporation of configurable connectors to component-based approaches. In its specifics, this involves the incorporation of variability capability in the component-modeling approaches. Variability has already been practiced in Software Product Line Engineering. However, in general terms, this research is about supporting technologies for the combination of parts that are not directly compatible. Interoperability and adaptation are some associated concepts that are widely used. In other words, component connectors are fortified with the ability to be configured through variability-modeling to adapt incompatible components. This adaptation can serve the various dimensions of incompatibility and even supports the coordination of the components in terms of their invocation control.

The main topic addressed in the article [10] is about the off-the-shelf connectors. The topic introduces the need for such mechanisms that actually require proof of concept at various levels: connectors are utilized for components of different granularity, hence corresponding to different levels. Systems can be thought to be at the largest granularity level. As the accompanying work concerning this level, the gateway work [11] was developed. Here, platforms are adapted that can be considered as systems, and from another perspective, even higher-level than systems because many systems can be implemented on a platform. The configurable connector concept has been implemented as a gateway application that governs the connectivity between two middleware platforms, namely DDS (Data Distribution Service for Real-Time Systems) and HLA-RTI (High Level Architecture-Runtime Infrastructure).

In another aspect, the IIoT has been studied and reported in the related article [12].

As an IIoT-based architecture is proposed, this work can be assumed as comprising different levels of granularity from the devices at the physical level to the systems at the application level. Also, due to the popularity of IoT technologies, their specifics serves a valid arena for adaptability research. Heterogeneity is especially reported as a major problem in the IoT domain, hence suggesting this field as a very suitable experimentation candidate.

As a summary, the three publications that were produced as the side-products of this dissertation work can be classified as the main idea and two related implementations. The two implementations have targeted different levels for the purpose of covering the extremes for the broad area of implementation alternatives.

Contributions of this thesis research can be summarized as follows:

- A configurable connector based approach is proposed to provide interoperability for component-based systems.
- Based on the proposed configurable connector approach, a DDS-HLA gateway is developed and successfully deployed to an industrial application.
- An IIoT-based architecture is proposed, and its practicality is shown in the calibration domain to achieve interoperability in metrology applications.

1.3 The Outline of the Thesis

The rest of this thesis is constructed as follows:

- In Chapter 2, the configurable connector based methodology is explained. How interoperability can be provided for components of different domains are shown through academic case studies.
- Chapter 3 explains how the interoperability of LVC (Live-Virtual-Constructive) simulation systems is achieved through a configurable gateway approach. The gateway focuses on data interoperability between DDS and HLA. An industrial case study and other academic work have been conducted to show the practicality of the gateway approach.

- In Chapter 4, interoperability among different stakeholders is demonstrated in Metrology and the calibration domain. An IIoT based architecture is proposed to help to establish standards and make basic services available for everyone in the calibration industry. A “Scope of Accreditation” editor implementation is adapted to the proposed architecture.
- Chapter 5 concludes the thesis with remarks and possible future work.

CHAPTER 2

CONFIGURABLE CONNECTOR APPROACH

Solutions based on software components, especially for heterogeneous constituents such as those pertaining to different disciplines suffer the interoperability burden. Adaptor technologies have been introduced before, as a potential remedy and utilized here through implementing them in component connectors. The main objective of this research is to offer a consistent development environment that provides seamless development especially for the component heterogeneity cases. A set of connectors are introduced to a component-based development environment where a variability model drives the configuration mechanisms in the flow of the application, components, and connectors. The offered set of connectors are the enabling technology incorporated through their selection and configuration. As demonstrated through an example in this chapter, academic experimentation revealed the practicality of the approach. Required adaptation can be achieved in connectors as the appropriate constituent, avoiding additional functional load on the domain components.[1]

2.1 Introduction

Different approaches have been offered to software developers that exploit the attractive capabilities of components. In an ever growing world of demand, supplying software on time within expected quality has always been problematic and a source of motivation for software engineers to devise techniques for quickly developing dependable software. Currently, it is possible to classify such efforts into two categories as compositional [13] and generative [14]. Either venue aims to reduce human code

[1] The study described in this chapter was published in *Journal of Integrated Design and Process Science* in 2018 [10].

development which is known to be unpredictable and error-prone [15]. Components, therefore, emerge as an inevitable resource to offer reuse in a structured way that bypasses code writing. Of course, one should ignore the development of components themselves which is a lot easier and more predictable than the development of huge software-intensive systems. We assume the sufficient existence of necessary software components to support the desired level of production. We refer to a web-service also as a component unless in a context that is dependent on the specific technology. Between the two alternatives, generative techniques were experimented decades ago. However, their formal specification requirements and lack of powerful tools incapacitated such early attempts. Model-driven approaches, however, attained an impressive level of success today in the generative direction. Besides the existence of complete model-driven processes, it can be observed that many different approaches are also incorporating some model-driven techniques. Meanwhile, this study capitalizes on the other alternative that is compositional. Looking at the compositional techniques, it is possible to see the building blocks offered in the forms of component or web-service technologies at a satisfactory level of capability to enable software development by integration. However, such integration often requires code writing, at least for the 'glue code'.

In this chapter, efforts for increasing reuse of software connectors are presented. The main motivation is considering only bringing components and connectors together, rather than developing the required modules. A development environment is proposed that works based on the principles of the component-oriented approach. The goal is adding connectors to the system through a drag and drop manner using a graphical user interface (GUI). The necessary changes to adapt the connector to the system, in other words, altering the reused connector for customization is done through the GUI again using simple pop-up windows. Thus, the suggested development style may prevent or at least reduce the need for the glue code.

Adaptation through connectors becomes especially important when a diversity concerning the component domains is considered. Most of the software-intensive solutions to new demands involve interdisciplinary collaboration, including the components. Early statements concerning this phenomenon [16] indicate the consistent development for interdisciplinary capabilities. Actually, a futuristic version of the

concept manifests itself as transdisciplinary engineering [17]. Connectors could support the evolution from interdisciplinary to transdisciplinary development through enabling the early and maturing attempts.

Some background information will be presented in the following section, on critical constituents of compositional approaches. These constituents are mostly components and connectors. Currently, components are commonly being used as important technologies. Recently connectors have also been emphasized besides the components. Also, coordination is a related topic that has been addressed utilizing connectors and will be introduced along with component-oriented representations such as COSEML. A top-down strategy is emphasized for even coordination besides the structural decomposition. Related work of the proposed methodology is presented. Then, an example of the system development will be demonstrated after introducing the underlying methodology. Some remarks are provided in the discussion section before the chapter is concluded.

2.2 Background

This section includes the introduction of notions that form the foundation for the suggested approach. Mostly component related technologies and engineering approaches that exploit them are involved.

2.2.1 Components

A component can be defined as an implemented building block for software development. Components are used to yield a software system through composing from pre-implemented blocks based on a composition protocol. They are independently deployable. Meaning that the developers can use any components they desire without having to worry about the obligation to use any other specific component. Interfaces should accompany components to publish their provided, and sometimes required services in order for a system to be synthesized and managed efficiently.

Originally introduced to the literature by Szyperski [18] the component technologies

appeared as practical building blocks for software. Related capabilities were already experimented using different approaches. For example, Dynamic Link Libraries offered the capability to connect at run-time, for the units not included in the application. This mechanism is reminiscent of the requirement for a component hence a component-based system, that is the ability to integrate at run time. Even the earlier concept of 'virtual memory' management for programs that move in a piece of code to the memory from secondary storage (and moving out some other to vacate space) is also a distantly related mechanism.

The era corresponded to the maturing times for graphical user interfaces that supported event-based programs: the main loop would wait for mostly a mouse or a keyboard event to activate some function connected to the associated screen element. Although components are far more general software units than serving such 'windows-based' user interfaces or event-based program control, early component protocols reflected the requirements of that era. That is why the definition of a component structure included events to subscribe and events to publish. These are in addition to the fields that a 'class' would also declare such as properties and methods.

At first look, the components look just like objects, with the addition of event declarations that could even be judged as methods with specific synchronization requirements – so, no big difference. Even though such declarations take place at different locations: they are inside a class for objects, whereas they are inside additional units that are called interfaces for components. However, the contribution does not lie in the declaration structure. What makes a component useful is the hidden mechanisms that implement compliance with the 'component protocol'.

For efficient utilization of components, the protocol becomes very important. Soon, common operating systems started to serve such protocols, almost including 'middleware' layers in them. Today, common desktop environments include component protocols that can communicate with components before their integration to an application, and after their integration, during the operation to carry messages in either direction.

Early component models allocated only published methods. The required fields were exclusive to events. Later models made room for required methods as well as pub-

lished methods. Actually, a whole interface that included methods etc. could also be declared as published or required. The interface concept became more established with the components. Such improvements also have reflected to object-oriented modelling and got accommodated in, for example, UML 2.0 [19].

As our current infrastructures move towards more and more automated composition, concepts such as late binding, polymorphism, and dynamic configuration. gain more importance. Sets of components for different domains continue to be offered as well as component protocols or models. While specific capabilities are continuing to be improved and added, fundamental design principles should be preserved with more rigor – they include separation of concerns, cohesion etc. Meanwhile, improvement in the utilization of components, (composition) has continued to be experienced. Less and less code writing is on the agenda. In this research, connectors are emphasized as the other important constituent of the composition.

2.2.2 Connectors

Modern complex software systems are usually a collection of components. Handling of interactions among these components can be hard to manage because of the complexity of the systems. To fight the complexity, the well-known principle of separation of concerns should be considered. A component should not carry information about the rest of the system. Also, communication with its exterior should better be assumed by external entities. This is where the connectors come to play.

Connectors were defined by Mehta et al. [20] as an architectural element that is responsible for the handling of the interactions among the components in a software system. Connectors are classified into four general classes according to services they provide: communication, coordination, conversion and facilitation. Furthermore, eight connector types are defined according to the way in which they realize their roles in the interactions: procedure call, event, data access, linkage, stream, arbitrator, adaptor and distributor. In this way, connectors can be identified more effectively in a complex system and can be differentiated from one another.

2.2.3 Component-Based and Component-Oriented Software Engineering

Component-Based Software Engineering (CBSE) can be viewed as an approach that supports the integration of components into applications. The spectrum can also include mechanisms that aid this objective: component search ability is nice to have as well as the capability to build components. There can be mixed approaches allowing for example object-oriented development that facilitates insertion of components into the object-oriented models and eventually to the code. Component-Oriented Software Engineering (COSE) on the other hand, does not address code-level development. Other modeling notions such as objects are not retreated. Only component concept is regarded throughout the lifecycle: in abstract representations for the earlier phases and physical for the later phases.

COSE suggests a structural decomposition of the solution in terms of component definitions. Referred to as packages, such definitions correspond to component abstractions declared with the consideration in mind to match later with existing physical components. Comparable to the product engineering avenue of Software Product Line (SPL) approaches, the emphasis is not in constructing the components – rather, how to utilize existing components once an ordered product becomes the concern. The methodology is based on iterations on the decomposition model, including the abstractions and components until the 'logical' levels of decomposition are completely matched with existing components.

So far, we have discussed a static model. The decomposition represents the structure completely. Even connections among the required and published methods of various components can be depicted. However, a dynamic view indicating the activation order among these method connections (i.e. messages) is missing. This view was introduced later [21] in the form of a collaboration diagram inspired by UML, superimposed on the decomposition: Numbering was superimposed on the message connections. Further, a textual modeling language [22] for COSE included process flow primitives to be listed inside packages. Although not centralized, now the process model was represented as distributed among the packages of the system. This language grew toward including variability in also connectors, after the work of Kaya [23] that introduced variability to COSE.

COSEML

Component-Oriented Software Engineering Modeling Language (COSEML) is a graphical modeling language that is designed to take full advantage of COSE. It promotes the build by integration paradigm rather than code writing from scratch. In COSEML, design elements have their own graphical representations [24]. For instance, an interface of a component can hold information about its properties, method-in and method-out fields. Graphical representation of an interface and its structure is shown in Figure 2.1. Method-in fields contain the published methods of a component while method-out fields represent the methods to access the resources outside the component (required methods).

The language has evolved through the years. Originally it was designed as a tool for hardware-software co-design inspired by the Abstract Design Paradigm [25]. The initial version was introduced to support building software by integration, however without component technologies being around yet. Therefore, the earlier versions were more declarative to specify general purpose software units. Abstractions were designated for their compliance with data, function, or control dimensions of design. UML was not around either. Consistent usage and evolution accommodated widely accepted concepts, as they emerged. Icons were adapted to their UML correspondences where possible. With the proliferation of component technologies, the approach finally was united with its enabling technologies. The whole idea would make more sense if the well-established code could be coordinated by COSEML models, to yield executing systems. Bottom-level notions were completely adapted to components and interfaces. Over the years, only the structural element among the logical level primitives, namely the 'package' survived for use, shadowing the data, function, and control. Finally, the language was supported by a component-compliant process [26].

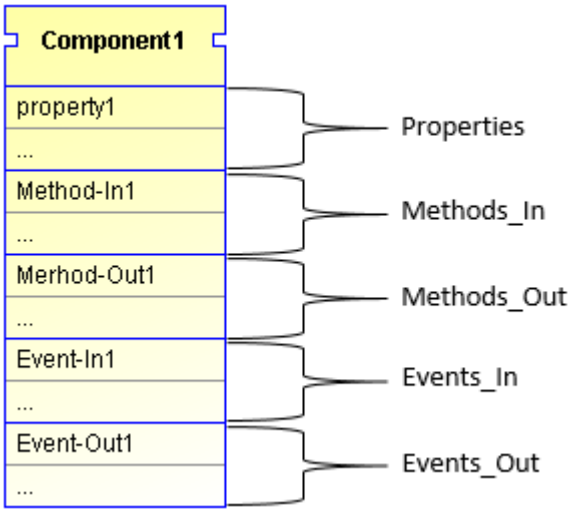

Figure 2.1: Graphical representation of an interface of a component.

The evolution kept following the improvement in the compositional approaches. Web services were added to the environment [27]. Later variability was added [28] as inspired by the SPL practices. The explicit modeling of the software process was improved to a mechanism that could mimic that of the SOA by including process flows [29]. Soon, connectors were reintroduced with serious roles to play [30]. The final enhancements now offered the enabling technologies for reusable connectors to complete the 'structural' spectrum for build by integration.

2.2.4 Coordination

Coordination can be described as managing data flow and exchanging control of the system or a resource among components in a software system. There are approaches that provide coordination through connectors. Also, some coordination languages are presented and protocols are defined especially for SOA [31].

Mehta et al. [20] categorize coordination as a major role of connectors as they connect interacting components in their research on connector classification. Coordination connectors are mainly responsible for conveying control among components according to their definition. They declare procedure call, event, and arbitrator as examples of coordination connector types.

Arbab [32] proposes Reo for the composition of software components. Reo is defined as a "channelbased exogenous coordination model". By composing simpler channels, complex coordinators can be built. Reo can be used as a 'glue code' language to compose software components or services to provide cooperative behaviour [33].

Linda [34] is proposed in the early 1980s as a coordination language. It is a communication and coordination model for parallel processes that operate as an ordered sequence. Another coordination language, Orc [35], is used for formal orchestration modeling. Also, Coordination Behavioral Structure (CBS) is proposed to formalize the service interactions and relationships [36].

The coordination concept is widespread in SOA literature. Papazoglou [37] describes coordination as a function provided by a service aggregator that encompasses other services to behave like a single composite service. This requires controlled execution of the included services and supervision of the dataflow among them.

Web Services Coordination (WS-Coordination) was developed as a framework to provide a coordinator and a set of coordination protocols for distributed web services [38]. WSAtomicTransaction [39] and WS-BusinessActivity [40] are covered in this framework to support short-duration and long-duration activities, respectively. The framework is extendable and can be used along with other protocols for web service domains.

To be able to offer off-the-shelf connectors, a set of connectors that are capable of serving any kind of connection needs should be available. This is not possible unless offered in a classification with a degree of modification capability. Luckily [20] laid out the fundamental expectations from connectors. Basically, a connection intermediates a function call. Functions are referred to as methods if they are members of an object, a component, or a web-service. Also starting with object orientation, the calls are supposedly carried by messages serving distributed requirements. Assuming the duty of such a connection, a connector should know the calling party and the method to call. On the other hand, a component should be system agnostic. In other words, it is developed to be used in different systems. Therefore, components may not know the exact names of the other components, the methods in them etc. yielding them dependent on connectors or the glue code to work together in a system. If such infor-

mation can be bound to the connectors during their configuration, it will be possible to use a small set of off-theshelf connectors constituting the only required connection technology. Here, a connector is mounted between a pair of components, or one end can be connected to the central process. It serves general purposes including the duties stated in [20]. Also, recent studies [41, 3] incorporated variability management in the connectors offering this 'configuration capability' as a systematic step in the development.

Connectors provide necessary services while also performing the basic message transfer duty. If the fundamental message mechanisms are taken for granted, the specific task for the connectors can be simplified as the adaptation. Actually, the categorization study addresses two issues that are synchronization and adaptation, although the former one can be thought of a kind of the latter. Messaging and synchronization have been studied in various other platforms whereas adaptation gained special attention since the introduction of components. An analytical study has been conducted by Jololian and Tanik [42] that considers adaptation as the main concern in composition and can be investigated in three dimensions that are data, function, and control. These dimensions are very fundamental for computer science, accounting for the modeling infrastructure for any kind of executable system [43].

Our abstraction of the connector duties includes coordination and adaptation. The categories and types of connectors defined in the work of Mehta et al. [20] can be viewed in this scope. While higher-level aspects of coordination are left to the central process, communication protocol level details are handled by the connectors. Sometimes the adaptation could relate to coordination. Different synchronization styles may need to be adapted. In this case, the coordination is the issue to be solved through an adaptation in the control dimension: the connector needs to provide a buffer to save the answer until the requestor is ready to receive it.

As a result, with the definition of new connector structures, an important missing link is substituted towards the realization of 'build by integration' paradigm. The central process can be defined in a graphical model. Existing components can be connected to this process through configuration-level input. Connectors will also be needed to establish communication among components, and in some cases between

a component and the central process. Components and connectors are also subject to configuration. Configurations will be derived via the variability modeling that is superimposed on the process and component model views.

2.3 Related Work

Studies that point out the importance of interoperability among components from different disciplines exist in the literature. One of them is the work of Costin and Eastman [44]. They describe the need for interoperability for smart and sustainable urban systems. They introduce a review including principles, methods, and requirements. According to the authors, semantic web technologies are promising for achieving interoperability. There are also efforts to achieve interoperability among the components developed for the same discipline by different producers, which causes heterogeneity. For example, Emmer et al. [45] introduce an approach for interoperability in the field of 3D measurement data management (MDM). There are diverse measurement equipment and measurement software from different producers. Therefore, MDM provides integration of information flow through the use of neutral data formats. They define a measurement data interface for this purpose.

The idea of adapting the connectors instead of components in case of incompatibility is proposed by Garlan [7]. This adaptation can be done by using higher-order connectors, operators that take connectors as parameters and produce them as a result. The aim is generating powerful and customized connectors by using existing ones.

Dashofy et al. [46] provide an approach for implementing connectors by using OTS middleware. This work differs from the proposed approach in this paper: while their concern is more related to implementing connectors, ours is to use them in a drag and drop manner. Authors also present their ideas of off-the-shelf connectors and middleware technologies in a different study [47]. They present their work in C2-Style architectures.

Xillio [48] defines 'off-the-shelf' connectors for content migration from different repositories. He also provides a middleware solution for content integration. The system also allows the creation of new connectors through APIs (Application Pro-

gramming Interface).

Perez et al. [49] propose a method called PRISMA that uses COTS in aspect-oriented architectural models that are produced by Model-Driven Development (MDD) approach. They consider connectors as architectural elements that are responsible for coordination among components. They claim that their connectors are reusable because they do not contain any reference to interacting components.

Spalazzese and Inverardi [8] describe the term 'mediating connector' for the interoperability of heterogeneous components. They also define 'mediator patterns' targeting the basic mismatches that probably occur during component interaction. In another work, Inverardi et al. describes a synthesis of application-layer connectors [50]. In a more recent study, the similar connector synthesis approach is used to address functional and non-functional interoperability of networked systems [51].

There are also studies that focus on connector variability. Cetina et al. [52] offer the Model-Based Reconfiguration Engine (MoRE) that focuses on adaptation to changes at runtime in the context of autonomic computing. Dynamic reconfiguration of the system elements is done by activation/deactivation of features on a feature model. The approach hides how variability in the feature model is applied to the connectors. This may make the management difficult for large-scale systems.

FX-MAN [53] extends the X-MAN component model [54] with feature models. FX-MAN uses a logical architecture of the system as a tree of interacting components along with the feature model. By using variation operators and family connectors, product families can be constructed. However, the connector variability logic is hidden in the logical architecture which derives variations for connectors.

Finally, a literature survey conducted to outline the status of variability in component models has demonstrated the weakness of such capability [55]. Moreover, the current state seems to exclude the connector emphasis.

The connector related research is improving. Our approach leverages on its outcome for exploitation towards 'build by integration' that eliminates code-writing. Majority of the work on connectors target at least partially, development of new connectors.

2.4 Methodology

Off-the-shelf connectors for the component-oriented approach are presented in this section. Then, the proposed software development approach is elaborated.

2.4.1 Off-the-shelf Connectors

Complying connectors, with the ambitious reusability objectives, can be an alternative to writing code for integration. They are also instrumental in completing the spectrum of drag-and-drop activities for the composition of software. Figure 2.2 depicts a screen prototype for a tool where connectors can be selected and inserted in the design, as well as components.

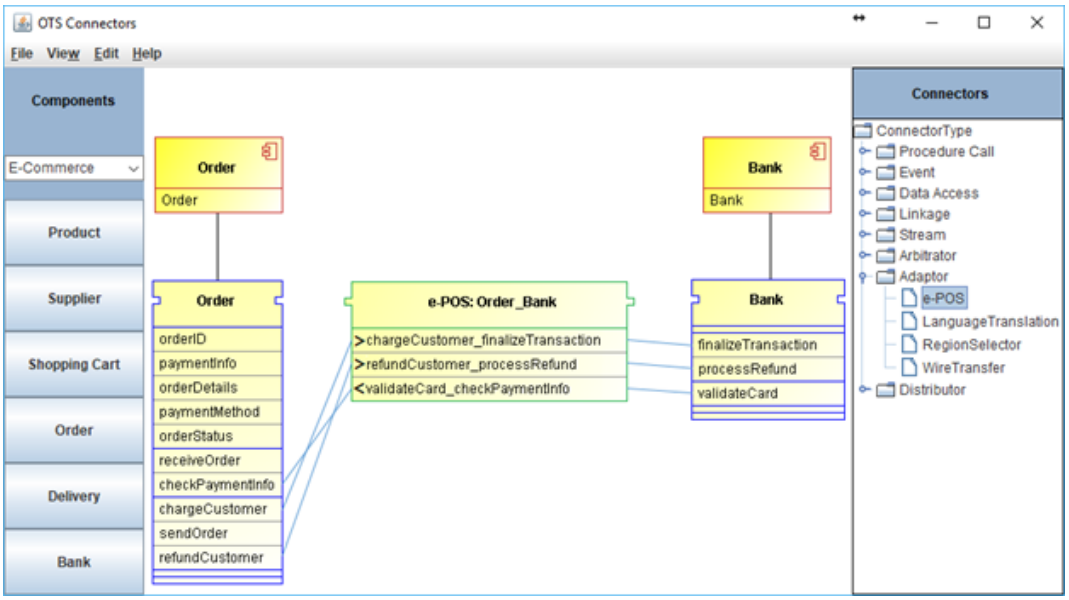

Figure 2.2: Tool bars for connectors and components for composition.

'Connectors' toolbar on the right-hand side of Figure 2.2 includes all of the connectors which can be used in a specific domain. Also, these connectors are grouped under eight classes to be identified more effectively based on the roles that they can possibly take in the system. Some pre-built and ready to use adaptor connectors are given for the 'e-Commerce' domain. The 'e-POS' connector is responsible for the handling of credit card transactions between two parties. Similarly, the 'WireTransfer' connector

can be used to manage direct money transfers among parties such as electronic funds transfer (EFT). The 'RegionSelector' connector allows a business to gain the ability to differentiate their services based on different regions. The 'LanguageTranslation' connector can be used to manage translations for previously defined conditions or events occurring at runtime.

Connectors readily include some methods to carry out their duties. Also, connectors have the ability to be configured in case of a required customization. A prototype screen for the interface that can be used to configure a connector is given in Figure 2.3.

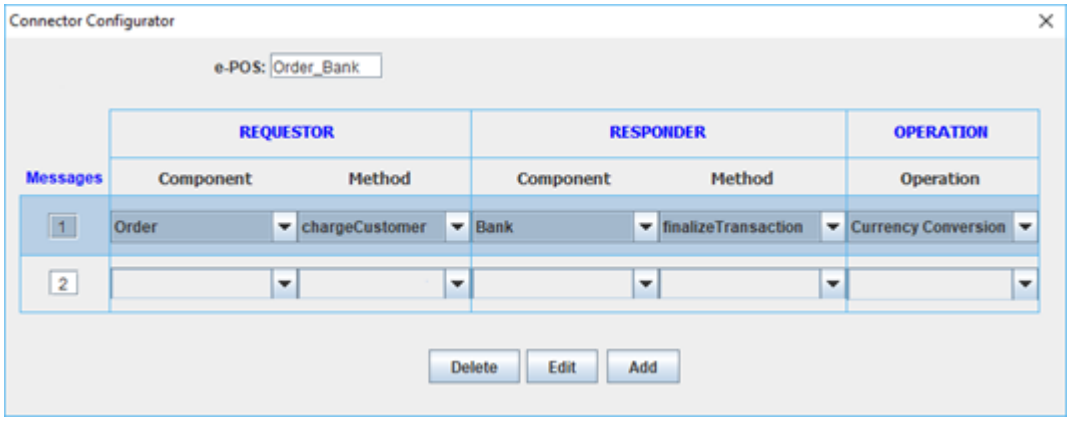

Figure 2.3: Configuration of a connector in the composition.

2.4.2 The Proposed Approach

Our aim in this research is to minimize, if not completely eliminate code writing. For satisfying this objective, three domains in the modelling of any executable medium can be identified: the overall coordination expressed as a process model, a set of components, and connection mechanisms. The motivation behind such an architectural style owes itself to the observation about the success in the Service Oriented Architecture (SOA) field. There, a de-facto two-level architecture is proposed that is supported with effective tools, protocols, and technologies. This architecture includes a global process model at the top level and a set of operations at the bottom level that implement the 'activities' in the process model. Figure 2.4 presents a simplified example for money withdrawal scenario represented in this architecture. Al-

though many developers do not exploit this infrastructure and they combine many techniques in the production of their code, this infrastructure offers a very easy to use and a settled method to developing distributed systems. A central control structure that can be developed in the graphical tools is the process model that would order all the invocations in the system comprising the first level in the architecture. The outcome is an 'executable' process model. Next, a collection of web services that are connected to the activity nodes in the process model provide the necessary operations. Consequently, existing technologies support the two domains (process model and components) satisfactorily. The last one of the three domains corresponds to connections that serve two purposes: 1) adaptation among components; and 2) lower-level coordination duties. Main coordination can be thought of being handled by the central process model. This research introduces connector structures that abide by this objective and offer the finalizing of software development that avoids code writing.

The model presented in Figure 2.4 corresponds to an executable system. To describe its execution in a consistent form, a simplified C language syntax is used for the code provided in Listing 1 where the process model corresponds to the main program and methods of the components correspond to functions. The main program corresponds to the higher level in the two-level decomposition and the functions correspond to the lower level. The bodies of the functions are not listed for brevity.

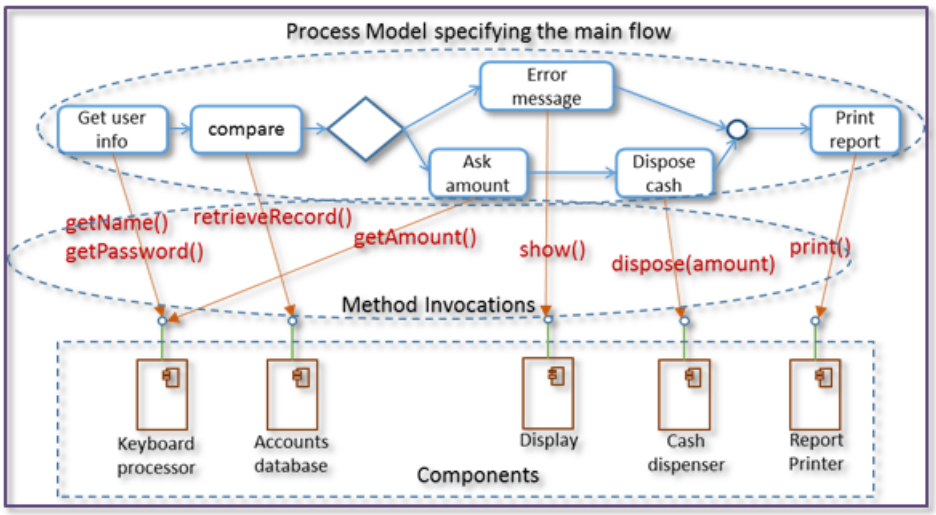

Figure 2.4: Money withdraw scenario in the SOA-inspired architecture.

A brief description of the methodology for utilizing connectors in the efficient composition approach within a two-level decomposition architecture has been provided in [22]. Here we are articulating more on the aspects that concern the plug-and-play nature of the connectors. The whole process depends on the persuasion that no code will be written but drag-and-drop operations will be supported with slot-based parameter instantiations to yield an executable system. That persuasion is materialized with the trust on configurable connectors that will fill in for the gaps and incompatibilities among a central process and a set of components.

Table 2.1: Modeling the executable system in a simplified C language representation.

```
// Level 2: A set of functions
getName()    .....  ;
getPassword()    .....  ;
RetrieveRecord(AccountID)    .....  ;
getAmount()    .....  ;
dispose(amount)    .....  ;
printReport(report)    .....  ;

// Level 1:  Ordered invocation of the functions
int main()
getName();    getPassword();  // sequential order
retrieveRecord();          // sequential order
if (verified)                  // conditional order
     amount = getAmount();     // seq. inside cond. order
     dispose(amount); ;     // seq. inside cond. order
else                 // conditional order
     errorMessage();
print();
 ;
```

Figure 2.5 depicts the process model for the suggested software development approach. The process starts with the construction of a central control model. An

executable process modelling tool will be instrumental in this step. There are conventional executable engines for the Business Process Execution Language (BPEL) that is graphical. Also, a more powerful modelling option is offered by the Business Process Modelling Notation (BPMN) that is also supported by interpretive execution, however not being as common.

The two-level hierarchy accommodates a single process model that is at the focus of the execution: providing a central control analogous to a state machine that processes the transitions of the whole system in a global, hence central state space. This corresponds to a central control that excludes parallel or multithreaded execution modes. However, this model can easily be adapted to its parallel versions by allocating more than one processes that communicate. The BPMN or BPEL technologies are currently capable of supporting such infrastructure. The discussed parallelism actually paves the way for the implementation of distributed algorithms which are more powerful than the central control model.

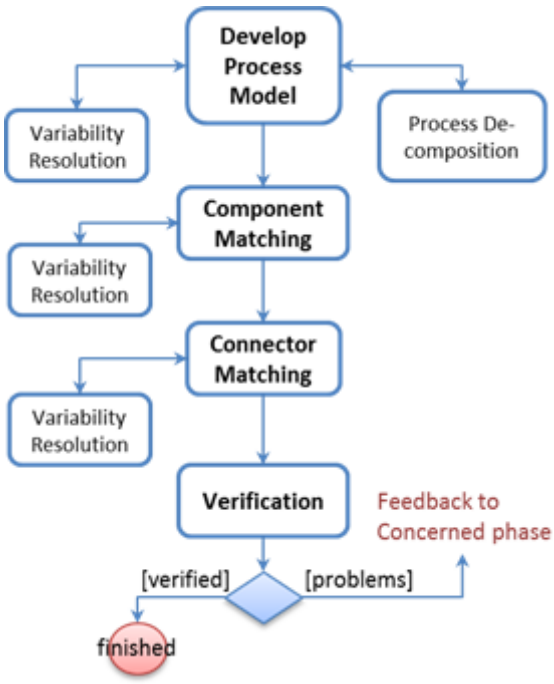

Figure 2.5: Connector supported two-level architecture-based composition.

Distributed or central, the main process is the important specification of the executable system, due to our top-down persuasion for development that supports the

convergence of the design decisions better, to the final product. Top-down approaches allow the holistic view to be kept within consideration throughout the development. Otherwise, starting with the components and integrating the application by partial ordering (partial process models) and combining such models to the global process model(s) is also possible.

The activity boxes in a process model can be linked to the methods of existing components for executing the duties specified in the box. It is possible to execute parts of the process model for prototyping purposes. It is not necessary therefore to complete the whole process model before exercising with it.

Supporting the process model development with process decomposition, and especially offering a methodologic decomposition (hence, top-down definition) is also studied in [56]. There, a little definition in any level of the architecture is confirmed with the corresponding definition in other levels of the architecture. Here, definition means a decomposition step because any new module appears as a result of decomposition. Actually, the referred work suggests a simultaneous definition (decomposition) of the process model and the other models such as the set of components. This is like considering one level at a time as in a breadth-first strategy. In either case, completing the whole process first or only a small part of it before components are connected, we suggest a top-down approach. The process model comes first as it is the centre of execution.

Process can also be shaped through instantiation from a domain model. A generic process model could exist complementing a domain model such as in SPL approaches. SPL approaches also propagate the variability management concept within their practices and to some extent, export the idea to other approaches. Here, variability capability will result in the configuration of a more generic process model to a specific one, tailored for the current application. An example to instantiating a generic process model can be given considering a multi-way branch that corresponds to a selection. In an executable process model for an application, the branches all exist and at run time, based on data values or decisions, one of the branches will be taken. However, during the instantiation of a domain process model, the alternative paths that are marked as variants for a variation point will be erased and excluded in the final product.

An important expectation is the establishment of domain-specific development infrastructures, with developers who are familiar with their contents. Such mature domains, that may currently be existing for the internal use of SPL incorporating organizations, should be more commonplace. In that case, a match between the process model and the components will be more successful. A better set of components to serve the whole domain with known uses and descriptions provides leverage. Also, scenarios that are experimented earlier to offer ordered set of activities to solve sub-problems (as partial processes) will aid in more effective development of process models. Activities defined at the process level, will match existing components with probably a higher success rate. A similar pattern has been experienced in service orientation where existing web-services that correspond to business functionalities have proven desirable.

Development will continue with matching a component, actually a method in it, to an activity in the process model. The component may go through configuration before this connection. Variability specifications in the domain model will provide the desired guidance, preferably with totally automated configurations.

Configuring a component is not sufficient for its execution-level match with the system. Connectors are required to connect their interface slots to correct counterparts. In the simpler case of connecting the central process to one component, one would assume the process playing the client role whereas the component playing the server role. For a specific method invocation, the required interface in the process model, specifying a function it needs, should be connected to the published interface in the component for the specific service. Usually, we cannot tamper with a component except for some small modification. It is possible that changing its method names may not be allowed. Luckily the central process is ours, and it would include slots to identify the component and its method so that it can send a message to the desired target at run-time. In such cases of perfect matches, no need appears for a connector. However, connectors do more than just name binding. Also, in the case of component-to-component connections the existing method call from the requestor (client) with a pre-defined function name, may require a name translation. Our connectors have slots that record the expected method definitions for both the requesting and the responding party's existing method names.

2.5 Case Study: e-Commerce System

In order to demonstrate the proposed approach and its abilities, an e-commerce system is designed in COSEML. The system consists of 'Item', 'Supplier', 'Shopping-Cart', 'Order', 'Delivery' and 'Bank' components. Structural decomposition of the system is given in Figure 2.6. All of the interactions among these components will be handled through connectors. It should be recalled that the higher-level units correspond to logical level declarations.

A connector is used to relay messages. Therefore, messages are declared in connectors. We followed a convention in such name assignments that would indicate the name of both the requester and the responder parties. There could be more than one message contained in a connector. When a requestor interface performs a request through its 'Method-out' slot from a related connector: the connector triggers a connector message which starts with the received 'Method-out' call. The activated connector message, in turn, activates the 'Method-in' of the responder interface. The direction of the interaction is determined by a symbol attached before the connector name. The '<' symbol shows that the direction of the flow is from right to left, so the interface given on the left would be the responder interface and the interface given on the right would be the requester interface. The '>' symbol is used to represent the reverse direction.

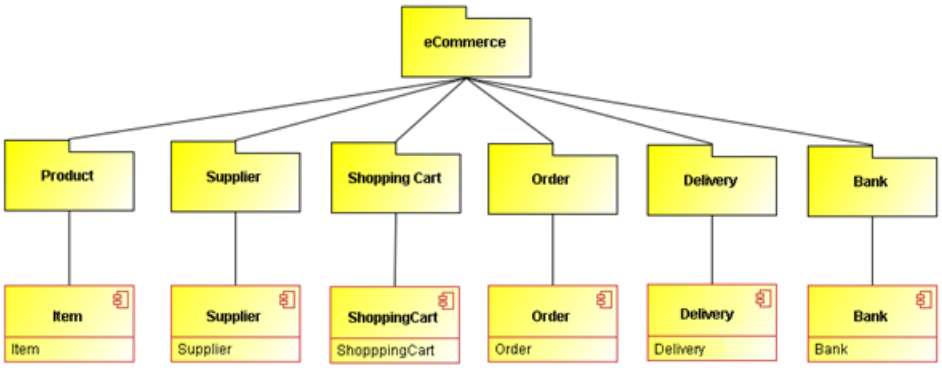

Figure 2.6: Structural decomposition of the e-commerce system.

The connector with the name OrderCreator:ShoppingCart_Order handles the inter-

actions between the ShoppingCart and the Order components is shown in Figure 2.7 where the components are not shown to save space: only the connection related units (their interfaces and a connector) are included. In this figure, it is shown that the orderItems_receiverOrder connector message is triggered through the orderItems methodout of the ShoppingCart (requester) component. When the connector message is triggered, it calls the receiveOrder method-in of the Order (responder) component.

Figure 2.7: Interactions between the interfaces of the ShoppingCart and the Order components through the OrderCreator:ShoppingCart_Order connector.

In the e-commerce system, the communication among the 'Order', the 'Delivery' and the 'Bank' components are handled through the 'ShipmentManager:Order_Delivery' and the 'e-POS:Order_Bank' connectors as it can be seen in Figure 2.8. When the Order component receives an order through its 'reveiveOrder' method, it interacts with the Bank component through the 'e-POS:Order_Bank' connector to charge the customer with the 'chargeCustomer' method. However, this money transfer interaction may require currency conversion that could consider a variety of currencies. It is possible to have different currency expectations on both sides of this interaction. Such differences can add up to structures where mapping from one big list to another will be implemented. This choice for configuration of the adaptation is solved based on variability modeling, which will be explained soon below. When the transaction is completed, the 'Order' component interacts with the 'Delivery' component through the 'ShipmentManager:Order_Delivery' connector with its 'shipmentRequest' method. This operation completes the purchase. Additionally, the Bank

component has the ability to validate the payment method for security reasons if it finds necessary. This validation related interaction would also be performed through the ‘e-POS:Order_Bank’ connector. Also, the Order component has the functionality to refund the customer if a customer returns the order with its ‘refundCustomer’ method.

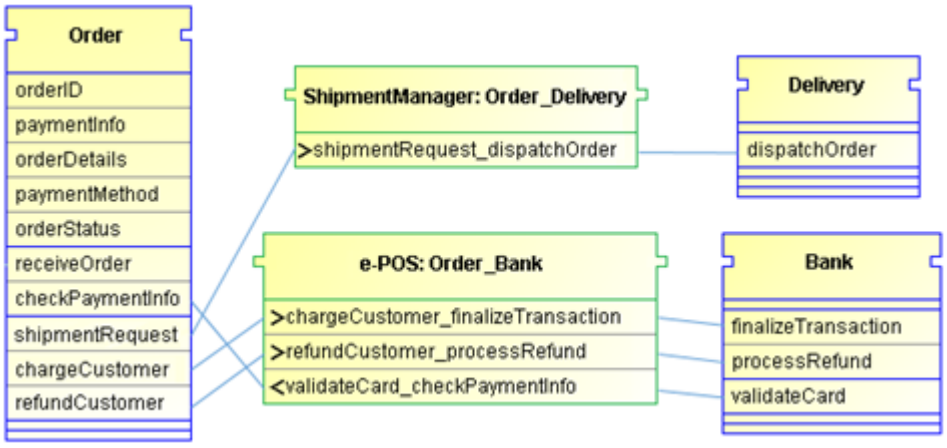

Figure 2.8: Interactions between the interfaces of Order-Delivery and Order-Bank Components through connectors.

The interaction involving the ‘charge customer’ request and the ‘finalize transaction’ service is exploited in this example to illustrate the variability related development steps. The ‘Order’ and the ‘Bank’ components are assumed to be manufactured for a fixed money currency. This is a typical simple example where some adaptation is required. Since connectors are the chosen media for adaptation, the ‘e-POS:Order_Bank’ connector will assume such responsibility.

The adaptation corresponding to matching different currencies forms a good example for variability: the options for converting currency offer a wide choice area for different currencies. A short list is presented in this example.

Where to implement the variability is another issue. A connector housing only one message could be selected for the mentioned interaction. However, assuming a common usage for such a domain-specific connector, we suggested this ‘Order_Bank’ connector to include the three messages that are expected to be used in most of the order-to-bank connections. The message connecting the ‘charge customer’ request

to the ‘finalize transaction’ is where such a currency conversion is needed. That is why the mentioned variability is embedded in this method. Recalling that the connectors have methods declared once per message connection, this ‘charge customer’ to ‘finalize transaction’ message has its own operation. Any connector capability is potentially possible to be coded in this operation. Here, the operation will conduct the adaptation, that will be for currency conversion and that will be configurable. The configuration is how the variability is resolved. Also, coding a capability is a domain engineering activity assumed completed before.

Figure 2.9 illustrates a screen design corresponding to the configuration of the ‘Order_Bank’ connector. Once an instance of this connector (e-POS) is selected for modification such as through a double click or a right click on it, the developer should be able to pick a method for its configuration. In this case, the first method ‘chargeCustomer_finalizeTransaction’ is selected and variability management offers the available conversion options at the design-time.

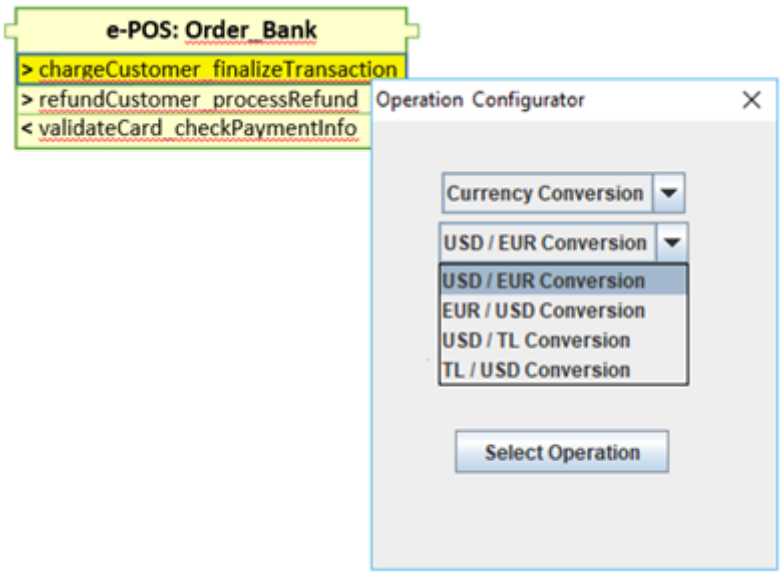

Figure 2.9: Resolving the variability on currency conversion through connector configuration.

Actually, the configuration was conducted due to the variability specifications that is part of earlier specification activities than development. A feature model [57] could be developed for the domain, that is used for specialization into separate feature mod-

els for each product. Such specialization is conducted through resolving the variability. Although it is possible to model variability in a feature model, it is preferred to complement a feature model with a separate variability model. Such a separation is necessary for huge models. Excerpts from a feature model and a variability model are represented in Figure 2.10. The variability modeling representation complies with the Orthogonal Variability Model [58] approach.

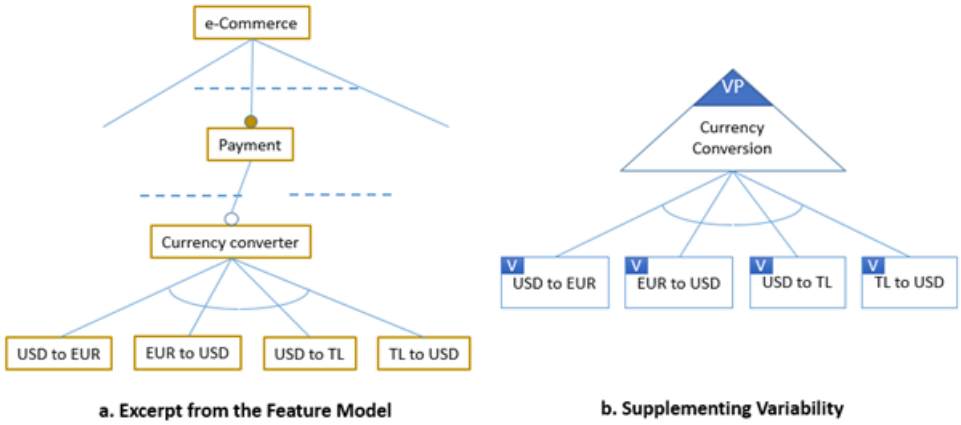

Figure 2.10: Excerpts from domain feature model and variability model.

2.6 Discussion

New connector structures are proposed to support development by integration, to be used with component technologies. Although these connectors are flexible to be used in any kind of composition frameworks, also a central process-driven architecture is promoted. Inspired by the success in the utilization of SOA based technologies, this approach suggests a two-level architecture where the ordering of operations is specified in a process model and the activities in this model are conducted by existing components or web services. This is analogous to a main program that uses the control structures (such as if, switch and loops), to execute the flow which includes only function calls inside these structures. Whereas the algorithmic computations are dispatched to a set of functions that are called by the main program. This view suggests the modelling of a system in those two levels, ignoring nested function call mechanisms and hierarchical organization of the process model. The 'ignoring' keyword is important here, meaning they are not categorically ruled out. However, the

main model should emphasize a two-level hierarchy. As a result, the vast experience gathered in SOA can be leveraged. The process model can be developed by experienced developers who know the component library and also similar solutions devised before. Supported with the existence of many components/web-services at different granularities and preferably handling business requirements directly, such two-level decomposition would yield a very efficient development process.

Exceptions will always be with us: more levels in the decomposition can be introduced where necessary. That may be due to a new problem that has not been coded before requiring the construction of subassemblies that are not available in the form of directly available components. Anyway, flexibility is usually desired. However, simpler architectures as recommended here will prove more effective in the production of software intensive systems. This will be possible as the development environments mature rendering the saturation of the domains with components.

Connectors seem to be assuming partially the responsibilities of a middleware. This is not one of the objectives of this study. For practical purposes, even connectors may need to be implemented through component technologies (as components) and they would also consult to a middleware for connectivity. The idea is to provide a complete set of assets for development through configuration and integration. Any systematic tool or support is welcome. Anyway, there is a potential for connectors to mimic a middleware partially and hence provide some independence from them.

The proposed connector approach is an opportunity for the interaction of components from various disciplines. In the e-Commerce case study producers, suppliers, delivery departments and banks are meant to represent different domains. Incorporated processes are required to be handled by capabilities residing in more than one discipline. The communication with the third-parties that are out of discipline, can be handled with the help of connectors. Reusing existing connectors and customizing them for different applications reduce the cost of system development.

2.7 Conclusions

An approach for effectively incorporating connectors in system development is presented. Domain specific environments are assumed for a default functionality served by a set of present connector types. Those functionalities are not usually ready for use, allowing room for finalization. Through variability modelling those expected capabilities for a domain are offered in our 'domain model'.

Product specific integration effort is supported by partially expected capabilities that shape into the domain models in terms of variability. Final tailoring is conducted through variability resolution. Although variability is supported to be specified at other times, mostly design time configuration is incorporated.

Our experimentation with example designs has demonstrated the usability of the approach. Validation for the approach, however, suffers some serious drawbacks: A real world development environment with its commercially developed requirements and matured software assets and tools is not easy to acquire and then, to use. Such items need to be artificially generated for use in proof-of-concept studies. Although the overall feasibility of the approach can be demonstrated through artificial environments, efficiency of validation may critically depend on the fidelity of the assets. Following an expectation for the near future, such frameworks should appear and, mature in time for more efficient development support.

CHAPTER 3

LIVE-VIRTUAL-CONSTRUCTIVE INTEROPERABILITY THROUGH THE DDS-HLA GATEWAY

Software systems need to be more complex and large-scale to keep up with growing user expectations with ever-increasing technological improvements. Building these systems from scratch is costly and time-consuming; thus, the importance of reuse and interoperability is increasing. Employing more than one subsystem to yield a more complex system in Live-Virtual-Constructive (LVC) simulation systems is frequently seen. These subsystems may have different implementations and designs. In such multi-architecture LVC environments, gateways are promising solutions to address interoperability issues. In this chapter, a gateway-based solution is proposed to achieve LVC interoperability with a particular focus on two standard middleware, viz., Data Distribution Service for Real-Time Systems (DDS) and High-Level Architecture (HLA) for distributed simulation. The gateway is capable of providing two-way data transfer between DDS and HLA. The design of the gateway adheres to the idea of configurable connectors, which allow users to generate a customized gateway. The gateway is capable of converting primitive and structured data-types between DDS and HLA. These conversions are specified by users resulting in different configurations of the gateway. The gateway can also be adapted to another configuration at runtime. The applicability of the gateway is shown in academic and industrial case studies.[1]

[1] The study described in this chapter is currently under submission for publication.

3.1 Introduction

As systems are getting more sophisticated and grow in size, the need for reusing existing software and combining heterogeneous subsystems into a bigger system becomes a necessity. This case is usual for LVC simulation systems, where different kinds of systems that have unique architectures are operated together. Thus, the interoperability of multi-architecture LVC simulation systems is an open problem.

LVC Architecture Roadmap (LVCAR) was initiated to examine major simulation architectures from different perspectives to increase interoperability in multi-architecture simulation systems. The main goal is to reduce development time and costs and improve quality [59]. The development of gateway and bridges to provide interoperability solutions for LVC simulation systems take place in the core efforts of the LVCAR study. Developing gateways specific to a system is an issue preventing reuse of it for another system [60]. Configuring a pre-built gateway for a new system would improve interoperability efforts for multi-architecture systems.

Combining real-time systems and simulation systems into a bigger one is a known solution. The interoperability of these two kinds of systems in LVC simulation and large-scale cyber-physical systems is crucial [61]. Using real-time systems and simulation systems in a combination decreases production costs, especially in the defense industry [62]. Thanks to their advantageous combination, DDS and HLA are frequently used together. Since their application areas differ, various solutions are developed for the interoperability of these two middleware standards. While some solutions have a single interface serving as a combination of both architectures [63], some distributed simulation systems use DDS as a communication infrastructure for its considerable quality of service (QoS) capabilities [64, 65].

The work presented in this chapter is efforts through supporting LVC interoperability. DDS-HLA gateway is developed to help using these two architectures in the same system. The initial need for DDS-HLA interoperability arises from an industrial project that is a DDS-based avionic system development platform, namely HAVESIS [66]. This DDS-based system is intended to cooperate with an HLA-based tactical simulation system. An architectural connector-based gateway is developed as a solution.

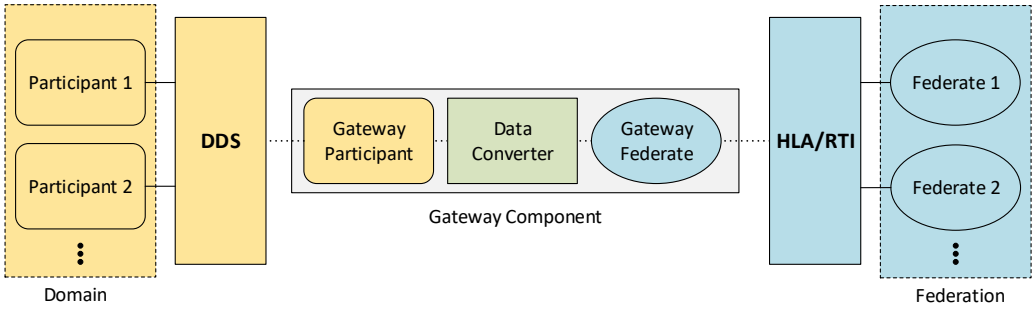

Figure 3.1: Overall structure of the DDS-HLA gateway.

In software architecture, connectors are the elements where communication concerns are handled. Accordingly, the proposed gateway handles the data transfer and conversion tasks between DDS-based and HLA-based systems without modifying their original structures. In our solution, we assume that the HLA-based system does not use time management. Alternatively, it is assumed that the simulation system uses physical time, as it is the case in the DDS-based system.

The gateway is conceptually composed of three main components: a *gateway participant*, a *gateway federate*, and a *data converter*, as shown in Figure 3.1. The gateway participant and the gateway federate establish the communication with DDS and HLA systems. The data converter contains a shared data area that has primitive and structured data type declarations, and functions to transfer and convert data between DDS and HLA. The data converter is the configurable part of the gateway to allow customization [3]. Thus, only necessary data types, data structures, and related functions are included in the gateway. The configuration of the gateway can be done both at compile time and runtime.

The whole work done on the DDS-HLA gateway can be summarized as follows:

- **Previous work:** The prototype of the gateway was developed and integrated into an industrial project [11]. Besides providing two-way communication between DDS and HLA, this version was also allowing the configuration of primitive data type conversions.

- **In this chapter;**

- The gateway is adapted to be configured dynamically.
- The gateway is enhanced to support the transfer and conversion of structured data types.
- More tests are conducted and use cases are provided to show the usability of the gateway with its updated features.

The tests, including an industrial case study, show the applicability of our gateway approach. Although other research provides DDS-HLA interoperability methods in the literature, our approach differs in terms of configurability. Runtime adaptability is also a plus since DDS and HLA environments themselves are usually prone to unexpected changes during execution.

The rest of the chapter includes required background information that covers brief information about DDS, HLA, and dynamic adaptation. The related work is discussed, including LVC interoperability, DDS-HLA interoperability in a comprehensive way, and interoperability of different middleware technologies. Then, the DDS-HLA Gateway is elaborated: its design, configuration mechanism, and operation details are presented. Afterward, case studies are provided that include an industrial application, other possible use case scenarios, and tests. The chapter is concluded after a discussion section that contains some remarks.

3.2 Background

The necessary background information is provided in this section. Brief information about DDS and HLA is given. Moreover, dynamic adaptability is mentioned.

3.2.1 Data Distribution Service

Object Management Group (OMG) recommended DDS [67] as a publish-subscribe based standard for data sharing of large-scale systems to provide anonymous and decoupled communication. The communication can be asynchronous and also real-time. DDS has QoS policies for non-functional parameters, including data availability, data delivery, and data timeliness.

The distributed environment provided by DDS is through Global Data Space (GDS). Publishers and subscribers can join and leave GDS at any time of the execution. They are discovered dynamically by GDS. Because GDS has a fully distributed nature, there is no single point of failure. As a powerful tool with the features mentioned above, DDS is used for reliable and efficient data transmission and sharing in various areas of application such as the Internet of Things (IoT) finance, smart cities, air traffic control [68].

3.2.2 High-Level Architecture

HLA is a distributed simulation architecture framework composed of independent, potentially reusable, loosely coupled components that allow a complex simulation system to be decomposed [69]. The main goal is the reuse of simulation components. Therefore, the coupling of components should be decreased, especially in terms of communication dependencies. Runtime Infrastructure (RTI) is the mediator that provides services for simulation components, as a middleware.

A federate is a simulation application conforming to the HLA standards. A federation is a simulation environment that is composed of federates. Federates communicate to RTI using the standard services and interfaces to participate in the distributed simulation and exchange data [70]. HLA is widely used in the defense industry. Besides, it has many applications in civilian life, including space, aeronautics, air traffic management, production, disaster recovery, and transportation systems.

3.2.3 Dynamic Adaptation

Dynamically adaptable systems allow changes in their structure or behavior at runtime without stopping the whole system [71]. Changing conditions or requirements necessitate system reconfiguration. Then, reconfiguration is performed to keep the system functional, or provide a better functionality under the new circumstances [72]. These changes may also target non-functional parameters to improve them. Dynamically adaptable systems are in many fields, including IoT, ambient intelligence (AmI), robotics, machine learning, as well as distributed simulation systems [73, 74].

Dynamically adaptable systems are usually self-adaptive. These systems configure themselves without human interaction by monitoring changes in their operating environment [75].

One of our previous works is about the runtime adaptability of AmI systems. We describe how a component-oriented AmI system reacts to changing conditions at runtime in [76]. Runtime adaptability is performed by a runtime configurator based on an ontology-based mechanism.

3.3 Related Work

Related work is categorized into three titles: LVC interoperability, DDS-HLA interoperability, and the interoperability of other multi-architecture systems.

3.3.1 LVC Interoperability

LVCAR initiative resulted in notable findings for interoperability. In LVCAR implementation, bridges and gateways are investigated to offer the LVC user community better alternatives to discover, select, and configure [77].

In the LVCAR final report [59], architectural boundaries of interoperability are defined, and recommendations are put forward. Moreover, Coolahan and Allen present the results they obtain after applying recommendations of LVCAR [60]. They show remarkable findings of the development of multi-architecture simulation systems.

In [77] and [78], authors present LVCAR enhancements for selecting and using gateways. Gateway selection technologies and its process to select a convenient gateway are handled in [77]. Gateway Mapping Language (GML) is introduced in [78] as a formal language to specify required translations in a multi-architecture system. The language provides the format to document the required translations for stakeholders, and it helps to ensure agreement on them. Another formal language, Gateway Configuration Language (GCL), is also introduced in the same work. GCL is used to specify common gateway configuration parameters and some additional features. By using GCL, gateway parameters can be documented independently from the gateway

implementation. The documented information with the help of GML and GCL can be used in the future for other gateway implementations.

As Hodson and Hill also declare, the need for interoperability is caused by the demand for reuse [79]. Developing simulation systems from scratch is costly, especially if they are complex. Also, again in [79], authors explain the differences in the definitions of the terms gateway and bridge. Bridges are used to provide interoperability for different versions of the same architecture. However, gateways are utilized for connecting different architectures.

Although they are not targeting the defense industry directly and categorizing their work as a solution for an LVC training system, Ardila et al. [80] present an architecture that allows interoperability for joint real and virtual training in emergency management. They extend MPEG-V standard and develop a middleware, namely interconnection gateway, to provide interoperability for various applications.

3.3.2 DDS-HLA Interoperability

DDS-HLA interoperability approaches in the literature differ in terms of the methods used [81]. Some approaches use a combination of both architectures as infrastructures, aiming for a merge method abstracted from the characteristics of both architectures. Another common use is to use DDS as a communication infrastructure for HLA. In addition, there are models that provide interoperability with a gateway. In the gateway approaches, interoperability is achieved without interfering with the structure of the DDS-based and the HLA-based system. Consequently, DDS-HLA interoperability approaches can be categorized into three groups: the fusion model, the transport layer replacement model, and the gateway model.

3.3.2.1 Fusion Model

In the fusion model, an abstracted access model from both interfaces is obtained with an additional layer built on top of the DDS and HLA interfaces. The main purpose of this model is to isolate the interfaces of DDS and HLA compatible systems

from interoperability problems by combining them with minimum effort in the later stages. When this model is preferred for the integration of existing systems, it requires changes in the interface implementations.

An example application of this approach is NCWare [82], which is a software abstraction layer that provides interoperability for DDS and HLA by combining their standards. As a real-time networking middleware, it unifies DDS and HLA standards through a single API (Application Programming Interface).

3.3.2.2 Transport Layer Replacement Model

In the transport layer replacement model, DDS is employed in the transport layer, and HLA is used in an upper layer [64, 83]. Generally, DDS is the wire protocol of an HLA-based system. Thus, the broad range of QoS capabilities of DDS can be used while preserving HLA's inter-simulation high-level architectural properties. The resulting interface in the final system is HLA-compliant, and DDS acts as a communication infrastructure. In this regard, this solution can be categorized as a solution to increase the capabilities of inter-simulation middleware and a spectrum extension rather than strictly an interoperability solution.

HLA-DDS wrapper proposed by Park and Min [83] is an example of the transport layer replacement approach. In this method, the transport layer of an HLA-based distributed simulation system is replaced with DDS, and APIs of HLA and DDS are combined. The HLA-DDS wrapper is compatible with the HLA standard interface. Thus, the proposed solution supports network-controlled distributed simulation systems and allows preserving of existing HLA-based distributed simulation systems.

3.3.2.3 Gateway Model

The gateway model ensures the resulting system to exhibit the characteristics of both DDS and HLA, unlike the fusion and transport layer replacement approaches. In this approach, the final system should transfer the functionality directed from DDS to HLA as an HLA compliant federate, and from HLA to DDS as a DDS compliant

participant. Oh et al. [62], achieves DDS-HLA interoperability by an ontology-supported gateway solution. Their approach can serve for different systems that are based on DDS, HLA, and also DIS (Distributed Interactive Simulation).

Park and Min introduce HLA-DDS bridging component in [84] that can be categorized as a counterpart for a gateway. The bridging component uses DDS-based physical systems on HLA-based distributed simulations. It is located between a DDS participant, and HLA federates of the system and provides two-way communication. The component presents itself to the HLA-based system as a federate, and to the DDS-based system as a participant. The bridging component consists of three parts: an inner HLA federate, a data mapping object, and an inner DDS participant. Inner-federate and inner-participant receive data from the HLA federation and the DDS domain, respectively, and send data to the other side through the data mapping part.

Our gateway approach has a similar architectural design as the HLA-DDS bridging component, and they both aim to achieve DDS-HLA interoperability. The proposed solution in this study allows dynamic configuration capability through the deployment of different configurations at various execution stages, including runtime. Moreover, our work contains initial efforts for guiding the development of the gateway through variability models, as explained in Section 3.6.

3.3.3 Interoperability of Other Multi-Architecture Systems

This section includes similar interoperability approaches in other multi-architecture systems, not necessarily in LVC or between DDS and HLA. In [85], authors investigate solutions to combine DDS and ARINC-653 standards. A combination of these standards would be a remedy to interoperability needs in mission-critical and safety-critical partitioned systems, such as avionics. In this sense, different integration architectures are offered. Analysis of the integration of DDS and ARINC-653 is also provided in the same study. In another DDS-based approach [86], authors present their framework that provides interoperability solutions for real-time heterogeneous participants to interact dynamically in dynamic distributed architectures.

Middleware technologies are helping to integrate various heterogeneous components,

such as sensors and actuators, in robotics. However, using only one middleware technology may not be sufficient to provide all the required functionalities in an application. Therefore, many applications need more than one framework. Authors in [87] suggest a solution for interoperability between two standard robotics middleware. Their approach is based on the idea of using the existing code for an application while avoiding writing code for the cooperation of the two middlewares. Users provide specifications for the communication needs between the middlewares as a configuration file, then the necessary code for bridging is generated; this is a similar method to our gateway configuration approach.

3.4 DDS-HLA Gateway

This section tackles the DDS-HLA gateway in detail: design decisions of the gateway, the configuration ability, and execution details are provided.

3.4.1 Design Decisions for the Gateway

Architectural connectors inspire the design of the DDS-HLA gateway. Instead of being placed between two components, it links up two different systems.The gateway aims to establish a two-way connection between the distributed applications running on two different middleware. The application concept corresponds to the domain application on the DDS side and the federation on the HLA side. Simulation components that take part in the federation are called federates, and the members of the domain application are known as participants. As the same as the domain application, it is assumed that the federation is running in real-time. Therefore, the time management services offered by the RTI are not used.

The gateway is composed of three conceptual parts: Gateway Participant, Data Converter, and Gateway Federate. Figure 3.1 illustrates the general design of the gateway. The gateway is implemented in a multi-threaded manner where these three abstract parts are scattered into threads. In the current implementation, OpenSplice is used as a DDS middleware software; Portico [88] or OpenRTI [89] is preferred as an HLA/RTI middleware implementation. Qt [90] libraries are used to realize the multi-threaded

design. The implementation language is C++.

The gateway component, in Figure 3.1, appears as a participant to the DDS side (through "Gateway Participant"), and as a federate to the HLA/RTI side (through "Gateway Federate"). Therefore, the gateway must join the domain as a DDS participant, and the federation as a federate, and communication is conducted through Gateway Participant and Gateway Federate. In Data Converter, the shared data area contains the data type and structure definitions and declarations in which the communication data is recorded, and the functions used to perform read/write operations and type conversions on the data.

In the gateway design, two-way traffic that may occur during data transmission is taken into account. Messages that are sent from the DDS domain and the HLA federation may not be in a specific number and order. For this reason, the gateway should be listening to both sides continuously and be able to transmit data in both directions. Thus, a multi-threaded design is preferred for the gateway.

The detailed design of the gateway is depicted in Figure 3.2. The multi-threaded model is composed of four threads where two of them for Gateway Participant (1, 4), and two of them for Gateway Federate (2, 3):

(1) Inner DDS Subscriber: Receives messages from the DDS domain.

(2) Inner HLA Publisher: Sends messages to the HLA federation.

(3) Inner HLA Subscriber: Receives messages from the HLA federation.

(4) Inner DDS Publisher: Sends messages to the DDS domain.

These four sub-components of the gateway are allowed to access to the shared data area. When subscribers receive data from the system that they are listening to, they write this data to the corresponding locations in the shared data area and notify the publisher-side of their type. Publishers read the data from the shared data area and send it to the system that they are registered.

In the gateway design, all data types and structures are defined in a bi-directional manner to prevent congestion. This mechanism applies to all primitive and structured

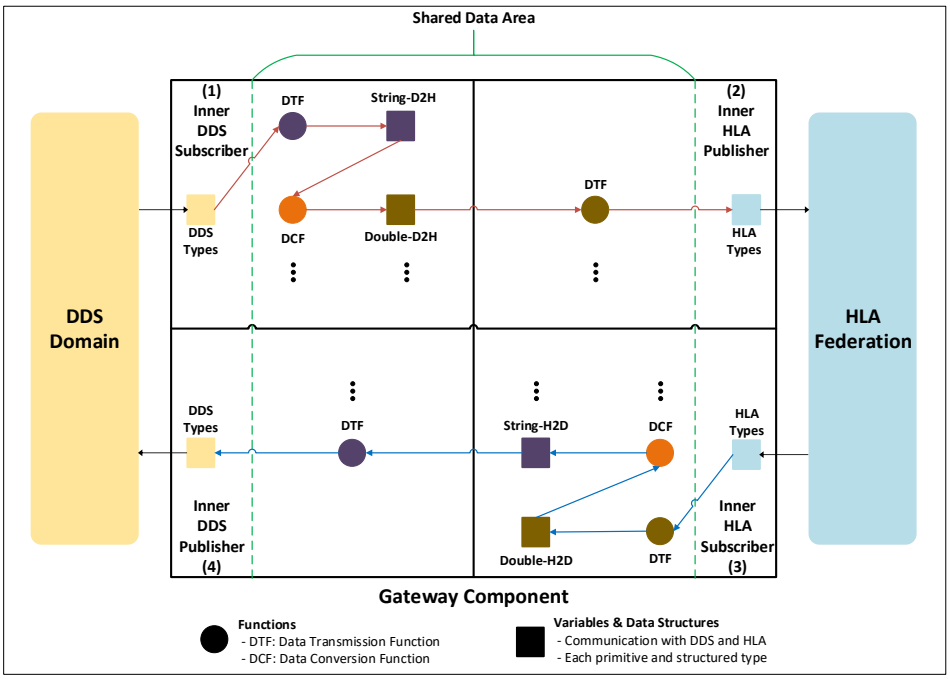

Figure 3.2: Detailed design of the DDS-HLA gateway.

(such as arrays, lists) types of data transmitted between the DDS domain and the HLA federation. Therefore, there is no waiting to write the incoming data to the shared data area while transmitting data at the same time. The data transmission functions conduct data transfer in the shared data area. These functions are used to read and write data from the DDS and HLA compatible variables and structures. Moreover, they are utilized in inter-thread communication.When type conversion is needed before transferring the data, data conversion functions are used. These functions accept the source type as an argument and return the target type after the conversion.

3.4.2 Configuration of the Gateway Component

Data types of DDS and HLA may differ even if they are of the same type essentially. For example, one side of the communication may use short integers while the other side uses long integers. Moreover, users may deliberately want to transfer data to the other side in a different format. For this reason, the gateway component is equipped with configurability. The configuration takes place in accordance with the data map-

pings defined by the user (or the target system). According to this configuration, the source code of the gateway is generated and compiled before the deployment. The deployment can occur at the development time or runtime.

The gateway component has a skeleton code structure. The code blocks for data type and structure definitions, and related functions for data transmission and conversion are automatically generated and added to this fixed structure according to the user-defined mappings. Users can map data types of the DDS-based system and the HLA-based system to each other through a textual or graphical UI. A tool is developed for the code generation as depicted in Figure 3.3. The tool uses a simple graphical UI for data type or structure mapping. After acquiring the mappings from the user, the gateway code is generated. For each of the user mappings, required data definitions and functions are injected into the skeleton code structure.

Data type conversions occur in the shared data area of the gateway (conceptually, the Data Converter, as shown in Figure 3.1). Conversion is needed during data flow, such as when transferring incoming data in a DDS topic to an HLA object instance. Furthermore, there may also be user-defined conversions. The required code blocks for data transmission and conversion are generated based on the types of source and destination formats. There is a standard naming convention for the generated variables and functions. The names are read from the IDL (Interface Description Language) file of DDS and FOM (Federation Object Model) file of HLA and used in the code generations adding postfixes to them indicating the direction of the communication (such as *dds_to_hla*).

The gateway component must be included in the DDS topic space as a DDS participant to be DDS compatible. As the scenario-specific topics may need to be published and listened, knowing or anticipating DDS topics in advance in the case of a potential data transformation scenario is not possible. Therefore, IDL is used to create scenario-specific topics for the gateway. After that, the Gateway Participant of the gateway component needs to be created by compiling the IDL source. The user provides data types in the IDL file, and rules regarding how to convert data in the configuration files. DDS-related gateway parts are created by compiling along with the topics that contain data types. The same processes also are also applied for the

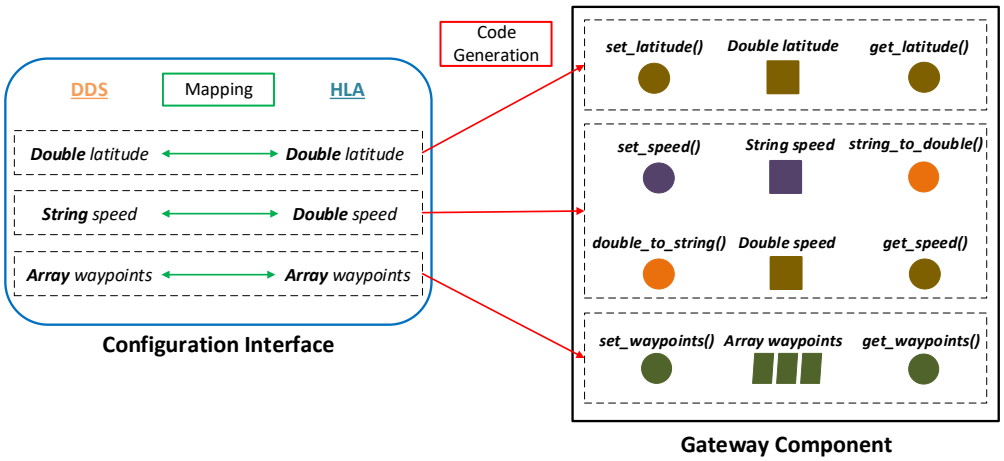

Figure 3.3: Gateway configuration.

HLA-related parts of the gateway.

3.4.3 Operating the Gateway

The user initiates the gateway and it is triggered to join the DDS domain by using its gateway participant. The gateway participant calls publishing and subscribing routines for topics related to the data sets defined at the compilation stage. Upon completion of this initialization, a similar process is initiated for the gateway federate. The DDS domain and the HLA federation need to be created before the gateway deployment. In other words, the gateway is not responsible for creating the domain and the federation since its concern is establishing communication for existing systems.

The gateway performs two-way data conversion (from DDS to HLA and HLA to DDS) during the execution after the deployment. The gateway participant receives data updates from the DDS domain, and transformation routines are executed so that the topic data can be interpreted by the gateway federate. Then, the gateway federate publishes the converted data on HLA/RTI. Similarly, data updates from HLA federation are received by the gateway federate, converted, and passed to the gateway participant to be published to the DDS domain.

A real-life scenario or a test case may have stages that require transferring different

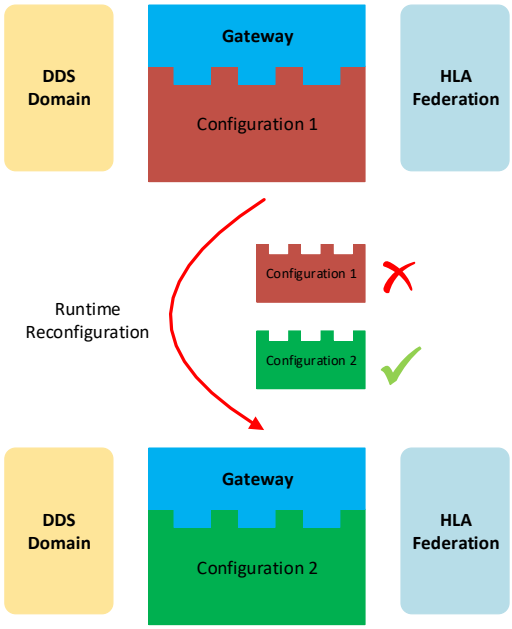

Figure 3.4: Runtime configuration.

types and amounts of data. When entering a new phase of the mission, the gateway should tolerate this transition. If these stages are known in advance, different configurations of the gateway can be created, and the corresponding configuration is deployed when needed, at the runtime. If the requirements change during the mission, and there is no previously created configuration for the new situation, the user can create a new configuration by using the gateway configurator. After the new configuration is done, the changed parts of the gateway are re-compiled, and new executables are created. The new executables with the new configuration are deployed, then. To allow this runtime reconfiguration, the gateway code that is responsible for user interaction is separated from the code that is conducting scenario-specific data transmission and conversion. In this way, the user can change the corresponding gateway parts at the runtime without stopping it completely. Figure 3.4 depicts this process.

3.5 Case Studies

This section includes case studies to show the applicability of the proposed gateway approach. An industrial case study by integrating the gateway into the HAVESIS

project, and an academic case study that tests different aspects of the gateway are conducted.

3.5.1 Distributed Tactical Environment Simulation

The DDS-HLA gateway was developed within the scope of the HAVESIS project. HAVESIS has four main components: Aircraft Emulator/Simulator Subsystem, Tactical Environment Simulation Subsystem, Middleware Subsystem, and External Load Subsystem. For an overview of the HAVESIS architecture, the reader may refer to [66]. The Middleware Subsystem is responsible for data transmission among the subsystems of HAVESIS. Therefore, The DDS-HLA gateway is deployed into the Middleware Subsystem.

Middleware Subsystem provides a publish/subscribe mechanism for communication of HAVESIS components ensuring certain quality requirements. The subsystem provides a DDS-based middleware service that enables virtual and real systems to be run together to facilitate platform integration of avionics systems. It is also responsible for the management of the middleware QoS, and it hosts the DDS-HLA gateway. Configuration management of the Middleware Subsystem uses the file management mechanism of the operating system on which the subsystem is running. The configuration files (".idl" and ".fed") are updated by the user to read/write data in the format required by the configuration technology (DDS and HLA) through an editor.

Middleware Subsystem composed of two components, namely Data Distribution Component, and DDS-HLA Gateway Component. Data Distribution Component contains OMG 1.4 [91] compatible DDS interface configuration and their functions. This component is offered as a commercial off-the-shelf product. Nevertheless, OpenSplice Community 6.7 [92] is used as a distributed open-source software for concept verification purposes. Thanks to DDS, the parties conduct communication by means of certain rules defined in the context of a publish/subscribe mechanism regardless of the communication media and the connection protocol.

A Tactical Environment Simulation (TES) software, which has been developed as an in-house project for HAVESIS, is employed as a proof of concept for the proposed

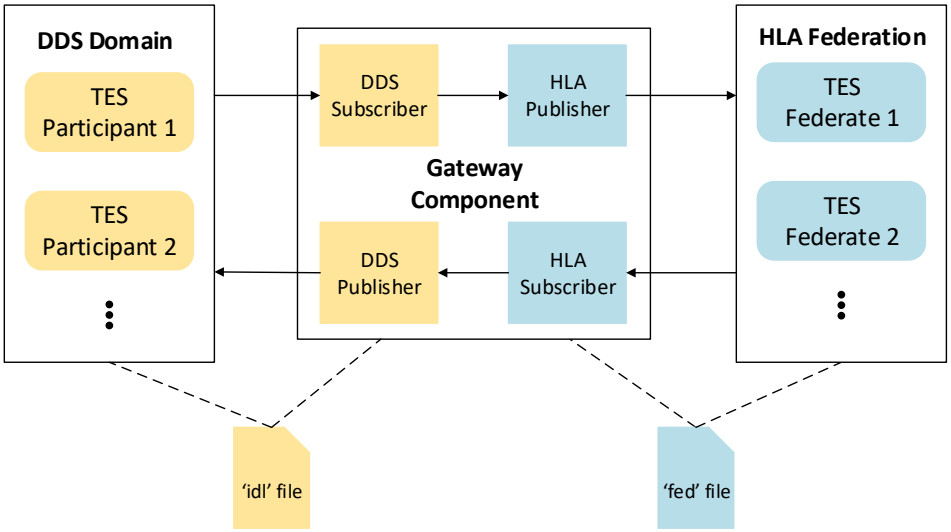

Figure 3.5: TES Dataflow through DDS-HLA gateway.

gateway approach. TES, as a flexible, deterministic, and monolithic discrete-event constructive simulation software, is a subsystem of the HAVESIS project. TES allows running in a distributed environment by executing the instances of the same core initialized to handle different portions of a simulation scenario. Moreover, the distribution infrastructure can be modified via TES build settings to operate either on DDS or HLA.

Two configurations of the TES software is used for this case study: The first one is the DDS compliant TES subsystem that uses OpenSplice 6.7 library. The second one is an HLA compliant TES subsystem and it uses Portico 2.1.0 [88] library. Both configurations are set up to exchange winged aircraft objects with id, name, state vector (position, velocity, roll, pitch, and heading) and other properties. Moreover, HLA compliant TES uses two interactions (AdminInteraction and ServerStateInteraction) for scenario management (such as start, stop, pause) operations, which are defined in its ".fed" file. For the DDS side, these HLA interactions and winged aircraft are defined as topics in the ".idl" file. The gateway component is configured and generated for this scenario. The relationship of the gateway component with the TES software is given in Figure 3.5.

TES runs in two different modes: master or slave. It reads the scenario file and

controls other TES software in the master mode. In the slave mode, it accepts commands from the master to run the scenario. A test scenario is established involving two aircraft: one controlled by the TES HLA Federation and the other by the TES DDS Domain. In this scenario, initially, TES Federation is executed in master mode and TES Domain is executed in slave mode. A DDS domain and an HLA federation are created. After that, Gateway Component is executed and its Gateway Participant and Gateway Federate join the pre-created domain and DDS federation, respectively. Then, the TES Federation starts the scenario and both TES instances create an aircraft entity and begin exchanging aircraft state updates over the gateway component. TES Federation publishes object attribute updates of its aircraft object. These updates are received by the HLA Subscriber of the gateway component since it has a relevant subscription to the federation. The data captured by the HLA Subscriber is converted to a DDS message in the gateway and transferred to the DDS Publisher of the gateway component. DDS Publisher publishes this message and the message is received by the TES Domain. For the other direction, TES Domain sends its aircraft's updates through the gateway in a similar manner but the relevant operations are conducted in reverse order. Figure 3.5 indicates the data flow in this scenario by using arrows.

The performance of the gateway component during this case study is provided in Table 3.1. The table contains delays experienced throughout a sample run of the defined scenario. Experiments are conducted on a computer with Intel© Core i7-6700 CPU and 8 GB of RAM. Both TES instances and the Gateway were executed on the same computer. In the scenario execution, the flight of two aircraft is simulated for 15 seconds. The provided results in Table 3.1 are the averages of 3000 messages that are sent during the execution. The provided results are intended to give a rough idea of scale and are prone to change with scenario variations and network configuration.

3.5.2 Use Cases for Dynamic Reconfiguration

Some inter-operations require the runtime configuration change during the execution of the process. This subsection introduces some typical use case scenarios illustrating the need for runtime reconfiguration capability of the proposed gateway.

It is sometimes desirable to transfer the attribute ownership for a simulation instance,

Table 3.1: Gateway delays introduced during data flow for the TES case study.

Flow direction	From	To	Delay (μsec)	Total (μsec)
DDS $\rightarrow$ HLA	TES DDS P.	Inner DDS S.	7209	24764
	Inner DDS S.	Inner HLA P.	8006	
	Inner HLA P.	TES HLA S.	9549	
HLA $\rightarrow$ DDS	TES HLA P.	Inner HLA S.	16195	103992
	Inner HLA S.	Inner DDS P.	82251	
	Inner DDS P.	TES DDS S.	5546	

in order to facilitate the cooperative modeling of an object within the simulation [93]. Although ownership transfer is supported both by HLA within a federation [70] and by DDS within a topic space [94], a transfer among HLA and DDS is not straightforward through a gateway. At this point, alteration of the ownership properties of required objects could be updated during execution, by means of the runtime reconfiguration property of the gateway. The transfer request can trigger a reconfiguration process, which starts with the syntactic update of the configuration files for both DDS and HLA sides, followed by the reconfiguration module to incorporate the new configuration during runtime.

For embedded training settings, embedded virtual simulation components are used for the integration of training functionality into operational equipment [95]. While the integration through DDS is quite common in operational platforms, HLA is the standard for simulation interoperability. The DDS-HLA gateway might offer a more flexible design for inter-protocol embedded training networks. Moreover, with runtime reconfiguration, it is easier to cope with hardware equipment related errors, since it provides the opportunity to make recovery plans beforehand to replace malfunctioning subsystems online during training.

A straightforward, yet very handy use of runtime reconfiguration is the closed-loop experimentation. If the simulation is closed-loop and driven by an experiment design, the same simulation scenario is to be repeated a number of times, depending on the experimentation parameters, such as levels, number of trials, etc. In that case, each execution of the scenario is, in fact, an altered instance of the *base* scenario. A

number of the altered scenarios are likely to require configuration changes, which can be managed via repetitive use of the runtime reconfiguration property of the gateway throughout the experiment execution. Obviously, an experimentation policy definition mechanism would be required to ease the reconfiguration definition and incorporation.

3.5.3 A test case for Dynamic Reconfiguration

In this scenario, simulation of an aircraft and its communication with ground stations is concerned. Each ground station has its area of responsibility. The aircraft communicates with one ground station at a time: the responsible ground station of the flight zone in which it is flying. When the aircraft moves from one zone to another, the station which it is in contact changes as well.

We consider a two-staged mission for the aircraft where it flies from one zone (Stage 1) to another (Stage 2). This transition also changes the contacted ground station. While ground stations simulated as an HLA-based system, the aircraft is simulated by DDS. Thus, the DDS-HLA gateway mediates the communication between the aircraft and the ground station. We assume that these stations have different communication parameters, e.g., they read/write through various object attributes in the HLA federation. The exchanged information contains the current coordinates of the aircraft, commands from the ground station, etc. For testing purposes, the size of the exchanged messages is set to be the same.

In Stage 1, the aircraft is expected to send its geographic coordinates in every 12 seconds and receive the updated commands from the ground station. In stage 2, the new ground station wants to receive the aircraft's coordinates as a list in every 60 seconds, for a total of 5 messages of 12-second steps. In response, it sends its commands as a list of 5 items. The gateway has a different configuration for each stage to keep operating correctly.

The gateway is deployed between the aircraft and the first ground station at Stage 1. It is assumed that the communication details of the second ground station and the necessary configuration of the gateway to establish a connection are known (the

Table 3.2: Gateway delays introduced during Stage 1.

Flow direction	From	To	Delay (ms)	Total (ms)
DDS $\rightarrow$ HLA	Aircraft	Inner DDS S.	99.85	172.86
	Inner DDS S.	Inner HLA P.	14.54	
	Inner HLA P.	Ground St. 1	58.47	
HLA $\rightarrow$ DDS	Ground St. 1	Inner HLA S.	48.44	476.49
	Inner HLA S.	Inner DDS P.	12.35	
	Inner DDS P.	Aircraft	415.70	

configuration for Stage 2) before. However, it is preferred not to deploy codes needed for Stage 2 to the gateway in advance. In other words, the unnecessary code during the first stage is avoided. If the communication requirements for a new stage is discovered during the mission, and the configuration for the gateway is not prepared before, a new configuration can be generated during the mission and deployed, as explained in Section 3.4.3.

For testing purposes, the operation of each stage was observed for an equal amount of time. Firstly, in Stage 1, a total of 25000 messages were sent; 12500 from DDS to HLA and 12500 from HLA to DDS. Then the gateway is triggered for the dynamic configuration. After the successful deployment of the configuration for Stage 2, a total of 5000 messages were sent; 2500 from DDS to HLA and 2500 from HLA to DDS. The DDS-based system, the HLA-based system, and the gateway are simulated on the same PC that has a different processor and memory features than the PC described in Section 3.5.1. The network delays are provided in Tables 3.2 and 3.3 for Stages 1 and 2, respectively, to give a rough idea about the gateway execution. The network performance observed in Section 3.5.1 is different from the performance observed in this section: While DDS to HLA performance is better in the setting described in Section 3.5.1, HLA to DDS performance is better for the tests in this section. Therefore, we can say that the performance of the gateway is highly dependent on the host and systems that the gateway provides interoperability for them. It is worth noting that, our primary goal is achieving interoperability rather than improving the network performance of the gateway that may be subject to different research.

Table 3.3: Gateway delays introduced during Stage 2.

Flow direction	From	To	Delay (ms)	Total (ms)
DDS $\rightarrow$ HLA	Aircraft	Inner DDS S.	102.81	165.24
	Inner DDS S.	Inner HLA P.	17.10	
	Inner HLA P.	Ground St. 2	45.33	
HLA $\rightarrow$ DDS	Ground St. 2	Inner HLA S.	49.80	274.19
	Inner HLA S.	Inner DDS P.	14.13	
	Inner DDS P.	Aircraft	210.26	

There are also different results for Stage 1 and Stage 2. Although the size of the information exchanged in both stages are about the same, performance in Stage 2 seems slightly better. It is because the number of messages is decreased using data structures in Stage 2. Aircraft used a list of structs as a DDS participant in Stage 2, and the second ground station employed a list of fixed records in HLA. The gateway successfully converted these types to one another.

3.6 Variability-Guided Gateway Development

This section includes our suggestions about a variability-guided gateway development process for LVC systems. Our previous work [1, 96] regarding compositional development inspires this gateway development method.

The variability-guided development approach aims to increase reuse and reduce development time and costs while achieving interoperability. The systematic definition of common and variable parts of the gateway helps reusing already implemented parts of the gateway. Feature model [57] and Orthogonal Variability Model (OVM) [58] are famous models for showing common and variable parts of a system and can be used to guide the development of the gateway. Figure 3.6 is an excerpt from an example feature model for the aviation domain. From a top-down development point of view, gateway development may start choosing the two architectures to be operated together. In Figure 3.6 this option are represented by four options that are *TENA*, *DDS*, *HLA*, and *DIS*. Our assumption is placing the gateway between two architec-

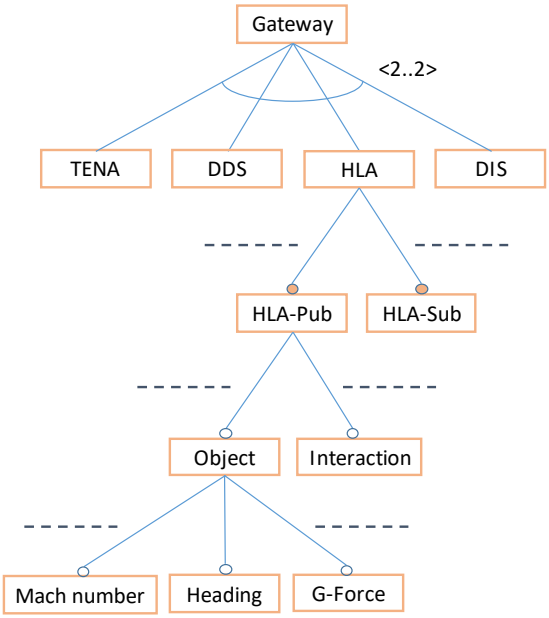

Figure 3.6: An excerpt from a feature model prepared for the aviation domain.

tures. Thus, a numerical constraint may represent this choice, such as <*2..2*>, forcing the developer to select the two architectures that the gateway to be placed in between. Then, as we use in the DDS-HLA gateway, publisher and subscriber of the selected architecture must be included in the gateway code. In another level of the model, different data descriptions, such as Object and Interaction classes for the HLA, can be included in the gateway. Aviation domain-related parameters can be added to these classes, as shown at the bottom-most level of the example feature model.

Another option for variability representation is the OVM model. Although the feature model can show commonality and variability as well, OVM explicitly differentiates variation points and variants. Also, OVM is a flat model, instead of being hierarchical, that may ease to assess alternatives for a specific variation point.

The development approach suggested in this section would be possible in the case of the existence of a mature domain [97]. A mature domain can be defined as populated with developed components. After having a mature domain, products can be derived directly from a configured domain feature model, i.e., the product feature model with the help of a code generation tool such as *pure::variants* [98].

3.7 Discussion

Gateway Component can be deployed and executed on any computer system on the local area network. To increase efficiency, the gateway should be located close to the publishers and/or subscribers of the related topics, e.g. on the same computer if possible. Moreover, it is possible to configure and deploy multiple gateway components for a single LVC application considering various topics and Simulation Object Models (SOMs). This allows for a balanced distribution of gateway functionality. In a scenario where multiple gateway components are deployed, the topics on the domain should be segregated among the gateways. In this way, efficient use of resources on the network through distribution is provided and the generated network traffic is minimized. In the final system, multiple gateways may be combined into a composite gateway and presented as a single application.

While providing interoperability between a DDS-based system and an HLA-based system, there may be a need for data conversion. Gateway configuration is done by using another tool and necessary data structures and functions are generated and compiled. The deployment of the corresponding gateway code between the two systems can be done at development time or runtime. Gateway configuration requires to know in advance the data types used by the DDS-based system and the HLA-based system. An IDL file contains the data types used by a DDS-based system. In the proposed gateway approach, HLA FOM files are assumed to include the necessary data type declarations just like in the IDL file of the DDS.

In order to prevent congestions, the gateway is designed for two-way communications. Moreover, the incoming data to the gateway, whether the source is the DDS-based system or the HLA-based system, has to be unpacked before transferred or converted. These decisions come with a cost by increasing the copying of the data inside the gateway. Therefore, it is worth noting that, most of the delay reported in Table 3.1 stems from parsing and conversion effort.

The presented gateway is assumed to be deployed between two different architectures. In other words, the current version is not capable of handling communication of more than two architectures. However, the architectural design of the gateway allows this

kind of enhancement. Moreover, there is some research, including [62], that claims to solve semantic mapping issues between DDS-HLA. In the current version of our gateway, we deploy the code for semantic mappings into the skeleton code structure, i.e., the *commonality* part of the gateway. Thus, we can say that our gateway is more appropriate for syntactic interoperability. Furthermore, in this version, we are not dealing with non-functional aspects of the interoperability, such as QoS parameters in DDS. To summarize, our gateway provides functional and syntactic interoperability for binary connections for components or systems.

Although increasing gateway's performance is not our primary goal at this stage of our work, we can state based on our observations that the gateway performance highly depends on the configuration of both the DDS-based system and the HLA-based system, and also the host of the gateway. For example, network delays for DDS to HLA are better in the TES example, as introduced in Table 3.1. However, in the scenario described in Section 3.5.3, HLA to DDS messaging is less time-consuming. It can easily be observed from the results that the own performance of the DDS-based system is better in the TES scenario: average network delay of messages that are sent from TES DDS Publisher to the gateway is better than the delays of messages from TES HLA Publisher to the gateway. This is the opposite of the scenario described in Section 3.5.3.

It is also notable that DDS is used a lot for interoperability purposes. Most of the works presented in the related work section use DDS as infrastructure or combine it with another architecture or standard to provide interoperability for heterogeneous components. This is because DDS is a powerful tool with its broad QoS support and has established itself in various industries.

In the current version of the gateway, the variability models are considered only for guiding the developer when configuring the gateway. The variable parts of this version are related to data transfer and conversion: other parts of the gateway (such as codes that help the gateway to enter a DDS domain) are considered as commonality and included in the skeleton code structure. Gateway code generation is conducted by the configuration UI, not by variability models yet.

3.8 Conclusions

A dynamically reconfigurable gateway application for DDS-HLA interoperability is presented. The increasing importance of LVC interoperability to reduce operation costs, especially in the defense industry, calls for this kind of solution. An architectural connector-based solution is implemented as a gateway between the DDS-based and HLA-based systems. An early version of the gateway was successfully integrated into the HAVESIS project. Then, the enhanced version of the gateway, in which structured data type transmission and runtime reconfiguration abilities are added, is developed and other case studies are conducted. What separates the proposed gateway approach and other similar solutions in the literature is the configurability. The gateway can be configured to transfer various data types, including structured ones, and perform conversions when needed for both sides of the communication: from DDS to HLA and vice-versa. Different configurations of the gateway can be deployed between the DDS and HLA systems at different development times. Moreover, the configurability of the gateway is related to systematic variability modeling.

CHAPTER 4

INTERNET OF MEASUREMENT THINGS ARCHITECTURE

Many industries, such as manufacturing, aviation, and power generation, employ sensitive measurement devices to be calibrated by certified experts. The diversity and sophistication of measurement devices and their calibration needs require networked and automated solutions. Internet of Measurement Things (IoMT) is an architectural framework that is based on the Industrial Internet of Things for the calibration industry. This architecture involves a layered model with a cloud-centric middle layer that solves interoperability issues of different applications. In this chapter, the realization of this conceptual architecture is described. The applicability of the IoMT architecture in the calibration industry is shown through an editor application for Scope of Accreditation. The cloud side of the implementation is deployed to Microsoft Azure. The editor itself is created as a cloud service, and IoT Hub is used to collect data from calibration laboratories. By adapting the IoMT architecture to a commonly used cloud platform, considerable progress is achieved to encompass Metrology data and serve the majority of the stakeholders.[1]

4.1 Introduction

The usage of the Internet of Things (IoT) spreads to different domains as it provides opportunities in big data analytics, machine learning, and cloud computing technologies in the industry. This emerging approach is called the Industrial Internet of Things (IIoT) or Industry 4.0. Considering the benefits of this new trend, such as increased productivity, short development periods, quickly developed customized products, and resource efficiency [99], numerous solutions are proposed in different areas, includ-

[1] The study described in this chapter was published in the *Sensors* journal in 2020 [12].

ing farming, manufacturing, and telecommunications. When these characteristics of IIoT technologies are taken into account, they seem to provide promising solutions for the open research problems in Metrology and the calibration industry.

The notion of the "Internet of Measurement Things (IoMT)" was proposed in [100] as a layered IIoT architecture that separates physical equipment, cloud-based services, and applications. This architecture is inspired by the Metrology Information Infrastructure (MII) initiative and previous experiences with the Metrology.NET platform. MII and Metrology.NET are efforts to develop community-driven standards and to increase the usage of automation in the Metrology world. IoMT architecture aims to advance previous work into an IIoT-based solution.

The IoMT architecture has three layers, namely *physical*, *MII Cloud Services*, and *application*. The physical layer contains equipment for calibration, generally in calibration laboratories (CLs). The MII-Cloud Services layer hosts the services that are provided for the calibration industry. The application layer constitutes different sorts of software that can be used in Metrology and the calibration industry, e.g., calibration automation systems, asset tracking systems, and scope of accreditation (SoA) editors.

CLs operate with different capabilities. Their services may cover one or more disciplines such as Dimensional, Electrical, and Mechanical. Customers choose CLs by checking their capabilities and whether the lab is certified by accreditation bodies (ABs) pertaining to these capabilities. However, using current methods, customers often have limited options to find the lab that best suits their needs.

An accreditation certificate declares the lab's capabilities and guarantees an approved quality of service to customers. For an accredited CL, SoA represents a documented list of calibration fields, specific measurements, uncertainty values, and other parameters. A certificate of accreditation is accompanied by the scope document, and the certificate is incomplete without it. The SoA document includes only the calibration areas that a laboratory is accredited for, and only the listed areas may be offered as accredited calibrations to customers. The format and other details are usually defined by the AB.

In this chapter, the realization of the conceptual IoMT architecture is elaborated. An SoA editor is chosen for the case study. Previous work showed the principles of the editor, some implementation detail, and how the editor fits the IoMT architecture. However, the relationship of the editor with the cloud environment was conceptually explained. One of the well-known and well-established cloud computing service providers, Microsoft Azure, is employed as the underlying platform, and the SoA editor is implemented as a cloud service. The data flow from the calibration devices in the labs to the cloud is managed by Azure IoT Hub.

The whole work done on the IoMT concept can be summarized as follows:

- The IoMT architectural framework was presented in [100]. Based on MII concepts and Metrology.NET experiences, an IIoT-based architecture was conceptually introduced to the calibration community.
- The SoA editor was introduced in [101] as an application conforming to the previously proposed IoMT architecture. The components of the SoA editor fit the architecture. However, the connections among the components of the editor and the MII services remained conceptual.
- In [12]:
  - The realization of the IoMT architecture is explained, including cloud-side implementations.
  - Microsoft Azure is chosen as the cloud provider.
  - The SoA editor presented in [101] is chosen for the case study and re-implemented as a cloud service.
  - The implementation is partial for proof of concept. On the other hand, the necessary steps to implement a full-scale IIoT application using Azure are explained.
  - Calibration devices are employed as IoT devices, and Azure IoT Hub is used to collect data from them.

Although some research enriches Metrology with IoT technologies in recent years, we can still say that we are witnessing the beginning of this transition process. More-

over, to the best of our knowledge, there is not much work done to make calibration automation easy for all stakeholders in the industry. In these aspects, this work is a contribution to adapt Metrology and the calibration industry to IoT technologies in the industrial scale. This improvement would ease the development of new applications that use machine learning and big data analytics.

We employ calibration equipment as data producers of an IoT application. In this sense, sensors in a common IoT application are replaced with calibration equipment in the IoMT architecture. From another viewpoint, our architecture comprises applications and labs that focus on sensor calibration. An example of the use of IoT in sensor calibration is provided in [102]. Moreover, standardization is essential for sensor calibration [103]. Using the cloud for commonly used services would help compliance with standards.

The rest of the chapter includes required background information that covers the SoA concept, the MII initiative, the Metrology.NET platform, and Microsoft Azure along with its IoT-related services. The related work is discussed, including similar IIoT applications from different industries and similar work that use Azure. After the related work section, the IoMT architecture is elaborated. Then, the SoA editor is provided in detail along with the cloud implementation: how to develop the SoA editor as a cloud service and other Azure services used in the proof of concept case study. The chapter is concluded with remarks and possible future work.

4.2 Background

In this section, an introduction to the essential calibration domain terminology used in this work is provided. A brief overview of the MII is also given.

4.2.1 Calibration

Calibration is an area of metrology, the science of measurement. It indicates the quality and accuracy of measurements performed in a domain. Measurement equipment may have errors in their results as time progresses. These drifts can be caused by

misuse or some external factors, such as temperature and humidity.

Each measurement device has a margin of error on its measurements, namely the measurement uncertainty. By adjusting the device, it is intended to minimize the measurement uncertainty or ensure that it remains at an acceptable level. Adjustment is a different step that is separate from calibration and usually follows a calibration job. Calibration only includes the testing of a customer unit, while the adjustment is made to fix the possible divergence of a unit from its regular operation [104].

4.2.2 Scope of Accreditation

All CLs that are accredited as per ISO/IEC 17025 [105] (International Organization for Standardization/International Electrotechnical Commission) have an SoA. The scope of a lab represents its technical capabilities for which the lab has requested the accreditation and was deemed to be competent by an AB. Customers can refer to the SoA of a CL to find out if their unit can be calibrated by that lab and with how much accuracy. The scope includes further details such as testing methods, types of inspections, and certifications. This way, customers can also compare calibration service providers and choose the one that best suits their needs.

Calibration and Measurement Capability (CMC) represents the best achievable measurement uncertainty of ideal measurement equipment under normal operational conditions in a CL. ABs assess CLs by these numbers, based on CLs' personnel, equipment, and processes. These values are represented in the form of constant values or as formulas.

Table 4.1 shows an excerpt extracted from the SoA certificate of a CL issued by an AB. The excerpt includes two parameters concerning two different calibration disciplines covered in the CL scope. Temperature-Measure (first row) is a Thermodynamics parameter, and DC Voltage-Measure (second row) represents an Electrical parameter. The table shows uncertainty values (CMC column) for the specified ranges using specific test equipment mentioned in the *Comments* column. The *Range* column may include a fixed point or a range of values. Similarly, the *CMC* column may include a constant value or a CMC equation (second row). The parameters and the formulation

shown in an SoA certificate may consist of other variations and more complexities than what is shown in Table 4.1.

Particular infrastructure needs to exist to provide better solutions for SoA related problems. Such an infrastructure would provide data type standardization, communication protocols, technologies, and services, and it would lay the foundation for automation solutions for accreditation processes. In recent years, there have been endeavors in the advancement of such an infrastructure. The NCSLI (National Conference of Standards Laboratories - International) MII working group is one of the most active metrology communities working on digitalizing the SoA activities [106].

Table 4.1: An excerpt from an SoA certificate showing CL scope for two different parameters.

Parameter/Equipment	**Range**	**CMC (±)**	**Comments**
Temperature-Measure	(-50 to 0) °C	0.36 °C	Fluke 744
	0 °C	0.37 °C	
	(0 to 100) °C	0.41 °C	
	(100 to 250) °C	0.46 °C	
DC Voltage-Measure	(0 to 100) mV	6.8 μV/V + 0.86 μV	HP 3458A
	(0.1 to 1) V	6.0 μV/V + 0.80 μV	
	(1 to 10) V	6.7 μV/V + 1.3 μV	
	(10 to 100) V	7.0 μV/V + 32 μV	
	(100 to 1000) V	7.8 μV/V + 59 μV	

4.2.3 Metrology Information Infrastructure

Standards are essential for metrology to achieve reliable results. Similar to other domains, metrology also needs automation and standardization. Standards can help to increase quality in metrology when applied to various processes or data. Examples can be listed as documentation, conformance testing, risk analysis, uncertainty analysis, product inspections, service procurement, and accreditation.

Mark Kuster introduced MII in NCSLI Metrologist magazine in 2013 [107], initially as Measurement Information Infrastructure. The motivation of MII is standardizing data types in metrology, including SoA, instrument specification sheets, and calibration and testing certificates [108]. In this way, it is aimed to ease the communication of world-wide measurement-related systems and replace manually processed documents with unambiguous machine-readable ones.

Figure 4.1 illustrates the flow of data among some critical stakeholders of the metrology domain. Instrument specs, certificates, and SoA are different types of metrology data shared among the stakeholders. Arrows in the figure represent the directions of the data flow.

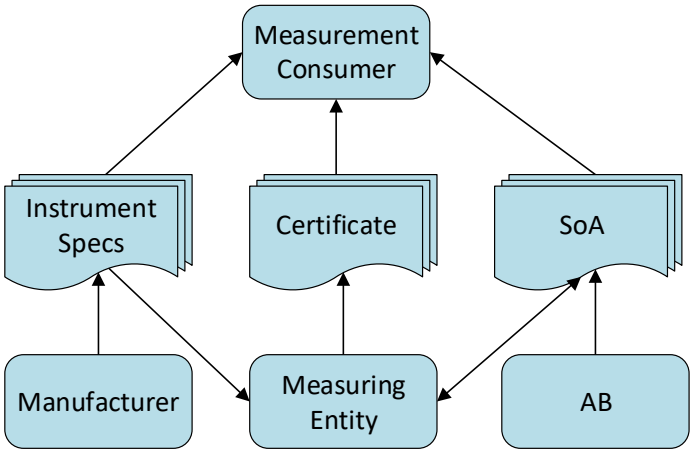

Figure 4.1: A Partial view of the metrology information flow.

4.2.4 Metrology.NET

Metrology.NET is a distributed automation platform for calibrating test equipment [109, 110]. The platform design follows a modular approach for data management and calibration automation. The platform connects multiple sub-systems to form a system of systems to fill in the gaps among various metrology software systems currently used at CLs. It follows the Lego® analogy, where sub-systems are similar to Legos that connect using a connector layer. The standardized connector layer is supposed to join different systems together, which allows the user to build up a total solution by choosing and configuring smaller block systems to work together.

Figure 4.2 shows an overview of the Metrology.NET platform, and the system decomposition with the connections among system components. The metrology engineer is usually in charge of the server-side that commands the agents for their jobs. The agent (testing terminal) is the computing machinery that communicates directly and physically with the testing hardware, including the unit under test (UUT) and reference equipment. The agent runs the automation software. The typical way of connecting the agent to the UUT and reference equipment is through the GPIB (General Purpose Interface Bus).

The Metrology.NET platform follows a client-server architectural style. The server side hosts the application services, and the clients act as testing workstations that run the automation. Depending on a CL capacity, it can only consist of a single machine running both the server and the client, or there may be a central server machine with several client terminals that consume the services provided by the server. The web interface of the server-side allows technicians who run calibration work orders to locally or remotely control and interact with the running automation process.

The server side of the platform can be physically located anywhere, as long as there is a network connection between the server and the client(s). A secure and fast network connection between the remote server and the clients ensures automation gets done in the correct order. The clients act like worker bees that collaboratively do the calibration job. They are all controlled through the central server. The server side also keeps a database of the calibration-related data, including test points. This provides a centralized monitoring of the increasing data and the involved processes. The server can manage shared calibration jobs that can occur across labs and the data communication among labs. Another option for the server is to be hosted by a cloud service and take advantage of the cloud technologies if that helps the business.

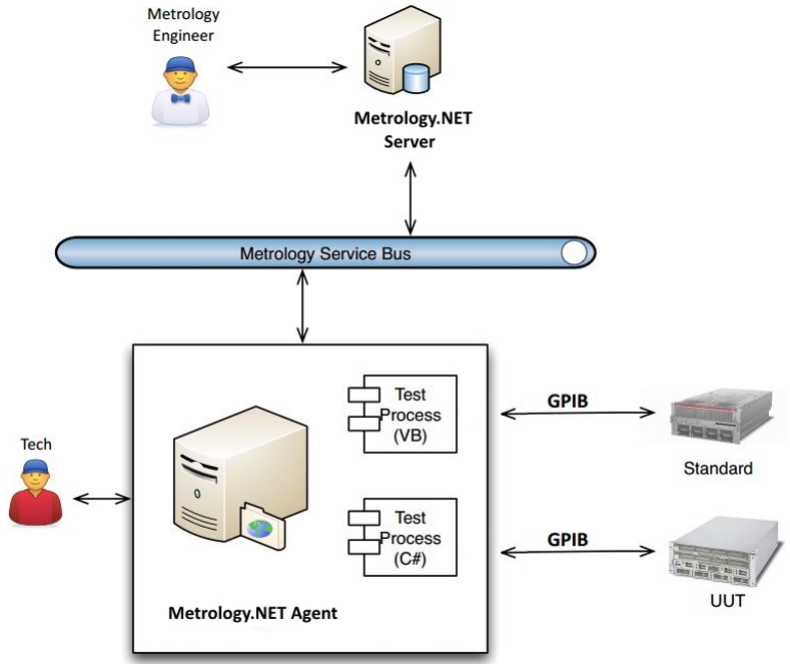

Figure 4.2: An overview of the Metrology.NET system.

Metrology.NET seeks for separation of concerns in handling calibration data and processes. A calibration technician examines a calibration work order in terms of a set of test points for specific customer equipment. From that aspect, the calibration task is the process of collecting measurement results for these test points. Once the results are obtained, the job is almost complete, and the server-side can review these data and issue the related certifications for the tested instrument.

Fully automated calibration increases productivity, accuracy, and repeatability in calibration processes. In Metrology.NET, a calibration task is a composition of smaller reusable test modules that aim to test specific functionalities of a UUT using known reference equipment configuration. Multiple subsets of test points may be passed to different modules to perform the automation. Modules are loosely coupled, and they handle their own job and send their results back to the application server.

4.2.5 Microsoft Azure and IoT Services

Microsoft Azure is a cloud computing platform that contains Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). Providing flexibility in using different programming languages and communication protocols, Azure simplifies the development, deployment, and management of distributed applications for various application fields.

Azure Cloud Services is the PaaS environment of Azure [111]. Locally created applications are deployed to Azure cloud service and can run together with other Azure services to compose a broader application.

Microsoft provides a set of cloud services to connect, monitor, and control IoT assets [112]. These services are collectively called *Azure Internet of Things*. Some of these services are *IoT Central*, *IoT solution accelerators*, *IoT Hub*, and *Azure Digital Twins*. Users can select the service they need by deciding the amount of control they want to have over the application. Also, some of these services provide templates based on common IoT solutions. Alternatively, users can build their applications from scratch.

4.3 Related Work

This section contains related work in two main categories: Metrology and IoT applications, and Azure usage in the contexts of IoT and IIoT.

4.3.1 Metrology and IoT

The history of using IoT technologies in metrology applications is not very old. In one of the recent works, Lazzari et al. [113] discuss the *smart metrology* term, arguing that the big data collected in the industry is meaningful when it is reliable. Smart metrology, a new interpretation of metrology based on reliability, is presented as a solution. As an example of the benefits of this approach, it encourages Metrologists to re-evaluate calibration intervals instead of regular calibrations enforced by the law.

In a literature survey, Daponte et al. present measurement applications based on IoT

[114]. The authors investigate related work in different application fields: intelligent transportation systems, smart and connected health, smart energy, smart environment, smart building, and smart factory.

In a white paper [115], Monnier presents smart grid solutions and combines smart meters with IoT. Also, existing smart grid connection approaches in the literature are discussed. Then, solutions provided by the author's company for smarter and more connected smart grids are given. Angrisani et al. propose a LabVIEW-based platform for remote programming of automatic test equipment [116]. It may be hard to have all the necessary devices in the same lab when training technicians. In such a scenario, it is critical that a lab with the necessary devices shares its resources with other labs. Moreover, the platform allows connecting to a device and programming it remotely.

4.3.2 Azure usage in IoT and IIoT systems

There is plenty of research leveraging Azure in IIoT and IoT applications in the literature. This section covers some of them.

A recent study that employes Microsoft Azure in the IIoT context is conducted by Haskamp et al. [117]. They explain the process of retrofitting a legacy automation system to obtain an Industry 4.0-compliant system. In another study, Raju and Shenoy explain the benefits of using the capabilities of cloud and IoT in the industrial domain [118]. They use Azure IoT Hub in their case study.

Forsström and Jennehag evaluate an IIoT system's monetary cost and network response time performance [119]. The system uses OPC-UA (Open Plant Communication Universal Architecture) and Microsoft Azure IoT Hub. Their case study includes a 1500-sensor real-life industrial system. They present results for fiber-based and mobile internet communication. Also, a cost-wise evaluation is provided based on the price plan of Azure IoT Hub.

Another category of research covers the Azure usage in IoT applications, not necessarily in the industrial context. Shi et al. present their robust end-to-end security solutions for IoT systems with limited budget in [120]. They integrate Azure Sphere microcontroller unit and Microsoft Azure cloud services in their solution. Azure IoT

Hub is used as the cloud gateway that handles message traffic between Azure services and IoT end devices. Detailed descriptions of the system both for hardware and software components and test results are provided in the study.

Microsoft Azure IoT cloud server is used to handle real-time traffic flow-data in another study [121]. Collected data is analyzed to evaluate signal timings, and waiting times are reduced. Another work suggests controlling of smart switches through a web application and cloud technologies [122]. Microsoft Azure SQL (Structured Query Language) database is used on the cloud side. Moreover, a reference architecture is presented for IoT applications to ensure security and privacy [123]. Microsoft Azure is used as the provider in their movie suggestion application case study.

Al-Masri et al. propose an approach to improve urban waste management [124]. Recycle.io, an IoT-enabled approach, allows categorizing wastes at the time of disposal. Smart recycle bins equipped with cameras and sensors are used. Collected images and data are processed at the edge of a network and cloud. Microsoft Azure IoT Hub is used for device management.

Another similar research that employs cloud technologies in IoT applications is conducted by Ferrández-Pastor et al. [125]. The user-centered design model is used to obtain knowledge of farmers to develop IoT systems for agriculture. Edge and fog computing paradigms are used to implement IoT architecture, operating rules, and smart processes.

4.4 Internet of Measurement Things

This section explains the previously proposed IoMT concept and provides a detailed description of the proposed architecture. Based on the need for standardized software in the calibration industry, an architectural framework was proposed [100]. This architecture contains three layers, as shown in Figure 4.3: the Physical layer, the MII Cloud Services layer, and the Application layer. This layered model conforms to the reference architecture for IIoT proposed by Industrial Internet Consortium (IIC) [126].

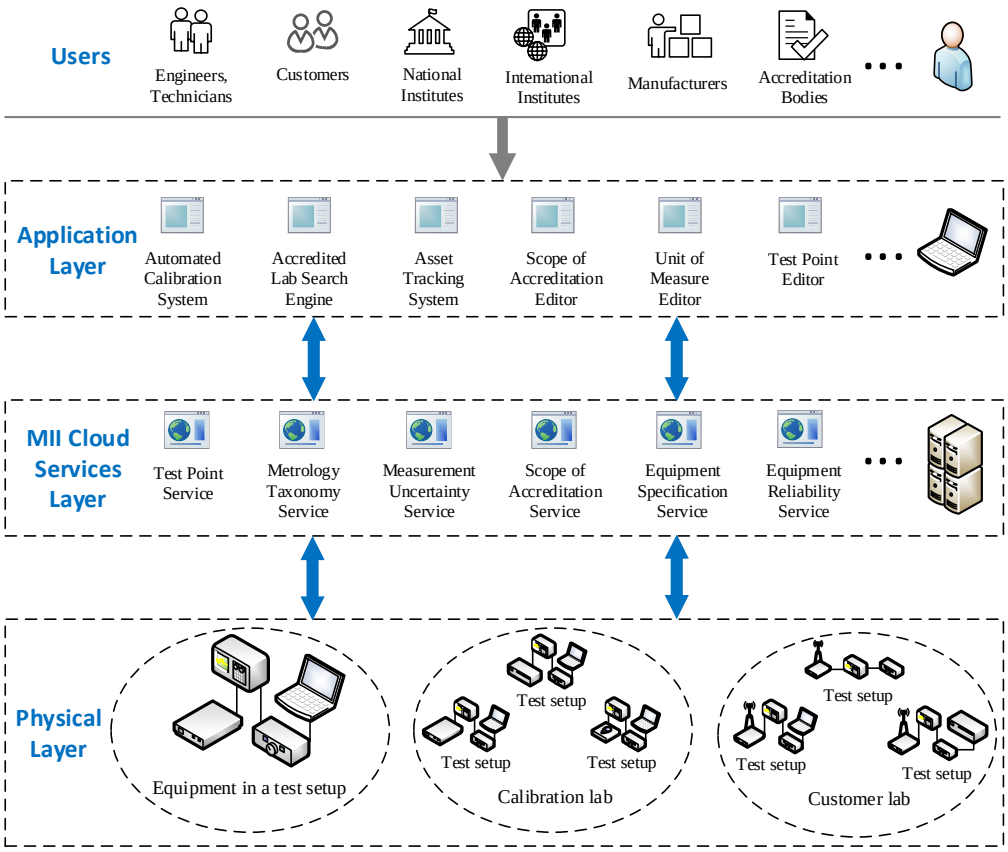

Figure 4.3: The IoMT architecture description.

The physical layer consists of all the physical infrastructure in the calibration industry, which produces measurement data. Physical equipment in this layer can be thought of in different scales and configurations. For example, a small setup may contain a UUT, and a reference device and a complicated setup may comprise tens of calibration setups running at the same time. The physical layer can encapsulate all organizations and domain people as a connected network at the largest scale.

The MII cloud services layer comprises the formatted measurement data and the services that use it. Robust services based on agreed standards and protocols can encourage different organizations in the domain to adhere to the standardization. Therefore, smaller solutions to the industry's common problems are provided in this layer to be added up to create bigger solutions. To provide interoperability for diverse applications, the data used by these services should be based on the standard formats and schemas developed by the MII community. In this scenario, applications that reside

in the application layer should also conform to the same standards.

The application layer consists of all different sorts of software that can be exploited in the domain. Applications in this layer use services and the data stored in the MII Cloud Services layer. An application can benefit from several services, and multiple applications can use a service at the same time. Some examples of applications in this layer are automated calibration system, accredited lab search engine, unit of measure editor, and scope of accreditation editor. Some of these applications are already implemented and in use (e.g., Qualer search engine [127]); they do not have an IoT perspective, though. However, they can easily be adapted to the IoMT architecture if they use MII cloud services since they are MII-aware.

In Figure 4.3, different users of the calibration industry who have different connections to the domain are depicted above the application layer. Some of these users are ABs, national and international metrology institutes, CLs, calibration software companies, equipment manufacturers, and customers, namely equipment end users. The gray arrow from these users to the Application layer in the figure indicates a *uses* relationship in the UML (Unified Modeling Language) terminology. Users have access to the MII services in the MII Cloud Services layer through the applications in the Application layer.

4.5 Case Study: SoA Editor

This section explains the SoA editor and its components, also, how they all fit into the IoMT architecture. Figure 4.4 illustrates the SoA editor components on different layers. Conceptually, the editor sits in the Application layer, and it uses services and data on the MII-Cloud services layer. Arrows indicate this relationship in Figure 4.4.

The SoA editor is supposed to be used by different users of SoA data. These are specifically CLs and ABs that can interchange SoA-related data through the infrastructure provided by MII components and the medium supplied by the editor. Since the data in the cloud are stored based on specific formats and standards, third-party application developers can develop other applications conforming to the provided interfaces. An example application that uses SoA data in MII format is the Qualer

Search Engine [127] that is shown as *CL Search Engine* in the Application layer in Figure 4.4. The engine aims to provide search capability for accredited CLs around the world. The engine works on the SoA repository developed by the MII group and for the time being works for the calibration entities in North America. The tool provides search criteria based on several parameters such as location, lab capability, and measured quantities. In the rest of the section, we will explain the traditional accreditation scenario and an alternative scenario that uses the SoA editor. Also, the details about the main MII software elements that give power to the SoA editor are explained.

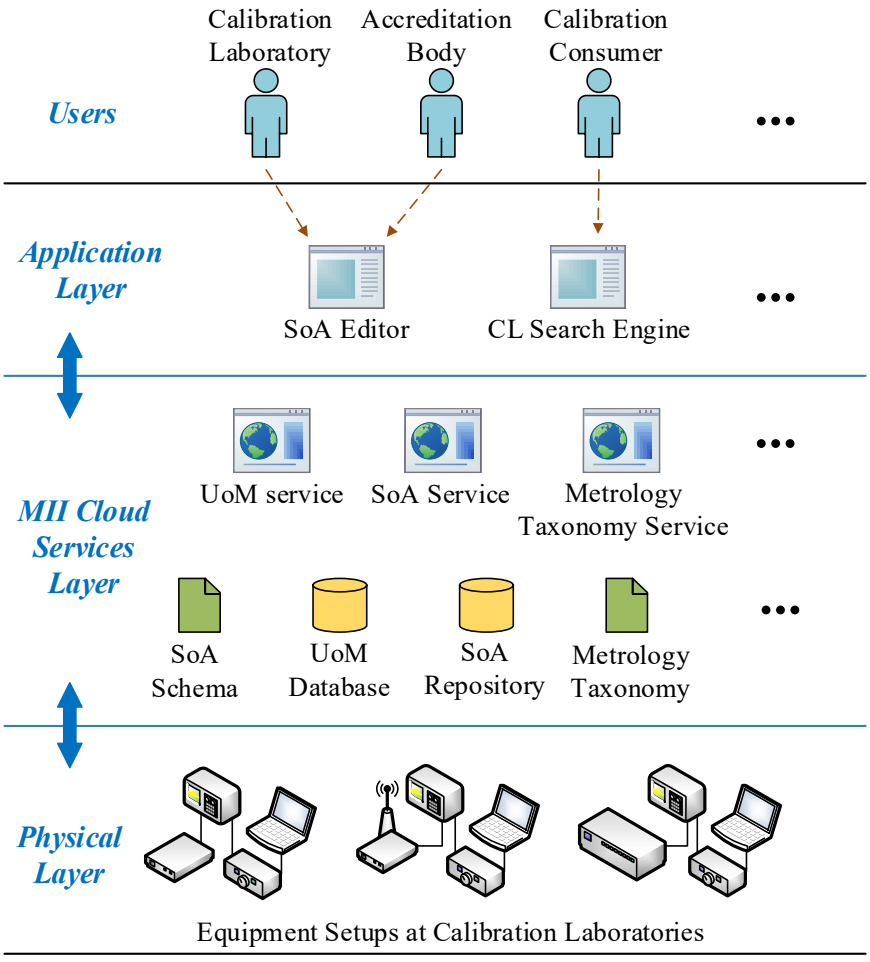

Figure 4.4: The SoA editor and the IoMT architecture.

4.5.1 Traditional vs MII-Aware Calibration Laboratory Accreditation

In this section, we present two scenarios for the accreditation process: The traditional scenario, and an MII-aware scenario. Figure 4.5 illustrates these scenarios. We first consider the traditional workflow of accreditation for CLs. The CL first prepares documents representing SoA based on its calibration parameters and uncertainties. These documents are then passed to an AB for assessment. The AB checks the scope, and after a detailed review, makes the decision. If it is approved, the lab scope is published on the AB's website in a known data format. The accreditation process is composed of a detailed and complex set of activities and interactions between the CL and the AB. It involves an intensive exchange of questions, answers, and modifications to the scope. Customers can access a CL SoA through the documents published on the AB website. The left-hand side of Figure 4.5 summarizes the process.

We now consider an alternative scenario for the accreditation process using MII technologies. CL creates SoA using an editor that stores the data in a standard format. The tool then exports an SoA document conforming to the AB requirements. Then, the document is passed to the AB over the Internet. Similar to the traditional scenario, the AB performs the review and makes a decision. If the AB approves the scope, it publishes the scope using a standard AB style sheet on its website. Along the process of accreditation, data between the CL and the AB goes back and forth in a standard way using the SoA editor. Because of the data format standardization, other tools can be developed to use SoA data and provide several features to calibration customers. The right-hand side of Figure 4.5 summarizes the process.

4.5.2 SoA Schema

SoA documents are written for human readers, and typically they are not in a machine-readable format. To read these documents and interpret the information contained in the CMC in them, one needs to have in-depth technical knowledge in Metrology. An SoA data format was introduced by David Zajac [128] in 2016. In this pioneering work, he presents an XML (Extensible Markup Language) schema that helps flexibly formulate every SoA parameter into the schema. He described his initial ideas on

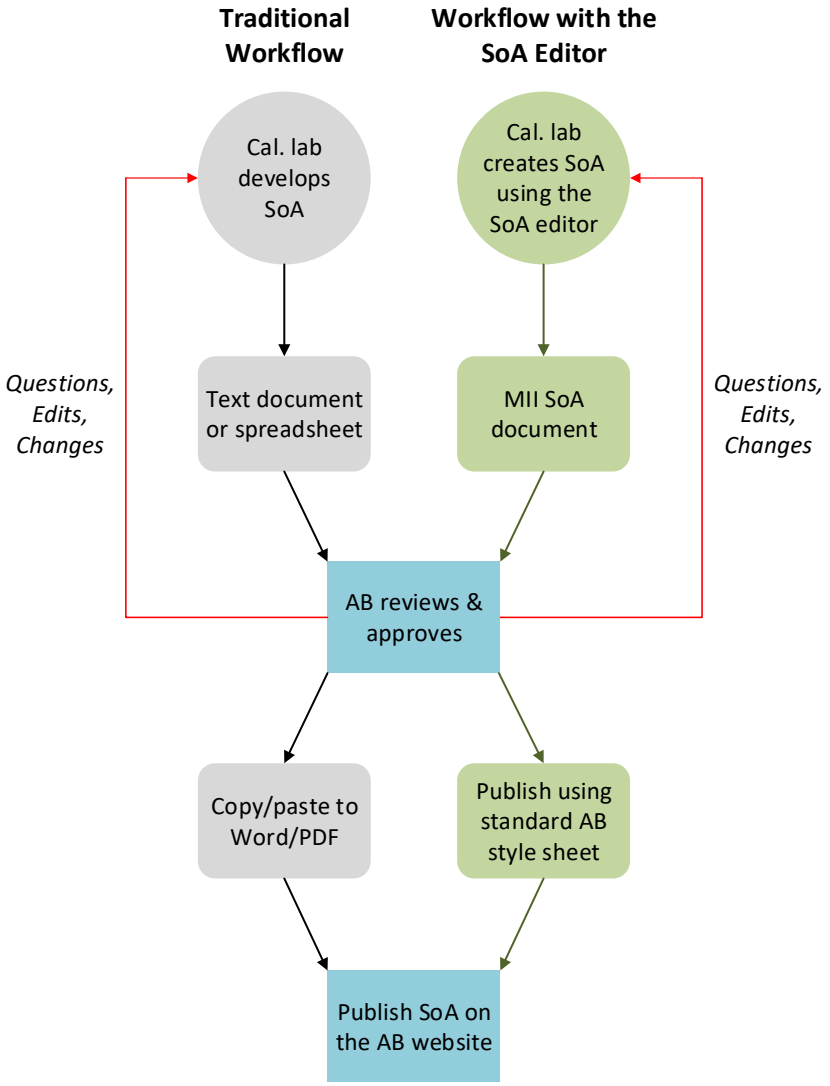

Figure 4.5: Traditional accreditation workflow vs workflow with the SoA editor.

how the schema should be designed and presented its overall functionality. An open-source API (Application Programming Interface) was also developed that allows for SoA data manipulation and calculations based on the given schema.

4.5.3 SoA Repository

The MII group is developing a cloud-hosted repository for SoA data [129]. The repository allows for validation, access, and storing data based on the presented SoA schema at [128] and other standards. Accordingly, the repository provides services offered by the MII community, such as units of measure (UoM) and metrology taxonomy. The repository presently includes several hundred thousands of CMCs from hundreds of calibration entities in the database. The data are mainly coming from North American accredited labs.

4.5.4 Units of Measure Database

A UoM database is being developed by the MII group. The aim is to provide the metrology application developers with a service that gives access to UoM data in a uniform way. An editor for the database was also developed to allow the metrology community to edit and expand the database [130]. The UoM editor takes advantage of the MathML (Mathematical Markup Language) [131] to provide a rendered presentation for each UoM.

One reason for developing such a uniform database is to resolve possible ambiguities. An example of ambiguity for UoM is the *fpm* case which can be interpreted in two ways: *Feet Per Minute* or *Flashes Per Minute*. MII presents the measurement quantity notion, which is paired with all measurement values and allows for the definition of UoMs in an unambiguous way. In this example, we would have *flash-rate* and *speed* quantities. The MII UoM also defines a *singular base UoM*, *convertable to UoM alternatives* and aliases for all UoMs. For the mentioned example, the singular base units would be *flashes-per-second* and *meters-per-second*, the convertable-to alternatives would be *flashes-per-minute* and *feet-per-minute* respectively and aliases could be *fpm* for both, since the quantities are different already.

4.5.5 Measurement Taxonomy

The measurement taxonomy aims to provide a set of naming conventions and a hierarchical data structure to encompass all types of measurements that can be taken. The MII group wants to specify unique types that can clearly distinguish every sort of measurement. The taxonomy of measurements helps to index, catalog, and easily share measurement related data. Based on the proposed convention, a measurement type starts with *source* or *measure* and progresses from general to more specific sub-categories. The taxonomy definition also includes extra information about measurement types, such as the specific required and optional input parameters used in that measurement.

Figure 4.6 shows an example of measurement taxonomy in a tree view. Starting from the root, it goes down to the leaves from general to more specific categories. For example, if the measurement is about the AC voltage, the measurement type can be formulated as "Measure.Volts.AC". The number of sub-categories for the measurement can vary based on its type. Each leaf on the taxonomy tree indicates the specific required and optional input parameters. For example: "Source.Volts.DC" would require Volts.

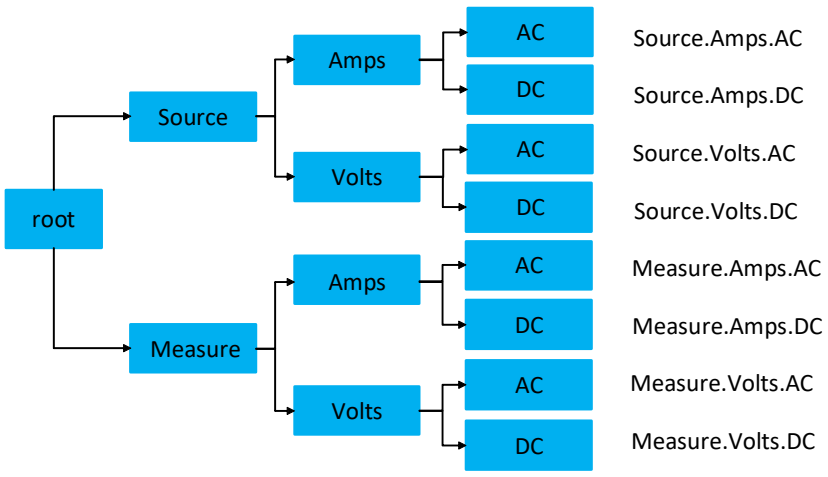

Figure 4.6: An example measurement taxonomy.

4.6 SoA Editor Design and Implementation

This section provides details of the cloud implementation of the SoA editor. Figure 4.7 shows the overall architecture of the system. This architecture contains the *core subsystems* defined in the Azure IoT reference architecture [132]. In our solution, calibration equipment setups are IoT devices. During a calibration process, usually, the device under test (DUT) and the reference device are connected to a PC (Personel Computer). This PC can connect to the Internet to send messages to the cloud. This part of the implementation composes of the physical layer of the IoMT architecture.

The IoT Hub is the cloud gateway in IoT applications. It allows scalable and secure two-way communication between IoT devices and the cloud. The data collected by IoT Hub is transferred to another service on the cloud side, such as Stream Analytics or Azure Functions. IoT Hub can tolerate the management of thousands of IoT devices. However, in the SoA editor scenario, scalability is not a critical issue because the number of calibration equipment is not as large as other IoT applications, such as wireless sensor networks. The number of devices may reach thousands when the usage of the application becomes widespread, e.g., comprising all CLs in a country.

In the IoMT architecture, devices in a calibration setup correspond to IoT devices in a typical IoT solution. However, UUT and the reference device do not have to be IP-capable. Generally, they are connected to a PC which is IP-capable and can connect to the cloud. This scenario fits well with the Field Gateway concept presented in [132]. A Field Gateway is used to connect IoT devices to the cloud gateway. This pattern is shown in Figure 4.8. Because it allows local processing, filtering, or data aggregation, using a field gateway reduces the data transferred to the cloud. In our scenario, the field gateway is a suitable place to do uncertainty calculations. To create an SoA document for a CL, we do not need all measurement data produced in a calibration setup. Uncertainty calculations involve statistical analysis and comparisons against universal standards after collecting a set of calibration data. These calculations can be done locally, and only the resulting value is sent to the cloud to reduce the workload on the cloud side. Our experience with the Metrology.NET platform helps us establishing this kind of setup for automating the calculations in the edge and transferring required data to the cloud.

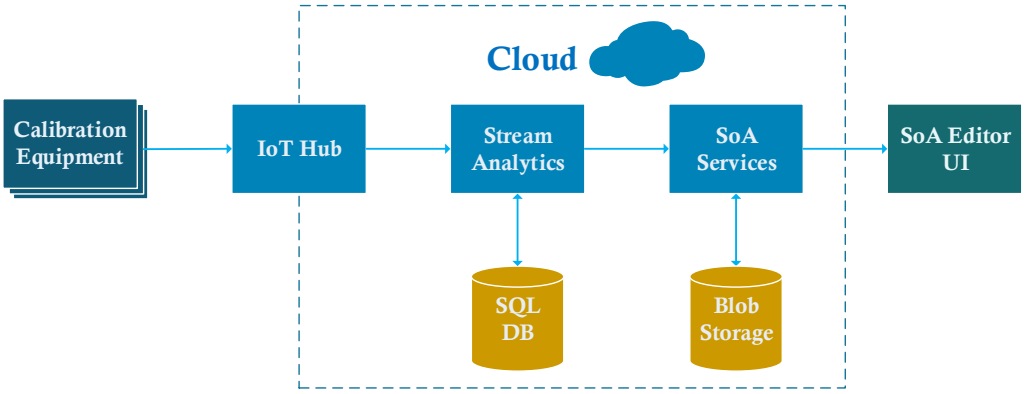

Figure 4.7: Overall system architecture.

The job of stream analytics is to pre-process the data collected by IoT Hub. It allows filtering the data based on queries and directing data to other services, such as a database or another cloud service. It also allows for creating user-defined functions to let more complex jobs. As an alternative to the edge computing in the SoA editor scenario, uncertainty calculations to define CMCs in the SoA document can be done in a stream analytics job. Then these values are forwarded to the SoA service. The SoA Service saves SoA data in a database in XML format.

SoA related data and other required databases, UoM Database and Metrology Taxonomy, as shown in Figure 4.4, are kept as XML files, as MII suggests. The SoA service uses these files to store the data of a CL, adding parameters to the company's calibration capabilities, etc. Blob Storage is a convenient environment to store these XML files [133], considering the amount of storage needed for the SoA editor case study. For another scenario that large-scale data is collected, Azure Data Lake is the alternative to be used. Azure Data Lake allows storing large amounts of relational and nonrelational data in a distributed manner [132]. It also allows big data analytics that will be beneficial, especially for device producers who want to analyze their device's calibration performance.

Besides the data directed to the SoA service, a stream analytics job can directly save some amount of data to the storage. For example, all of the calibration data of a specific device may have great importance for its producer. Even though this data is irrelevant for the SoA document itself, it can be significant for other services in

the IoMT concept. An SQL database can be employed for this purpose, as shown in Figure 4.7.

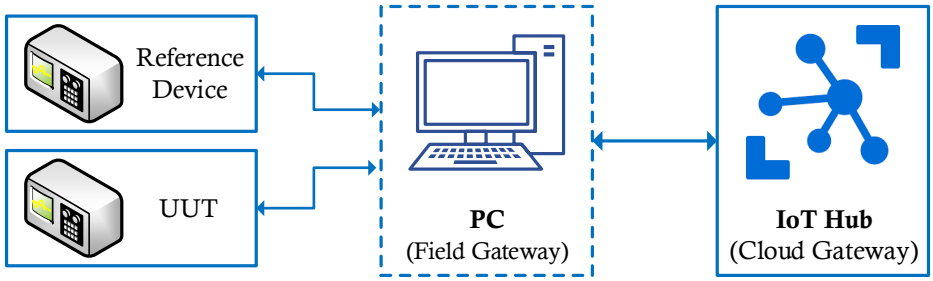

Figure 4.8: A simple calibration setup connected to the cloud through a field gateway.

4.6.1 SoA Editor as an Azure Cloud Service

The SoA editor was implemented as a desktop application in [101]. To provide a clear separation between the presentation layer (front-end) and business logic (back-end), the design of the editor follows the Model-View-ViewModel (MVVM) architectural design pattern [134]. The design pattern allows for the concurrent development of the system components. Also, when you need to change the model, there is no need to update the view, and vice versa. The UI (User Interface) of the editor was developed using the Windows Presentation Foundation (WPF) of Microsoft [135].

Figure 4.9 shows a screenshot from the desktop version of the SoA editor. The screen shows different elements that are used in adding a new measurement parameter to a CL scope. Parameters are added based on the predefined measurement taxonomy explained in Section 4.5.5.

The desktop application is converted to a cloud service to realize the IoMT architecture. Therefore, the editor can work in harmony with other cloud services. The editor's code run as a service on the cloud is presented in Figure 4.7 as *SoA Services*. To create the cloud service, the *WPF application* created before is converted to an *Azure Cloud Service* in Microsoft Visual Studio (VS). Both the applications are implemented with C#. The layered implementation of the desktop application enables the conversion to take place easily. The MVVM pattern is converted to ASP.NET

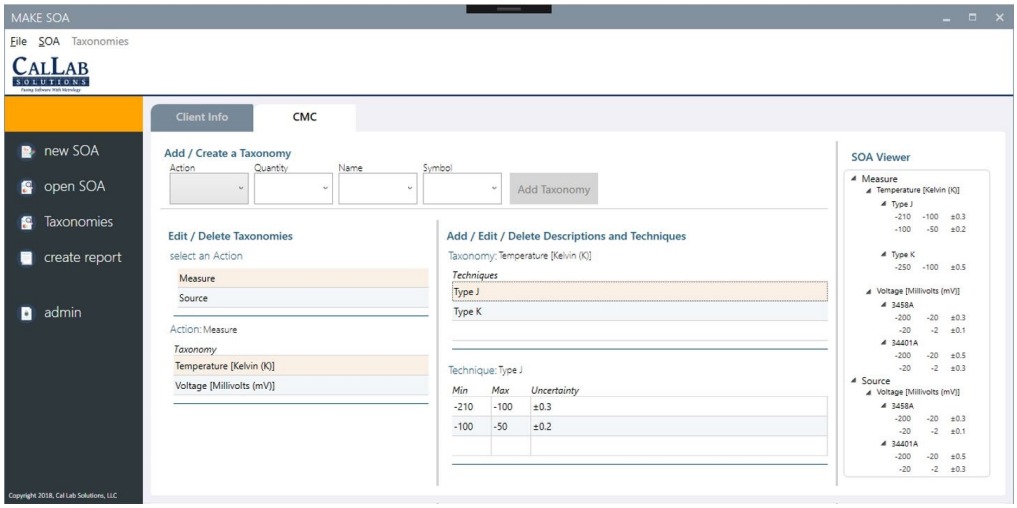

Figure 4.9: A screenshot from the SoA editor desktop application.

Model-View-Controller (MVC) [136] pattern for the cloud application, since it will be used as a web application. Model parts of the applications correspond directly. View-Model and Controller parts require slight changes since the connection methods for the UI elements are different. The View layer for the cloud application is re-written with ASP.NET. The following section provides the key details of the MVC implementation.

4.6.2 Model-View-Controller Pattern

The SoA editor components based on the MVC pattern are explained in this section. Figure 4.10 shows the layers. A brief explanation for each layer is provided below.

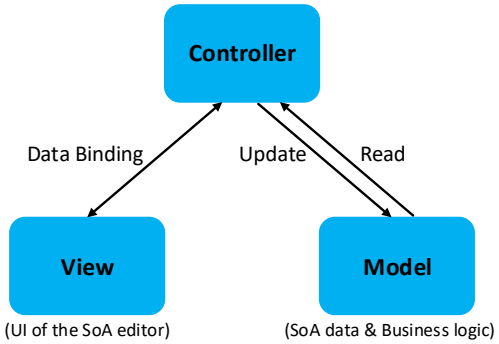

Figure 4.10: The SoA editor design in MVC.

4.6.2.1 Model

The Model layer represents the data and business logic of the application. To be decoupled from the Controller, it exposes the data through observables to that layer. The data, as shown in Figure 4.4, is stored in the MII Cloud Services layer. The Model uses the SoA Schema to handle the read/write of the SoA data, and the Metrology Taxonomy to categorize measurements in a standardized way. The Model also takes advantage of the UoM Database that keeps measurement units in a standard format defined by the MII group.

4.6.2.2 View

The View layer includes all the elements about the UI of the editor. Its role is to subscribe to the Controller observable(s). The UI events are also passed to the Controller, which are then reflected in the database. The UI provided in Figure 4.9 is an example of Views. The Controller layer handles the user inputs provided by the UI and does the necessary operations based on the data standards defined in the Model layer.

4.6.2.3 Controller

The Controller layer behaves as a mediator between the View and the Model layers. It interacts with the Model and supplies observable(s) to the View layer. As an implementation strategy for this layer, the Controller is decoupled from the View, which allows for an interchangeable View component for the editor. It also sends updates made in the View layer back to the Model layer.

4.6.3 Deployment to Azure

The account used for the Azure portal can be the same Microsoft account that is used in VS. In this way, deployment of the locally created service to the Azure can be done easily. The required resources to run the service on cloud (namely, Cloud Service, Storage Account, and Application Insights) can be created simply using a configura-

tion window of VS, and deployment is done in this way. After deployment, Azure provides a link to access the editor's UI through the created cloud service. Other services that are depicted in Figure 4.7 (IoT Hub, Stream Analytics, SQL Database, and Blob Storage) are created by Azure Portal, and the required connections are performed among the services to yield a running application. The resources and services used in the SoA editor case study comply with the lowest pricing policy of the Azure. Figure 4.11 shows the resources created on the Azure portal.

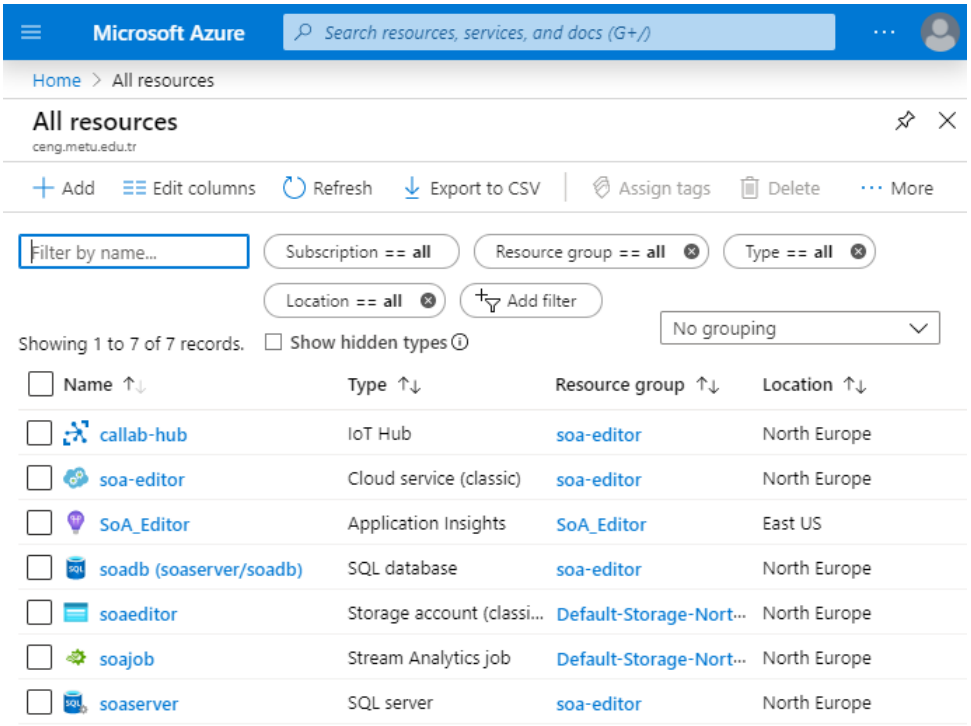

Figure 4.11: Allocated resources and services on the Azure portal.

4.7 Implementation Alternatives

This section includes implementation alternatives for the IoMT architecture realization. We briefly explain options for the cloud provider and an alternative application, a test point editor.

4.7.1 Cloud Providers

Although we use Azure in our implementation, other cloud services could be used as well. For example, AWS (Amazon Web Services) [137], Google Cloud [138], and IBM Cloud [139] are other widely used cloud computing platforms.

As an alternative to the Azure application, we explain which AWS services can be used for the SoA editor example. AWS IoT Core is the cloud application that allows communication and management of IoT devices. IoT Core can be used to collect data from CLs and transfer it to the cloud for further processing. AWS IoT Analytics can be used to process collected data, e.g., for filtering or uncertainty calculations in our SoA editor scenario. Amazon Relational Database Service (Amazon RDS) is an option to store calibration data on the cloud. AWS S3 storage can store taxonomies and other XML files that SoA Service uses and produces. SoA service can be implemented on AWS Lightsail that allows creating simple web applications and websites.

4.7.2 Test Point Editor

IoMT architecture covers various applications and services other than that are used for the SoA editor scenario. A *test point* editor implementation can be another example application for the realization of IoMT, which includes other stakeholders in the industry, namely equipment manufacturers.

Equipment manufacturers provide test points in the manuals that are delivered with the equipment. Calibrators use this device-specific data during the calibration process. These manuals are for humans, and they are not machine-readable. Because there is no standard data format, manufacturers use their way to prepare these manu-

als. This lack of standardization may cause a vast amount of diversity.

During a manual calibration, CL technicians read and follow calibration manuals. Alternatively, CL can use off-the-shelf software for automation or create its own automation software. For the off-the-shelf software case, CL does not have to deal with test point data as the off-the-shelf software already contains it. When CL uses its own software, it has to translate the human-readable test point data to a machine-readable format. This translation is cumbersome as it requires excessive time and effort.

IoMT architecture may help to handle test point data in a better way. Manufacturers would use a test point editor to create test points. Produced test point data would be stored in the cloud to allow the usage of manufacturers and other stakeholders. Services that would be used by the test point editor are *test point service* and *metrology taxonomy service*. CLs are the consumer of the test points as they use this data during the calibration process. Therefore, CLs would use the test point editor with a different UI or role. It is also possible that other applications in the application layer use the test point service as test points are central to calibration. Calibration automation software might be an example of such applications. Therefore, standardization helps different applications to use the same data as storing and accessing it are handled by a common service. Same as the SoA editor scenario, the physical layer of the test point editor example composed of CL equipment.

4.8 Conclusions

We demonstrated the modeling of an SoA editor as a proof of concept towards verification of the IoMT architecture. We proposed the IoMT, an IoT-based architectural framework for the calibration industry that is inspired by the MII works. Then, we present the realization of the framework by implementing the SoA editor as a cloud application. Also, the necessary steps to have a full-scale IIoT scenario by combining physical equipment and cloud services are explained. Microsoft Azure is chosen as the cloud environment of the application. We believe the Metrology community can benefit from the IoMT concept by complying with specific data standards and employing more automation in calibration processes.

SoA encompasses the statement of a CL's capabilities. It can be of great importance both to CLs and customers who want their equipment calibrated. ABs certify a CL's capabilities and thus the services offered by them. There were attempts to digitalize the traditional accreditation process. The MII initiative proposes new standards and software tools and components to help digitalize this process. The SoA editor can be classified in this kind of attempt. The traditional accreditation process involves intense data exchange, which is paper-based, thus, cumbersome. The editor facilitates this data exchange among stakeholders. The SoA editor is an open-source project hosted at Github [140]. Therefore, the project is open to contributions from the metrology community.

Bringing automation to the accreditation process also helps with the traceability. Along the process, there may be a need for a high degree of interaction and data exchange between a CL and AB. These may include several rounds of analyses done by an AB on the calibration scope provided by a CL. This also can include some minor and major editions required by an AB to be fulfilled by a CL. Considering all this communication done in an automated manner, the software can keep track of all the steps along the whole process and thus improve the process traceability.

Other than traceability, there are different non-functional concerns to be considered for the IoMT architecture. Some of them are confidentiality, reliability, governance/-community process, and usability. Although some of these concerns are handled by the cloud provider, they need to be investigated and addressed appropriately for the architecture to gain acceptance.

CHAPTER 5

CONCLUSIONS

This thesis includes efforts to provide interoperability for systems that have heterogeneous components by means of configurable connectors. The foundations of this idea are introduced, and as an application of the proposed methodology, a configurable gateway between DDS (Data Distribution Service for Real-Time Systems) and HLA (High-Level Architecture) for LVC (Live-Virtual-Constructive) simulation systems domain is presented. Moreover, for Metrology and the calibration industry, a layered architectural solution based on the IIoT (Industrial Internet of Things) is proposed, and necessary steps to take to develop applications conforming to this architecture are explained. Industrial and academic studies for proof of concept support the idea of applicability of proposed methodologies. The rest of this chapter includes brief summaries of each chapter, remarks, and possible future work.

5.1 Summary

In Chapter 2, a set of connectors, described as *off-the-shelf connectors*, are introduced to a component-based development environment where a variability model drives the configuration mechanisms in the overall flow of the application, the components, and connectors. The proposed solution includes using connectors in a new system by selection and configuration. Solving communication heterogeneity problems by adapting connectors may avoid additional functional load on the domain components.

In Chapter 3, a configurable DDS-HLA gateway solution is elaborated for the LVC simulation domain. The gateway is capable of providing two-way data transfer between DDS and HLA. The design of the gateway adheres to the idea of configurable

connectors, which allow users to generate a customized gateway. The gateway is capable of converting primitive and structured data-types between DDS and HLA. These conversions are specified by users resulting in different configurations of the gateway. The gateway can also be adapted to another configuration at runtime.

In Chapter 4, the Internet of Measurement Things (IoMT) architecture is presented. This architecture involves a layered model, inspired by the IIoT architectures, with a cloud-centric middle layer that solves interoperability issues of different applications. The applicability of the IoMT architecture in the calibration industry is shown through an editor application for Scope of Accreditation.

5.2 Remarks

The DDS-HLA gateway application presented in Chapter 3 validates the configurable connectors paradigm provided in Chapter 2 to some extend. To fully validate the proposed approach, ideally, there should be a mature domain, and gateway configurations should be derived from totally existing implementations. Another shortcoming is the gateway being implemented as a local application instead of a web service that should be made available through the Internet. The current platform choice decreases the ability to serve a broader community and collect data about how effective is the reuse of existing implementations. Moreover, the IIoT architecture presented in Chapter 4 does not have an implementation of configurable connectors. Instead, it is aimed to cover the interoperability in other points of view.

For the sake of simplicity, deploying one connector between two components in our Component-Oriented Software Engineering development paradigm is one of the design decisions. Accordingly, descriptions and implementations presented for off-the-shelf connectors in Chapter 2 and the DDS-HLA gateway in Chapter 3 follows this principle. Therefore, the gateway is capable of handling the communication of two different architectures, not more. However, from an architectural perspective, it may be possible for the gateway to be deployed to provide interoperability for heterogeneous systems that contain more than two subsystems that each has a different architecture. Also, the multi-threaded application of the gateway may allow this improve-

ment by employing more threads to communicate with the newly added architecture.

The gateway is not focused on non-functional interoperability, e.g., Quality of Service (QoS) parameters of the DDS-based system. Furthermore, a gateway may solve semantic interoperability issues as well, which is not the case for our current version. In a nutshell, the current version of the DDS-HLA gateway provides functional and syntactic interoperability for binary components that themselves are independent systems in our application.

Performance analysis of the gateway application is in a preliminary stage. The gateway achieves interoperability; performance is also a critical factor for the gateway to be employed as a solution, however. Because our first objective is interoperability, we did not spare enough time to increase the gateway performance. However, in Chapter 3, we provide results for the gateway performance in terms of network delay. In this stage, we observe that the performance is highly dependent on the configurations of DDS-based and HLA-based systems. In two different settings, network performance changed in favor of one side to another: While in the industrial application, network delays were less in the DDS to HLA direction, whereas in the academic tests, HLA to DDS performance was better.

5.3 Future Work

The future work for the configurable connectors is planned to offer a toolset over the Internet for supporting wider usage. Both the approach and various domain models are expected to be improved in an open environment. The tools should allow the formation of components and connectors for example, defining a domain for software development in new application fields.

For the DDS-HLA gateway, the required QoS parameters for LVC interoperability will be investigated in the future. HLA's QoS properties are limited compared to DDS. The effects of this shortcoming on the interoperability with DDS should be examined. Moreover, how much of this issue can be handled via the gateway approach is an open problem. Also, generating gateway code using configured variability models is considered as future work. It requires extending configuration options for the

gateway, besides configuring the gateway for data interoperability. Configuration of the system at a higher abstraction level, such as DDS topic and HLA FOM (Federation Object Model), can be considered.

The future work of the IoMT (Internet of Measurement Things) may contain adapting more applications to the architecture. Although there are already implemented applications in the application layer of the IoMT architecture and further, they are locally in use by some calibration laboratories, they are restricted in terms of data sharing with the metrology community. In the future, these applications should be encouraged to adapt themselves to IoMT architecture and provide service to other stakeholders and calibration laboratories. To ensure that more applications are served on IoMT, more common services are likely to be shared by many applications that will be developed and deployed.

REFERENCES

[1] S. Suloglu, M. C. Kaya, A. Cetinkaya, A. Karamanlioglu, and A. H. Dogru, *Cloud-Enabled Domain-Based Software Development*, pp. 109–130. Cham: Springer International Publishing, 2020.

[2] V. Issarny and A. Bennaceur, *Composing Distributed Systems: Overcoming the Interoperability Challenge*, pp. 168–196. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013.

[3] M. C. Kaya, M. Saeedi Nikoo, S. Suloglu, B. Tekinerdogan, and A. H. Dogru, *Managing Heterogeneous Communication Challenges in the Internet of Things Using Connector Variability*, pp. 127–149. Cham: Springer International Publishing, 2017.

[4] S. Bandyopadhyay and A. Bhattacharyya, "Lightweight internet protocols for web enablement of sensors using constrained gateway devices," in *2013 International Conference on Computing, Networking and Communications (ICNC)*, pp. 334–340, 2013.

[5] P. Desai, A. Sheth, and P. Anantharam, "Semantic gateway as a service architecture for iot interoperability," in *2015 IEEE International Conference on Mobile Services*, pp. 313–319, 2015.

[6] E. Palomar, X. Chen, Z. Liu, S. Maharjan, and J. Bowen, "Component-based modelling for scalable smart city systems interoperability: A case study on integrating energy demand response systems," *Sensors*, vol. 16, no. 11, 2016.

[7] D. Garlan, "Higher order connectors," in *Workshop on Compositional Software Architectures*, (Monterey, CA), 1998.

[8] R. Spalazzese and P. Inverardi, "Mediating connector patterns for components interoperability," in *European Conference on Software Architecture*, pp. 335–343, Springer, 2010.

[9] N. Nostro, R. Spalazzese, F. D. Giandomenico, and P. Inverardi, "Achieving functional and non functional interoperability through synthesized connectors," *Journal of Systems and Software*, vol. 111, pp. 185 – 199, 2016.

[10] M. C. Kaya, A. Cetinkaya, and A. H. Dogru, "Off-the-shelf connectors for interdisciplinary components," *Journal of Integrated Design and Process Science*, vol. 22, no. 3, pp. 35–53, 2018.

[11] M. C. Kaya, A. Karamanlioglu, İ. Ç. Çetintaş, E. Çilden, H. Canberi, and H. Oğuztüzün, "A configurable gateway for dds-hla interoperability," in *Proceedings of the 2019 Summer Simulation Conference*, p. 36, Society for Computer Simulation International, 2019.

[12] M. C. Kaya, M. Saeedi Nikoo, M. L. Schwartz, and H. Oguztuzun, "Internet of measurement things architecture: Proof of concept with scope of accreditation," *Sensors*, vol. 20, p. 503, Jan 2020.

[13] T. Erl, *Service-oriented architecture: concepts, technology, and design*. Prentice Hall PTR, Upper Saddle River, 2005.

[14] K. Czarnecki, "Generative programming: Methods, techniques, and applications tutorial abstract," in *International Conference on Software Reuse*, pp. 351–352, Springer, 2002.

[15] D. C. Schmidt, "Model-driven engineering," *COMPUTER-IEEE COMPUTER SOCIETY-*, vol. 39, no. 2, p. 25, 2006.

[16] A. Ertas, M. M. Tanik, and T. Maxwell, "Transdisciplinary engineering education and research model," *Journal of Integrated Design and Process Science*, vol. 4, no. 4, pp. 1–11, 2000.

[17] M. M. Tanik and A. Ertas, "Interdisciplinary design and process science: A discourse on scientific method for the integration age," *Journal of Integrated Design and Process Science*, vol. 1, no. 1, pp. 76–94, 1997.

[18] C. Szyperski, D. Gruntz, and S. Murer, *Component software: beyond object-oriented programming*. Pearson Education, 2002.

[19] Object Management Group, "Uml 2.0." http://www.omg.org/spec/UML/2.0/, 2020. Accessed Feb. 01, 2020.

[20] N. R. Mehta, N. Medvidovic, and S. Phadke, "Towards a taxonomy of software connectors," in *Proceedings of the 22nd international conference on Software engineering*, pp. 178–187, 2000.

[21] A. Dogru, "Toward a component-oriented methodology to build-by-integration," *Development of component-based information systems*, pp. 49–69, 2005.

[22] A. Cetinkaya, M. C. Kaya, and A. H. Dogru, "Component integration through connector supported process models," in *Proceedings of the SDPS 22nd international conference on emerging trends and technologies in convergence solutions*, (Birmingham, AL), pp. 31–37, 2017.

[23] M. Ç. Kaya, "Modeling variability in component oriented software engineering," Master's thesis, MSc Thesis, Middle East Techical University, 2015.

[24] A. H. Dogru, "Component oriented software engineering modeling language: Coseml," *Computer Engineering Department, Middle East Technical University, Turkey, TR*, pp. 99–3, 1999.

[25] M. M. Tanik and E. S. Chan, *Fundamentals of computing for software engineers*. Van Nostrand Reinhold Co., 1991.

[26] A. H. Dogru and M. M. Tanik, "A process model for component-oriented software engineering," *IEEE software*, vol. 20, no. 2, pp. 34–41, 2003.

[27] E. K. Akbıyık, "Service oriented system design through process decomposition," Master's thesis, 2008.

[28] M. C. Kaya, S. Suloglu, and A. H. Dogru, "Variability modeling in component oriented system engineering," in *Proceedings of the 19th international conference on transformative science and engineering, business and social innovation*, (Kuching, Sarawak, Malaysia), pp. 251–259, 2014.

[29] M. C. Kaya, A. Cetinkaya, and A. H. Dogru, "Composition capability of component-oriented development," in *5th International Symposium on Inno-*

vative Technologies in Engineering and Science (ISITES), (Baku, Azerbaijan), pp. 1299, 1307, 2017.

[30] A. Çetinkaya, "Variable connectors in component oriented development," Master's thesis, MSc Thesis, Middle East Techical University, 2017.

[31] S. Süloğlu, *Model-driven variability management in choreography specification*. PhD thesis, PhD Thesis, Middle East Techical University, 2013.

[32] F. Arbab and F. Arbab, "A channel-based coordination model for component composition," in *Mathematical Structures in Computer Science*, Citeseer, 2002.

[33] Reo, "Reo coordination language." `http://reo.project.cwi.nl`, 2020. Accessed Feb. 01, 2020.

[34] D. Gelernter and A. J. Bernstein, "Distributed communication via global buffer," in *Proceedings of the first ACM SIGACT-SIGOPS symposium on Principles of distributed computing*, pp. 10–18, 1982.

[35] Orc, "Orc coordination lanugage." `http://orc.csres.utexas.edu/index.shtml`, 2020. Accessed Feb. 01, 2020.

[36] Y. Zhao, D. Ma, M. Liu, and C. Hu, "Coordination behavioral structure: a web services coordination model in dynamic environment," in *Seventh IEEE/ACIS International Conference on Computer and Information Science (icis 2008)*, pp. 611–617, IEEE, 2008.

[37] M. P. Papazoglou, "Service-oriented computing: Concepts, characteristics and directions," in *Proceedings of the Fourth International Conference on Web Information Systems Engineering, 2003. WISE 2003.*, pp. 3–12, IEEE, 2003.

[38] F. Cabrera, G. Copeland, T. Freund, J. Klein, D. Langworthy, D. Orchard, J. Shewchuk, and T. Storey, "Web services coordination (ws-coordination)," *joint specification by BEA, IBM, and Microsoft*, 2002.

[39] L. F. Cabrera, G. Copeland, M. Feingold, R. W. Freund, T. Freund, J. Johnson, S. Joyce, C. Kaler, J. Klein, D. Langworthy, *et al.*, "Web services atomic transaction (ws-atomictransaction)," *MSDN Library (November 2004)*, 2005.

[40] L. F. Cabrera, G. Copeland, M. Feingold, R. W. Freund, T. Freund, S. Joyce, J. Klein, D. Langworthy, M. Little, F. Leymann, *et al.*, "Web services business activity framework (ws-businessactivity)," *IBM Web Service Transactions Specifications*, 2005.

[41] A. Cetinkaya, M. C. Kaya, and A. H. Dogru, "Enhancing xcoseml with connector variability for component-oriented development," in *Proceedings of the SDPS 21st international conference on emerging trends and technologies in designing healthcare systems*, (Orlando, FL), pp. 120–125, 2016.

[42] L. K. Jololian and M. M. Tanik, "A framework for a meta-semantic language for smart component-adapters," *Journal of Systems Integration*, vol. 10, no. 3, pp. 269–297, 2001.

[43] L. Jololian, "Towards semantic integration of components using a service-based architecture," *Journal of Integrated Design and Process Science*, vol. 9, no. 3, pp. 1–13, 2005.

[44] A. Costin and C. Eastman, "Need for interoperability to enable seamless information exchanges in smart and sustainable urban systems," *Journal of Computing in Civil Engineering*, vol. 33, no. 3, p. 04019008, 2019.

[45] C. Emmer, T. M. Hofmann, T. Schmied, J. Stjepandić, and M. Strietzel, "A neutral approach for interoperability in the field of 3d measurement data management," *Journal of Industrial Information Integration*, vol. 12, pp. 47–56, 2018.

[46] E. M. Dashofy, N. Medvidovic, and R. N. Taylor, "Using off-the-shelf middleware to implement connectors in distributed software architectures," in *Proceedings of the 1999 International Conference on Software Engineering (IEEE Cat. No. 99CB37002)*, pp. 3–12, IEEE, 1999.

[47] N. Medvidovic, E. M. Dashofy, and R. N. Taylor, "Employing off-the-shelf connector technologies in c2-style architectures," in *Proceedings of the California Software Symposium*, 1998.

[48] Xillio. `https://www.xillio.com/`, 2020. Accessed Feb. 01, 2020.

[49] J. Pérez, I. Ramos, and J. Á. Carsí, "Taking advantage of cots for developing aspect-oriented software architectures," in *15th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ecbs 2008)*, pp. 245–254, IEEE, 2008.

[50] P. Inverardi, R. Spalazzese, and M. Tivoli, "Application-layer connector synthesis," in *International School on Formal Methods for the Design of Computer, Communication and Software Systems*, pp. 148–190, Springer, 2011.

[51] N. Nostro, R. Spalazzese, F. Di Giandomenico, and P. Inverardi, "Achieving functional and non functional interoperability through synthesized connectors," *Journal of Systems and Software*, vol. 111, pp. 185–199, 2016.

[52] C. Cetina, P. Giner, J. Fons, and V. Pelechano, "Autonomic computing through reuse of variability models at runtime: The case of smart homes," *Computer*, vol. 42, no. 10, pp. 37–43, 2009.

[53] S. Di Cola, K.-K. Lau, C. Tran, and C. Qian, "Towards defining families of systems in iot: Logical architectures with variation points," in *International Internet of Things Summit*, pp. 419–426, Springer, 2015.

[54] N. He, D. Kroening, T. Wahl, K.-K. Lau, F. Taweel, C. Tran, P. Rümmer, and S. S. Sharma, "Component-based Design and Verification in X-MAN," in *Embedded Real Time Software and Systems (ERTS2012)*, (Toulouse, France), Feb. 2012.

[55] S. Suloglu, M. C. Kaya, A. Karamanlioglu, S. Entekhabi, M. Saeedi Nikoo, B. Tekinerdogan, and A. H. Dogru, "Comparative analysis of variability modelling approaches in component models," *IET Software*, vol. 12, no. 6, pp. 437–445, 2018.

[56] C. Togay, G. Tokdemir, and A. H. Dogru, "Process decomposition using axiomatic design theory," in *Proceedings of the SDPS 22nd international conference on emerging trends and technologies in convergence solutions*, (Birmingham, AL), pp. 1–7, 2017.

[57] K. C. Kang, S. Kim, J. Lee, K. Kim, E. Shin, and M. Huh, "Form: A feature-

oriented reuse method with domain-specific reference architectures," *Annals of software engineering*, vol. 5, no. 1, p. 143, 1998.

[58] K. Pohl, G. Böckle, and F. J. van Der Linden, *Software product line engineering: foundations, principles and techniques*. Springer Science & Business Media, 2005.

[59] A. E. Henninger, D. Cutts, M. Loper, R. Lutz, R. Richbourg, R. Saunders, and S. Swenson, "Live virtual constructive architecture roadmap (lvcar) final report," *Institute for Defense Analysis*, 2008.

[60] J. E. Coolahan and G. W. Allen, "Lvc architecture roadmap implementation-results of the first two years," tech. rep., ARMY PEO (SIMULATION TRAINING AND INSTRUMENTATION) ORLANDO FL JOINT TRAINING INTEGRATION AND EVALUATION CENTER, 2012.

[61] Y. Park and D. Min, "WiP Abstract: Message Bridging Structure Between HLA and DDS for Integrating Cyber-Physical Systems," in *2012 IEEE/ACM Third International Conference on Cyber-Physical Systems*, pp. 210–210, Apr. 2012.

[62] M.-S. Oh, Y.-H. Son, and K.-C. Lee, "An Ontology System for Interoperation between DDS and HLA," *Indian Journal of Science and Technology*, vol. 8, no. 27, 2015.

[63] J. C. Díaz, I. Gálvez, and J. M. López-Rodríguez, "How to develop true distributed simulations? hla & dds interoperability," 2011.

[64] F. Greff, E. Dujardin, A. Samama, Y.-Q. Song, and L. Ciarletta, "A Symbiotic Approach to Designing Cross-Layer QoS in Embedded Real-Time Systems," in *8th European Congress on Embedded Real Time Software and Systems (ERTS 2016)"*, (Toulouse, France), Jan. 2016.

[65] A. Hakiri, P. Berthou, and T. Gayraud, "Design of Low Cost PC-based Simulators for Education and Training Purpose Using DDS," in *International Conference on Computer as a Tool, EUROCON 2011*, p. 103, 2011.

[66] E. Çilden, E. Gültekin, D. Poyraz, and M. H. Canberi, "A generic distributed architecture to integrate simulated participants with modular avionics," in *2018*

IEEE/ACM 22nd International Symposium on Distributed Simulation and Real Time Applications (DS-RT), pp. 1–2, Oct 2018.

[67] Object Management Group, "Data distribution service for real-time systems," *OMG Document, v1.4, formal/15-04-10*, 2015.

[68] A. Corsaro, "The data distribution service tutorial," *Technical Report 4.0*, 2014.

[69] IEEE Standards Association, "1516–2010-ieee standard for modeling and simulation (m&s) high level architecture (hla)," 2010.

[70] O. Topçu and H. Oğuztüzün, *Guide to Distributed Simulation with HLA*. Springer, 2017.

[71] N. Huynh, "State transfer management in adaptive software: An approach from design to runtime," in *2019 IEEE-RIVF International Conference on Computing and Communication Technologies (RIVF)*, pp. 1–6, 2019.

[72] M. Bashari, E. Bagheri, and W. Du, "Dynamic software product line engineering: a reference framework," *International Journal of Software Engineering and Knowledge Engineering*, vol. 27, no. 02, pp. 191–234, 2017.

[73] J. A. Bonache-Seco, J. A. Lopez-Orozco, E. B. Portas, and J. L. Risco Martín, "Adaptive event driven framework for real time multi-agent missions," in *2018 IEEE/ACM 22nd International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*, pp. 1–5, 2018.

[74] W. T. Tsai, Chun Fan, Yinong Chen, and R. Paul, "Ddsos: a dynamic distributed service-oriented simulation framework," in *39th Annual Simulation Symposium (ANSS'06)*, pp. 8 pp.–167, 2006.

[75] S. Adjoyan and A.-D. Seriai, "Reconfigurable service-based architecture based on variability description," in *Proceedings of the Symposium on Applied Computing*, SAC '17, (New York, NY, USA), p. 1154–1161, Association for Computing Machinery, 2017.

[76] M. C. Kaya, A. Eroglu, A. Karamanlioglu, E. Onur, B. Tekinerdogan, and A. H. Dogru, *Runtime Adaptability of Ambient Intelligence Systems Based*

on Component-Oriented Approach, pp. 69–92. Cham: Springer International Publishing, 2019.

[77] K. Lessmann, D. E. Cutts, N. Horner, and D. Drake, "Lvcar enhancements for selecting gateways," in *Proc., 2011 Spring Simulation Interoperability Workshop*, 2011.

[78] M. J. O'Connor, K. Lessmann, and D. Drake, "Lvcar enhancements for using gateways," in *Proc., 2011 Spring Simulation Interoperability Workshop*, 2011.

[79] D. D. Hodson and R. R. Hill, "The art and science of live, virtual, and constructive simulation for test and analysis," *The Journal of Defense Modeling and Simulation*, vol. 11, no. 2, pp. 77–89, 2014.

[80] L. Ardila, I. Pérez-Llopis, M. Esteve, and C. E. Palau, "Interoperable architecture for joint real/virtual training in emergency management using the mpeg-v standard," *Computer Standards & Interfaces*, vol. 41, pp. 39 – 55, 2015.

[81] M. C. Kaya, E. Çilden, H. Canberi, and H. Oğuztüzün, "A literature survey on DDS-HLA interoperability approaches (in Turkish)," in *Proceedings of the Yedinci Ulusal Savunma Uygulamaları Modelleme ve Simülasyon Konferansı*, (Ankara,Turkey), pp. 60–71, 2017.

[82] J. M. Lopez-Rodriguez, R. Martin, and P. Jimenez, "How to develop true distributed real time simulations? mixing ieee hla and omg dds standards," *Nextel Aerospace Defense & Security (NADS), http://simware. es*, 2011.

[83] Y. Park and D. Min, "Development of HLA-DDS wrapper API for network-controllable distributed simulation," in *2013 7th International Conference on Application of Information and Communication Technologies*, pp. 1–5, Oct. 2013.

[84] Y. Park and D. Min, "Publish/Subscribe Mapping between HLA-DDS Middleware for Cyber-Physical System Simulation," *Research Notes in Information Science*, vol. 14, 2013.

[85] H. Pérez and J. J. Gutiérrez, "Handling heterogeneous partitioned systems through arinc-653 and dds," *Computer Standards & Interfaces*, vol. 50, pp. 258 – 268, 2017.

[86] J. Dianes, M. Díaz, and B. Rubio, "Using standards to integrate soft real-time components into dynamic distributed architectures," *Computer Standards & Interfaces*, vol. 34, no. 2, pp. 238 – 262, 2012.

[87] M. Aragão, P. Moreno, and A. Bernardino, "Middleware interoperability for robotics: A ros–yarp framework," *Frontiers in Robotics and AI*, vol. 3, p. 64, 2016.

[88] Portico, "Portico." `https://github.com/openlvc/portico`, 2019. Accessed Mar. 01, 2019.

[89] OpenRTI, "Openrti." `https://sourceforge.net/projects/openrti/`, 2019. Accessed Mar. 01, 2019.

[90] Qt, "Qt." `https://www.qt.io/`, 2019. Accessed Mar. 01, 2019.

[91] OMG, "About the data distribution service specification version 1.4." `https://www.omg.org/spec/DDS/1.4/`, 2019. Accessed Mar. 01, 2019.

[92] ADLINK, "Vortex data distribution service." `https://www.adlinktech.com/en/data-distribution-service.aspx`, 2019. Accessed Mar. 01, 2019.

[93] K. Morse, M. Lightner, R. Little, B. Lutz, and R. Scrudder, "Enabling simulation interoperability," *Computer*, vol. 39, pp. 115–117, Jan. 2006.

[94] G. Pardo-Castellote, B. Farabaugh, and R. Warren, "An Introduction to DDS and Data-Centric Communications," p. 16, 2005.

[95] T. Alexander, R. Sottilare, S. Goldberg, D. Andrews, L. Magee, and J. J. Roessingh, "Enhancing Human Effectiveness through Embedded Virtual Simulation," 2012. Publisher: Unpublished.

[96] M. C. Kaya, S. Suloglu, G. Tokdemir, B. Tekinerdogan, and A. H. Dogru, *Variability incorporated simultaneous decomposition of models under structural and procedural views*, pp. 95–116. Taylor & Francis, 2019.

[97] C. Togay, A. H. Dogru, and J. U. Tanik, "Systematic component-oriented development with axiomatic design," *Journal of Systems and Software*, vol. 81, no. 11, pp. 1803 – 1815, 2008.

[98] D. Beuche, "Modeling and building software product lines with pure:: variants," in *Proceedings of the 16th International Software Product Line Conference-Volume 2*, pp. 255–255, 2012.

[99] H. Lasi, P. Fettke, H.-G. Kemper, T. Feld, and M. Hoffmann, "Industry 4.0," *Business & Information Systems Engineering*, vol. 6, pp. 239–242, Aug 2014.

[100] M. Saeedi Nikoo, M. C. Kaya, M. L. Schwartz, and H. Oguztuzun, *Internet of Measurement Things: Toward an Architectural Framework for the Calibration Industry*, pp. 81–102. Cham: Springer International Publishing, 2019.

[101] M. S. Nikoo, M. C. Kaya, M. L. Schwartz, and H. Oguztuzun, "An mii-aware soa editor for the industrial internet of things," in *2019 II Workshop on Metrology for Industry 4.0 and IoT (MetroInd4.0 IoT)*, pp. 213–218, June 2019.

[102] R. Srujan, M. K.Y., P. M., A. M .S., C. K. N. Guptha, and S. M R, "Design and fabrication of iot enabled temperature sensor calibration experimental setup for metrology lab," in *2018 3rd International Conference on Computational Systems and Information Technology for Sustainable Solutions (CSITSS)*, pp. 333–336, Dec 2018.

[103] R. Benitez, R. Benitez, C. Ramirez, and J. A. Vazquez, "Sensors calibration for metrology 4.0," in *2019 II Workshop on Metrology for Industry 4.0 and IoT (MetroInd4.0 IoT)*, pp. 296–299, June 2019.

[104] I. S. O. Vim, "International vocabulary of Metrology –Basic and general concepts and associated terms (VIM)," *International Organization*, 2012.

[105] "Testing and calibration laboratories." `https://www.iso.org/home/standards/popular-standards/isoiec-17025-testing-and-calibra.html`. Accessed: 2019-10-11.

[106] "MII Initiative Knowledge Base." `http://miiknowledge.wikidot.com/`. Accessed: 2019-10-11.

[107] M. J. Kuster, "Toward a Metrology Information Infrastructure," in *Metrologist NCSLI Worldwide News*, 2013.

[108] M. Kuster, "Toward a Measurement Information Infrastructure: Smart Accreditation Scopes," *Metrologist*, 2015.

[109] "Metrology.NET Calibration Automation Platform." `https://www.metrology.net/`. Accessed: 2019-09-23.

[110] "Cal Lab Solutions Inc.." `http://www.callabsolutions.com/`. Accessed: 2019-09-23.

[111] "Cloud Services Documentation." `https://docs.microsoft.com/en-us/azure/cloud-services/`. Accessed: 2019-10-11.

[112] "Azure IoT Documentation." `https://docs.microsoft.com/en-us/azure/iot-fundamentals/`. Accessed: 2019-10-11.

[113] A. Lazzari, J.-M. Pou, C. Dubois, and L. Leblond, "Smart metrology: the importance of metrology of decisions in the big data era," *IEEE Instrumentation & Measurement Magazine*, vol. 20, no. 6, pp. 22–29, 2017.

[114] P. Daponte, F. Lamonaca, F. Picariello, L. De Vito, G. Mazzilli, and I. Tudosa, "A survey of measurement applications based on iot," in *2018 Workshop on Metrology for Industry 4.0 and IoT*, pp. 1–6, IEEE, 2018.

[115] O. Monnier, "A smarter grid with the internet of things," *Texas Instruments White Paper*, 2013.

[116] L. Angrisani, U. Cesaro, M. D'Arco, D. Grillo, and A. Tocchi, "Iot enabling measurement applications in industry 4.0: Platform for remote programming ates," in *2018 Workshop on Metrology for Industry 4.0 and IoT*, pp. 40–45, IEEE, 2018.

[117] H. Haskamp, F. Orth, J. Wermann, and A. W. Colombo, "Implementing an opc ua interface for legacy plc-based automation systems using the azure cloud: An icps-architecture with a retrofitted rfid system," in *2018 IEEE Industrial Cyber-Physical Systems (ICPS)*, pp. 115–121, May 2018.

[118] H. S. Raju and S. Shenoy, "Real-time remote monitoring and operation of industrial devices using iot and cloud," in *2016 2nd International Conference on Contemporary Computing and Informatics (IC3I)*, pp. 324–329, Dec 2016.

[119] S. Forsström and U. Jennehag, "A performance and cost evaluation of combining opc-ua and microsoft azure iot hub into an industrial internet-of-things system," in *2017 Global Internet of Things Summit (GIoTS)*, pp. 1–6, June 2017.

[120] J. Shi, L. Jin, and J. Li, "The integration of azure sphere and azure cloud services for internet of things," *Applied Sciences*, vol. 9, no. 13, 2019.

[121] H. F. Chong and D. W. K. Ng, "Development of iot device for traffic management system," in *2016 IEEE Student Conference on Research and Development (SCOReD)*, pp. 1–6, Dec 2016.

[122] V. M. Reddy, N. Vinay, T. Pokharna, and S. S. K. Jha, "Internet of things enabled smart switch," in *2016 Thirteenth International Conference on Wireless and Optical Communications Networks (WOCN)*, pp. 1–4, July 2016.

[123] I. D. Addo, S. I. Ahamed, S. S. Yau, and A. Buduru, "A reference architecture for improving security and privacy in internet of things applications," in *2014 IEEE International Conference on Mobile Services*, pp. 108–115, June 2014.

[124] E. Al-Masri, I. Diabate, R. Jain, M. H. Lam, and S. Reddy Nathala, "Recycle.io: An iot-enabled framework for urban waste management," in *2018 IEEE International Conference on Big Data (Big Data)*, pp. 5285–5287, Dec 2018.

[125] F. J. Ferrández-Pastor, J. M. García-Chamizo, M. Nieto-Hidalgo, and J. Mora-Martínez, "Precision agriculture design method using a distributed computing architecture on internet of things context," *Sensors*, vol. 18, no. 6, 2018.

[126] S. Lin, B. Miller, J. Durand, G. Bleakley, A. Chigani, R. Martin, B. Murphy, and M. Crawford, "The Industrial Internet of Things volume G1: reference architecture," *Industrial Internet Consortium*, pp. 10–46, 2017.

[127] "Qualer Search Engine." `https://qualer.com/qualer-search/`. Accessed: 2019-10-11.

[128] D. Zajac, "Creating a Standardized Schema for Representing ISO/IEC 17025 Scope of Accreditations in XML Data," in *NCSL Int. Workshop and Symposium (NCSLI)*, 2016.

[129] "Project Beagle." http://beagledev.azurewebsites.net/. Accessed: 2019-10-11.

[130] "Unit of Measure Editor." http://testsite2.callabsolutions.com/UnitsOfMeasure/index.html. Accessed: 2019-10-11.

[131] P. Ion, R. Miner, S. Buswell, and A. Devitt, *Mathematical Markup Language (MathML) 1.0 Specification*. World Wide Web Consortium (W3C), 1998.

[132] "Azure IoT Reference Architecture 2.1 release." https://azure.microsoft.com/en-us/blog/azure-iot-reference-architecture-2-1-release/. Accessed: 2019-10-11.

[133] B. Familiar, *Microservices, IoT, and Azure*. Springer, 2015.

[134] J. Smith, "Patterns-WPF apps with the Model-View-ViewModel design pattern," *MSDN magazine*, vol. 72, 2009.

[135] E. Sorensen and M. Mikailesc, "Model-View-ViewModel (MVVM) design pattern using Windows Presentation Foundation (WPF) technology," *MegaByte Journal*, vol. 9, no. 4, pp. 1–19, 2010.

[136] J. Galloway, P. Haack, B. Wilson, and K. S. Allen, *Professional ASP. NET MVC 4*. John Wiley & Sons, 2012.

[137] "AWS Documentation." https://docs.aws.amazon.com/index.html?nc2=h_ql_doc. Accessed: 2019-10-11.

[138] "Google Cloud Documentation." https://cloud.google.com/docs/. Accessed: 2019-10-11.

[139] "IBM Cloud Documentation." https://cloud.ibm.com/docs. Accessed: 2019-10-11.

[140] "SoA Editor Source Code." https://github.com/CalLabSolutions/Metrology.NET_Public, https://github.com/mckayadd/TestSoA. Accessed: 2019-10-11.

CURRICULUM VITAE

PERSONAL INFORMATION

Surname, Name: Kaya, Muhammed Çağrı
Nationality: Turkish (TC)
E-Mail: mckaya@ceng.metu.edu.tr
Phone: 0 312 2105592

EDUCATION

Degree	Institution	Year of Graduation
M.S.	Middle East Technical University	2015
B.S.	Firat University	2009

PROFESSIONAL EXPERIENCE

Year	Place	Enrollment
2010-2020	Middle East Technical University	Research and Teaching Assistant

RESEARCH PROJECTS

- TÜBİTAK: TEYDEB Project,
  No: 3151162
  Title: DDS-Based Hybrid Avionic Emulator System Development (HAVESİS)
  Company: STM Defence Technologies Engineering and Trade Inc.
  Role: Researcher
  Advisor: Prof. Dr. Halit Oğuztüzün

PUBLICATIONS

Journal Articles

Science Citation Index Expanded

[1] Kaya, M.C., Saeedi Nikoo, M., Schwartz, M.L. and Oguztuzun, H., 2020. Internet of Measurement Things Architecture: Proof of Concept with Scope of Accreditation. *Sensors*, 20(2), p.503.

[2] Suloglu, S., Kaya, M.C., Karamanlioglu, A., Entekhabi, S., Nikoo, M.S., Tekinerdogan, B. and Dogru, A.H., 2018. Comparative Analysis of Variability Modelling Approaches in Component Models. *IET Software*, 12(6), pp.437-445.

Other Indexes

[3] Kaya, M.C., Cetinkaya, A. and Dogru, A.H., 2018. Off-the-Shelf Connectors for Interdisciplinary Components. *Journal of Integrated Design and Process Science*, 22(3), pp.35-53.

National - Reviewed

[4] Temizer, S. and Kaya, M.C., 2013. Range Measurement Based Reliable Localization Techniques and Sample Applications for Land and Air Vehicles (Mesafe Ölçümü Tabanli Güvenilir Konum Tespiti Teknikleri ve Kara ve Hava Araçları için Örnek Uygulamalar). *Journal of Aeronautics and Space Technologies*, 6(2), pp.33-48.

Book Chapters

[5] Suloglu, S., Kaya, M.C., Cetinkaya, A., Karamanlioglu, A. and Dogru, A.H., 2020. Cloud-Enabled Domain-Based Software Development. In Software Engineering in the Era of Cloud Computing (pp. 109-130). Springer, Cham.

[6] Nikoo, M.S., Kaya, M.C., Schwartz, M.L. and Oguztuzun, H., 2019. Internet of measurement things: Toward an architectural framework for the calibration industry. In The Internet of Things in the Industrial Sector (pp. 81-102). Springer, Cham.

[7] Kaya, M.C., Suloglu, S., Tokdemir, G., Tekinerdogan, B. and Dogru, A.H., 2019. Variability Incorporated Simultaneous Decomposition of Models Under Structural and Procedural Views. Software Engineering for Variability Intensive Systems: Foundations and Applications (pp.95-116). CRC Press.

[8] Kaya, M.C., Eroglu, A., Karamanlioglu, A., Onur, E., Tekinerdogan, B. and Dogru, A.H., 2019. Runtime Adaptability of Ambient Intelligence Systems Based on Component-Oriented Approach. In Guide to Ambient Intelligence in the IoT Environment (pp. 69-92). Springer, Cham.

[9] Kaya, M.C., Nikoo, M.S., Suloglu, S., Tekinerdogan, B. and Dogru, A.H., 2017. Managing heterogeneous communication challenges in the internet of things using connector variability. In Connected Environments for the Internet of Things (pp. 127-149). Springer, Cham.

International Conference Publications

[10] Kaya, M.C., Karamanlioglu, A., Çetintaş, İ.Ç., Çilden, E., Canberi, H. and Oğuztüzün, H., 2019, July. A Configurable Gateway for DDS-HLA Interoperability. In Proceedings of the 2019 Summer Simulation Conference (p. 36). Society for Computer Simulation International.

[11] Cetinkaya, A., Suloglu, S., Kaya, M.C., Karamanlioglu, A., Tokdemir, G. and Dogru, A.H., 2019, July. An Experimental Study on Decomposition: Process First or Structure First?. In International Symposium on Business Modeling and Software Design (pp. 279-289). Springer, Cham.

[12] Nikoo, M.S., Kaya, M.C., Schwartz, M.L. and Oguztuzun, H., 2019, June. An MII-Aware SoA Editor for the Industrial Internet of Things. In 2019 II Workshop on Metrology for Industry 4.0 and IoT (MetroInd4.0&IoT) (pp. 213-218). IEEE.

[13] Cetinkaya, A., Kaya, M.C. and Dogru, A.H., 2017, November. Component Integration Through Connector Supported Process Models. In Proceedings of SDPS 22nd international conference on emerging trends and technologies in convergence solutions (pp. 31-37), Birmingham, Alabama.

[14] Kaya, M.C., Cetinkaya, A. and Dogru, A.H., 2017, September. Composition Capability of Component-Oriented Development. In 5th International Symposium on Innovative Technologies in Engineering and Science (ISITES2017) (pp. 1299-1307), Baku-Azerbaijan.

[15] Cetinkaya, A., Kaya, M.C. and Dogru, A.H., 2016, December. Enhancing XCOSEML with Connector Variability for Component-Oriented Development. In Proceedings of SDPS 21st international conference on emerging trends and technologies in designing healthcare systems (pp. 120-125), Orlando, FL, USA.

[16] Kaya, M.C., Saeedi Nikoo, M., Suloglu, S. and Dogru, A.H., 2015, November. Towards verification of component compositions incorporating variability. In Proceedings of SDPS the 20th international conference on transformative science and engineering, business and social innovation (pp. 202-207). Fort Worth Texas USA.

[17] Kaya, M.C., Suloglu, S. and Dogru, A.H., 2014, June. Variability Modeling in Component-Oriented System Engineering. In Proceedings of SDPS the 19th international conference on transformative science and engineering, business and social innovation (pp. 251-259). Kuching Sarawak, Malaysia.

[18] Kaya M.C., Eroglu A. and Temizer S., 2013, May. Control and Motion Planning for Mobile Robots via a Secure Wireless Communication Protocol. In Proceedings of 1st International Symposium on Digital Forensics and Security (ISDFS'13) (pp. 227-230), Elazig, Turkey.

National Conference Publications

[19] Kaya M. C.,Çilden E.,Canberi H. and Oguztuzun H., 2017, November. A Literature Survey on DDS-HLA Interoperability Approaches (DDS-HLA Birlikte

Çalışabilirlik Yaklaşımları Üzerine Bir Literatür Araştırması). Yedinci Ulusal Savunma Uygulamaları Modelleme ve Simülasyon Konferansı (USMOS2017) (pp. 60-71), Ankara, Turkey.

[20] Cetinkaya, A., Karamanlioglu A., Kaya, M.C. and Dogru, A.H., 2017, October. Integration Approach through Coordinated Components (Eşgüdümlü Bileşenler ile Birleştirme Yaklaşımı). 11. Ulusal Yazılım Mühendisliği Sempozyumu (UYMS 2017) (pp. 619-631) Alanya, Turkey.

[21] Kaya, M.C., Karamanlioglu, A., Nikoo, M.S., Entekhabi, S., Suloglu, S. and Dogru, A.H., 2016, October. Bilesen Modellerinde Degiskenlik Yonetimi Yaklasimlarinin Karsilastirilmasi. In 10. Ulusal Yazılım Mühendisliği Sempozyumu (UYMS2016) (pp. 502-513), Canakkale, Turkey.

[22] Kaya M. C., Ciftci S. and Temizer S., 2013, June. Indoor Localization for Partially-Dysfunctional Autonomous Mobile Robots Using Minimum Number of Sonar Sensors (Kısmen Arızalı Otonom Robotlar İçin En Az Sayıda Sonar Sensörü Kullanılarak Kapalı Mekanlarda Konum Tespiti Gerçekleştirilmesi). Beşinci Ulusal Savunma Uygulamaları Modelleme ve Simülasyon Konferansı (USMOS2013) (pp. 135-144), Ankara, Turkey.

Master's Thesis

[23] Kaya, M.C., 2015. Modeling variability in component oriented software engineering, MSc Thesis, Middle East Techical University.