TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

HAFIZ M. AITZAZ HASSAN

SEMANTIC MODELS AND STATE TRACKING FOR PLUG & PRODUCE IN AUTOMATION SYSTEMS

Master of Science Thesis

Examiners: Prof. Jose L. Martinez Lastra and A.Prof. Andrei Lobov
Examiner and topic approved by the Faculty Council of the Faculty of Engineering Sciences
on 01 November 2017

ABSTRACT

HAFIZ M. AITZAZ HASSAN: Semantic Models and State Tracking for Plug & Produce in Automation Systems
Tampere University of technology
Master of Science Thesis, 50 pages, 06 Appendix pages
March 2018
Master's Degree Program in Automation Engineering
Major: Factory Automation and Industrial Informatics
Examiners: Professor Dr. Jose L. Martinez Lastra, Associate Professor Dr. Andrei Lobov
Supervisors: Dr. Andrei Lobov, Dr. Sten Grüner, Dr. Johannes O. Schmitt

Keywords: Information Models, Semantics, Factory Automation, State Machines, OPC UA, PackML

Modern manufacturing systems must be highly flexible, scalable and dynamic to adapt to changes necessary to deal with mass customization issues arising as result of changing customer behavior. The production systems must have the capability to integrate and remove system components in a short-time and ideally on the fly to meet changing product and customer requirements. This can be achieved by developing a hierarchical production system composed of dynamically configurable system components capable of interacting with each other. This thesis presents a methodology to develop configurable information models for shop-floor devices, state machine models for tracking the states of system building-blocks and their interaction mechanism.

This research work aims to explore information models and standards being used in Process and Factory Automation industry. It involves semantic modelling of production resources and exposing their functionality to an information modelling and machine-to-machine (M2M) communication tool using Service Oriented Architecture (SOA). It presents a mechanism to inter-link information models for bi-directional flow of information across the layers of manufacturing system. It also proposes a methodology to build a scalable hierarchical manufacturing system with state-tracking capability of systems and sub-systems on each hierarchical level. The proposed solution allows reusability of information models which makes system deployment and scaling easier and faster. Finally, the developed concept was tested on a manufacturing system and state-tracking, interaction between system components and bi-directional event-propagation were achieved.

PREFACE

This thesis is the result of continuous effort and consistent hard work of several months spent with the belief of doing something productive. Herein, I would like to acknowledge the support and encouragement, I received from different people during this period.

First of all, I would like to thank Almighty for giving me the physical and mental strength, courage and good health for working on this project. Words are not enough to describe His countless blessings upon me.

I am thankful to my group members at ABB Research Center, Germany for providing me a peaceful and excellent research environment to conduct this research work. Specially, I am grateful to my supervisors Dr. Sten Grüner and Dr. Johannes O. Schmitt for their sincere advices, active support, constructive criticism and right guidance. I would also like to thank them for listening and answering my questions and work-related issues with patience. Without their active participation, guidance and supervision this work was impossible to complete in time.

I would also like to express my gratitude to my supervisor Dr. Andrei Lobov from TUT for his guidance, valuable suggestions regarding the project direction and active participation in the team meetings. I am also thankful to Prof. Jose L. Martinez Lastra for being the examiner of this thesis.

In the last, I would like to thank my friends and fellow students at ABB for cracking jokes and interesting lunch time discussions. At this point, I cannot forget my University Friends especially Daniyal, Adnan, Shahbaz and Umer Iftikhar for the encouragement I received from them.

Finally, my heartiest gratitude to my parents, siblings and fiancee for their love, encouragement, wishes and countless prayers that kept me going during this period.

Mannheim, 18.03.2018

Hafiz Muhammad Aitzaz Hassan

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS AND ABBREVIATIONS

AI	Artificial Intelligence
API	Application Programming Interface
AR	Augmented Reality
COM	Component Object Model
DCOM	Distributed Component Object Model
ERP	Enterprise Resource Planning
FSM	Finites State Machine
HTTP	Hyper Text Transfer Protocol
IaaS	Infrastructure as a Service
ICT	Information and Communication Technologies
IIoT	Industrial Internet of Things
IP	Internet Protocol
IT	Information Technology
OMAC	Organization for Machine Automation & Control
OPC	Object Linking and Embedding for Process Control
OPC UA	OPC Unified Architecture
PaaS	Platform as a Service
PackML	Packaging Machine Language
PackML_FSM	PackML Finite State Machine
REST	Representational State Transfer
RFID	Radio Frequency Identification
SaaS	Software as a Service
SOAP	Simple Object Access Protocol
SOA	Service Oriented Architecture
TCP	Transfer Control Protocol
URL	Uniform Resource Locator

1. INTRODUCTION

In today's world, advancements in technology are shaping the manufacturing industry. Over the last few decades, manufacturing industry has changed entirely by continuously replacing the manual jobs with automated processes especially in European and American markets for several reasons. Automating the manufacturing processes has helped in achieving higher production rates, addressed the labor shortage issues due to aging and higher demand, replaced the routine tasks considered boring or cumbersome and contributed to worker safety by handling dangerous material. In addition, Automation has resulted into high-quality products with higher uniformity and compliant to quality standards. To benefit from the advantages of automation, manufacturing industry has been in an ever-evolving process and has already gone through three industrial revolutions.

1.1 Motivation and Objectives

Remarkable advancements in ICT technologies such as Industrial Internet of Things (IIoT), Cloud Technologies (IaaS, PaaS, SaaS), Artificial Intelligence (AI), Big-Data & Analytics and Augmented Reality (AR) have pushed the modern-day manufacturing practices forward. These cutting-edge technologies described as "exponential technologies" by Deloitte [1] are backbone of fourth industrial revolution commonly known as "Industry 4.0".

With the changing customer behavior there arises a need of mass customization. Smaller batch sizes and shorter production cycles are scheduled to address this mass customization issue which can increase the product cost considerably and in-time delivery can be a great challenge if the manufacturing systems takes large amount of time to adapt to changes. Modular, flexible, dynamically configurable and expandable manufacturing system can address these issues faced by the industry due to high customizability trends. Such dynamically configurable modular manufacturing systems are often composed of hierarchy of systems and sub-systems. Tracking the states of systems and sub-systems in such a hierarchical manufacturing system is of vital importance. State aggregation may result into many benefits such as condition monitoring, root cause analysis, production planning, efficient resource utilization, product cost reduction etc. Efficient integration and configuration of information models is essential for pragmatic state aggregation of such hierarchical manufacturing systems.

The idea of efficient state aggregation and configuration of systems and sub-systems to achieve perceived benefits can be realized by utilizing modern technologies.

Advancements in ICT has provided a great deal of IT knowledge which has made much more information available on all hierarchical levels than it was ever-before [2]. Service Oriented Architecture leverages to encapsulate and provide the functions of physical devices as discrete callable services e.g. REST/SOAP services, which can be exposed to communication systems such as OPC UA. Hence, the realization of a modular, expandable, configurable and hierarchical manufacturing system in which states of systems and sub-systems can be aggregated is realizable by utilizing modern manufacturing technologies.

In the context of above mentioned benefits which can be achieved by efficient state aggregation, the objectives of this thesis are to:

- Define a methodology to develop a modular hierarchical system composed of information models in which state of system and sub-systems can be tracked on each hierarchical level of the developed system.
- Define information/state models which are horizontally and vertically stackable.
- Define mechanism that how information models will be configured and inter-operate for upward and downward flow of information.
- Test the developed concept on a use case for proof of concept.

1.2 Problem definition

As mentioned in Section 1.2, this work focuses on developing a modular system in which information models can interact efficiently in a way that state of the system can be aggregated. Development of such a system consists of defining a system architecture, information models' definition, implementation of concept and testing on a use case. Keeping in view the objectives of this work, the desired outcomes can be achieved by answering the following research questions:

- Which information is required to represent shop-floor devices into the system so that their functionality can be exposed properly?
- What information is required for state machines so that devices' states can be aggregated into meaningful and sufficient system states?
- What mechanisms are required for integrating information models?
- How the developed information/state models will be configured so that they may interact with each other?
- How to develop reusable information models?

1.3 Work Description

This section gives an overview of how this research work will proceed by providing a research methodology and what are the limitations and assumptions made during the study.

1.3.1 Research Methodology

Research methodology for conducting this thesis work is described in this section. First, a literature review of state of the art technologies, relevant manufacturing standards, common practices and tools are discussed. It is followed by mechanisms and approach for developing information and state machine models for shop-floor devices. It also discusses the possible interaction mechanisms in-between different information models for building the perceived system. A general system architecture followed by the desired hierarchical system is presented.

After presenting the envisioned system and methodology, the concept is implemented on a mobile phone manufacturing system namely FASTory Line for the proof of concept. Use case is defined in detail with specific requirements. The developed concept is tested by defining information models for shop-floor devices, machine specific state machines, PackML compliant state machines and inter-linking those models. In the end, the goals achieved as a result of this work are discussed and thesis is concluded.

1.3.2 Assumptions and Limitations

Following assumptions and limitations were observed during the conceptualization and implementation phase of this work:

- The developed concept is applicable on modular production systems.
- Shop-floor devices provide their functionality as discrete operations which can be exposed to upper level digital system as service e.g. REST services.
- For proof of the developed concept, FASTory Simulator, which provides its functionality as RESTful API, is used for testing purpose rather than the real FASTory line. However, FASTory Line also provides its services as REST API so the developed system can be tested on the real production line.
- For use case implementation, it is assumed that the recipe that will add behaviour to the device will reside in the device-specific state machine.

1.4 Thesis outline

This work is structured in the following manner. Chapter 1 is Introduction which details motivation and desired objectives, defines the problem and lists the limitations of this

study. Chapter 2 is Theoretical Background which discusses state-of-the-art technologies concepts technologies, tools and standards relevant to this work. Chapter 3 proposes a general overview of the system architecture and approach for developing the system. Chapter 4 details the implementation of the developed concept on a use case. Chapter 5 discusses the outcomes of the study. Chapter 6 draws conclusion out of the conducted research and proposes the possible future work.

2. THEORETICAL BACKGROUND

This section provides a literature review of the relevant concepts that are important to understand the developed concept. It also discusses briefly the standards and state-of-the art technologies used for the realization of this work. This part will help the reader to develop basic concepts which are necessary to understand the essence of this work.

2.1 ICT and Manufacturing Systems

The role of Information and Communication Technologies cannot be overlooked in modern manufacturing systems. Tremendous increase in the use of ICT in contemporary manufacturing systems has been observed in past few years. In an attempt to build smart systems, the flow of information has also increased. IT support has made large amount of information flow possible across all layers of automation pyramid starting from shop-floor up to ERP level.

Advancements made in ICT over the last one and a half decade has not only changed the market place but it has also changed the buying habits of customers [3]. Customers are now more sensitive to product quality and demand highly customizability. This trend encourages companies to continuously improve their product quality and adapt to market changes in short time to gain a competitive edge over their competitors. This requires contemporary production systems to be modular, scalable and easily configurable so that they can produce small batches and adapt to new requirements in minimum amount of time. To realize such systems, effective use of ICT is inevitable. In order to complement contemporary manufacturing systems, [2] envisions that ICT has to support the following objectives:

- Provide the customer with desired product quality. This demands ICT to provide support for moving robust production to flexible production processes.
- Adapting to the changes in short time with regard to change in product requirement.
- Lower production costs while introducing new technologies and equipment to production process.

These objectives guide the design of IT infrastructure and system of modern manufacturing processes. The factories should be adjustable to changing market demands and adapt to required changes in IT infrastructure in a short amount of time to compete with other stakeholders in the market. With evolving product requirements, there might be need of changing the shop-floor devices, IT infrastructure supporting the

manufacturing system should have the capability to adapt to changes accordingly with minimum cost. Factories of the future are also supposed to provide monitoring and state tracking capabilities of systems and sub-systems on all levels of the manufacturing system.

2.1.1 Transformation of Manufacturing Industries IT Architecture

Improvements in Information and Communication Technologies (ICT) has also changed the IT architecture of manufacturing enterprises. Until recent past, IT infrastructure of manufacturing industries was reflected in multi-layered structures. Automation Pyramid (Figure 1) is one of the most popular models used for this purpose. Each layer in the pyramid represent some special functions that constitute that layer. Lower most layer collects or sends the raw data coming from shop-floor devices. Quality of the data improves as we move upward in the pyramid. Width of the pyramid on each level reflects the amount of data available on that layer. Shape of the pyramid also reflects the decrease in the amount of data as we move from bottom to top.

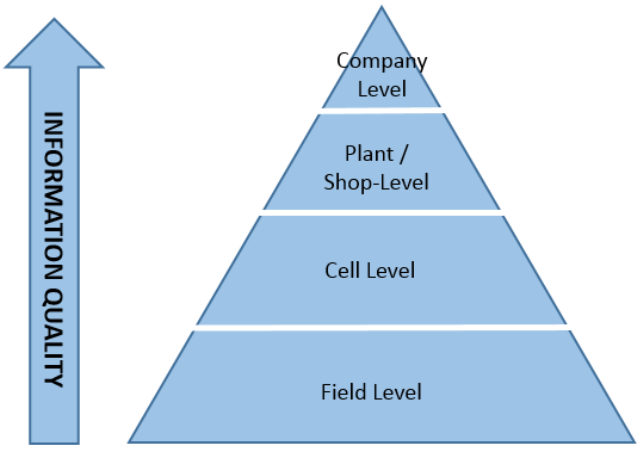

Figure 1. Automation Pyramid. Modified from [4]

Due to the increasing use of ICT on all levels of manufacturing enterprises, there is need to replace the old reference model to keep up with emerging trends of intensive information flow. A new 'reference model of industrial information architecture' is presented in [4]. This new reference model (Figure 2) covers three-dimensional integration: vertical integration aspect that emphasizes on shop-floor devices to be integrated with upper level control devices and monitoring systems, horizontal integration which requires the system elements to be modular and production equipment life-cycle integration. This reference model presents the idea of building information model for shop-floor devices so that they can be represented and accessed by upper level digital systems.

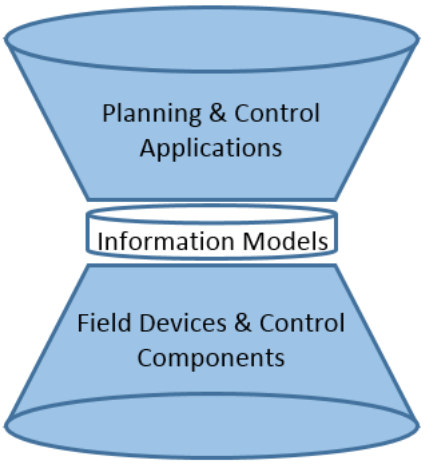

Figure 2. New Reference Model of Industrial Information Architecture. Modified from [5]

2.2 Packaging Machine Language (PackML)

Packaging Machine Language (PackML), as described by World Batch Forum, is a standard that helps achieve standardized way of communication between machines and provide the end-user with a standard process data. The standard after its initial release in 2002-03 has been widely adopted across the packaging industry.

PackML came into existence as a result of OMAC (Organization for Machine Automation & Control) group's effort of developing a method to integrate different vendor machines with ease and in short time. Since its release, the standard has gone through many changes. The template based upon the current version of the standard i.e. ISA-TR88.00.02 has a base model with 17 standard states and multiple control modes. [5]

2.2.1 Machine States

According to [5], "A unit/machine state completely defines the current condition of a machine." There are two types of states defined in [5]: Acting states and Wait states. Acting States can include any kind of processing activity whereas Wait states identify that a machine has reached to a desired condition-set. The base state model of PackML implementation is shown in the Figure 3. It includes 17 defined states shown in the figure. Each of this state can either be an Acting state (Green) or a Wait state (Yellow). Execute State coloured in Blue is an exception. Machine will be in Execute state once its start its normal production after passing through the starting state.

2.2.2 State Transitions

[5] defines transitions as "A passage from one state to another". In PackML transitions between states occur as a result of a local or remote procedural command. Theses state commands may be triggered by one or combination of the followings [5]:

- Line operator interference,
- Status change of procedural elements,
- Machine conditions parameter update,
- Acting state completion, and
- Interference from the Supervisory System.

2.2.3 Control Modes

Control mode determines the actions a machine takes to execute or accomplish a process. It specifies the states included in that mode, transition between the states and state commands [5].

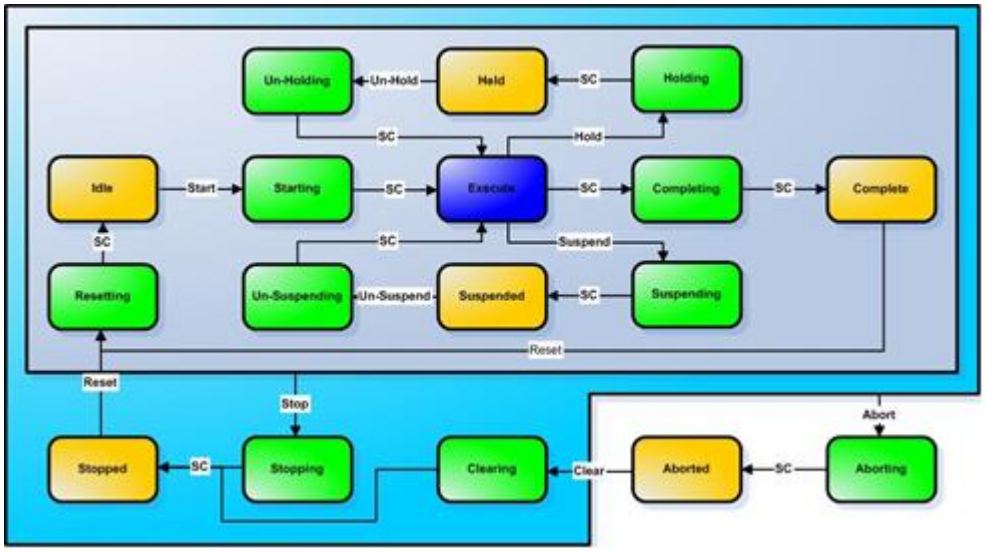

Figure 3. PackML-Base State Model [7]

ISA-TR88.00.02 defines some typical machine control modes such as production, maintenance, manual, clean in place, run out, semi-auto, dry cycle etc. It also supports user-defined control modes. A control mode typically consists of a subset of states defined in the base state model shown in Figure 3. Control modes distinguish from each other based on states, transition between states and state commands [5]. One important point to be noticed is that states with same name can have different operation/procedure associated with it in different control modes.

2.2.4 PackML Implementation Steps

OMAC has published a guide for implementing PackML standard [6]. Five implementation principles to be followed sequentially given in the guide are:

- Divide physical machine components using ISA-88 standard physical hierarchy for batch control which includes three entities i.e. Machine unit, Equipment module and control module.
- Define the PackML modes and States.
- Define the actions of modules of the machine during each state and mode.
- Bind or define the variables with the PackTags(data elements within the state model) for integration of the machines in the line and MES functionality.
- Define alarms and the conditions which gives the stop or abort commands.
- Develop the algorithm for the machine.

2.3 ISA-88 Batch Control Overview

ANSI/ISA-88 also known as S88 is a commonly used international standard in Batch Process industry. The standardization activity started by the ISA (International Society of Automation) back in 1988, was aiming to answer the following fundamental issues of the batch process control industry [8]:

- There was no common terminology being used in the batch industry and explaining the process among business as well as engineering circles was an uncomfortable task.
- Design and Instalment of a batch control process was a difficult process and large man power was needed.
- Integrating the equipment coming from different vendors was a painful and time-consuming task for control engineers.

ISA-88 addressed these above-mentioned problems. The batch process systems built following the ISA-88 standard experienced many benefits including shorter design & implementation time, more reliability, reduced process cycle-time, reduced raw material waste, reduced downtime, better data availability, better resource utilization, higher production efficiency and easier recipe modification [9].

ISA-88 has following five parts:

1. Models and Terminology: Documentation of reference models and common terminology used in batch process industry.
2. Data Structures and Guidelines for Languages: Describes data model, data structures to support communication between system elements and language guideline for recipe implementation.

3. General and Site Recipe Models and Representation: Defines reference model to implement site and general recipes.
4. Batch Production Records: Description of reference model for records which keep the information of batches or the entities involved in the production.
5. Machine and Unit States: An Implementation Example of ISA-88.

Keeping in view the scope of this thesis only Part 1 of the standard is discussed in the remainder of this section.

[10] describes batch control as "A process that leads to the production of finite quantities of material by subjecting quantities of input materials to an ordered set of processing activities over a finite period of time using one or more pieces of equipment."

ISA-88 Part 1 defines a physical model shown in the Figure 4. In any batch control process physical entities are mapped according to this model. This physical model maps the entities which take part in the manufacturing process starting from the Process Cell up to the lowest Control Module level. [10] The upper three levels are not described in detail in the Standard for being highly business-oriented. One can think of enterprise level as, for example, automotive producer company where site-level corresponds to one production plant and the area-level corresponds to one shop-floor of that plant site. The remaining four levels describe the structure of the equipment involved in the process. They are defined in the standard as follows:

Process cell: "A logical grouping of equipment that includes the equipment required for production of one or more batches. It defines the span of logical control of one set of process equipment within an area", e.g. a Packaging Line.

Unit: "A collection of associated control modules and/or equipment modules and other process equipment which can conduct one or more processing activities", e.g. A Machine unit such as a Work Station.

Equipment module: "A functional group of equipment that can carry out a finite number of specific minor processing activities", e.g. A Robotic Arm in a Work Station.

Control module: "The lowest level grouping of equipment in the physical model that can carry out basic control", e.g. Actuators such as servomotors.

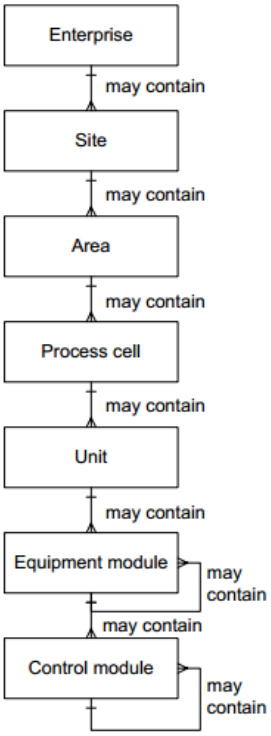

Figure 4. Physical Model of ISA – 88 [10]

ISA-88 Part 1 [10] also defines the recipe which is the information required to define the manufacturing process of an item. ISA-88 specification distinguishes recipes into different types. Figure 5 shows the hierarchies of recipe types defined in the standard. General Recipe describes the production information in a very abstract manner that is useful at enterprise level. It includes information about raw material type, material quantity as well as processing activities but the information is much generalized. A site-recipe is typically defined based on a General-recipe and includes site-specific information, for example, site-specific material, equipment and other local resources to be used for the batch production. For a process cell, Master-recipe containing information specific to that process cell equipment modules are specified. Control recipe is derived while keeping in view the Master recipe and it contains process specific information e.g. scheduling information and operation information.

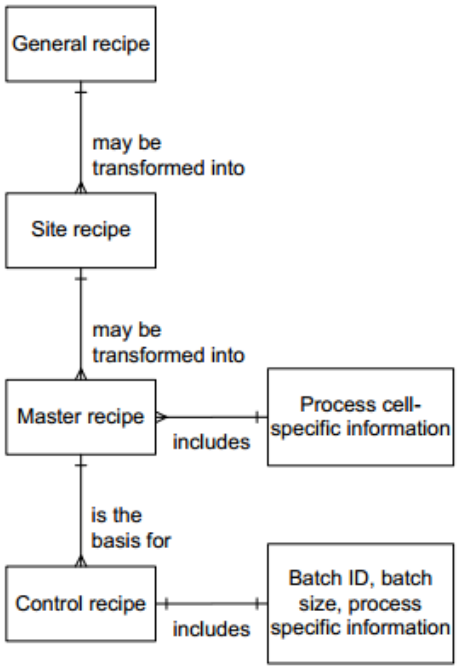

Figure 5. Reference Recipe Model ISA-88 [3]

2.4 OPC Unified Architecture

OPC UA (Unified Architecture) is the most recent machine to machine communication protocol provided by the OPC (Open Platform Communication) foundation for industrial automation. The main doctrine behind OPC UA is based upon the widely applied client-server communication [11], in which server keeps the data associated with different entities and the client access that data by interacting with the server in an effective way. For instance, a device on the shop floor can write to OPC server and then the OPC client can read and use that data for further processing.

OPC UA is the extension of Classic OPC specifications which has been widely followed as communication standard among the devices in industrial automation domain. The basic difference between the two standard lies in the communication protocols each of them use. Classic OPC made use of Microsoft's OLE (Object Linking and Embedding) protocols which was developed to exchange objects between different software systems. Classic OPC then evolved over the time and made use of Microsoft's COM (Component Object Model) and DCOM (Distributed Component Object Model) technologies [12]. OPC UA, on the other hand, exploits modern transport protocols namely TCP/IP, HTTP/SOAP and security protocols such as WS-SecureConversation and UA-SecureConversation [13].

Classic-OPC can be seen as a collection of standards developed over the time based upon the end-user needs. It has the components such as OPC Data Access, OPC Alarm & Events, OPC Historical Data Access etc. Since these components were developed in

different periods of time and independent to each other, many essential features were for client-server software were missing and hence, enforced the separate development of such features. Furthermore, Classic-OPC do not support the information model development and metadata modelling and there is no notion of address spaces since it was designed for representing basic automation data. OPC UA is a unified approach for data modelling and communication which keeps under consideration the other relevant aspects of automation. OPC UA services along with metadata associated with the models allow to pass and receive messages and thus, provides an ideal environment for the modular exchange of data between entities and sub-entities. "OPC UA provides SOA (Service Oriented Architecture) for industrial applications – from factory floor devices to enterprise applications" [14].

Another significant difference between Classic-OPC and OPC UA is the range of applications they can be used for. Classic-OPC was developed to cope the needs of process industry and later was used for other automation industry applications as well. However, it has limited or no support for data exchange in ERP applications. This shortcoming has been targeted in OPC UA by providing support for semantics modelling [16]. In contrast to Classic-OPC semantics are not pre-defined and can be defined based on the context of the data to be modelled.

2.4.1 System Architecture

This section explains some of the core concepts necessary to develop general understanding of the OPC UA system.

2.4.1.1 Client-Server Approach

Simplest structure of OPC UA system follows the client-server architecture in which server contain some services and offers those services to clients via interfaces. Clients can utilize the services provided by the server to complete certain tasks. Services are independent entities and can be thought of methods used to access the data on the server [15].

OPC UA client sends requests on the OPC UA server via services (defined in the later section) and receives a concise response back as defined in the OPC UA specifications. The communication is made possible via modern network and web protocols such as TCP/IP and HTTP/SOAP or HTTPS which have the capability to pass through firewalls [15]. Figure below gives an overview of a possible OPC UA system in which different field devices distributed over a network bus are integrated into UA Server.

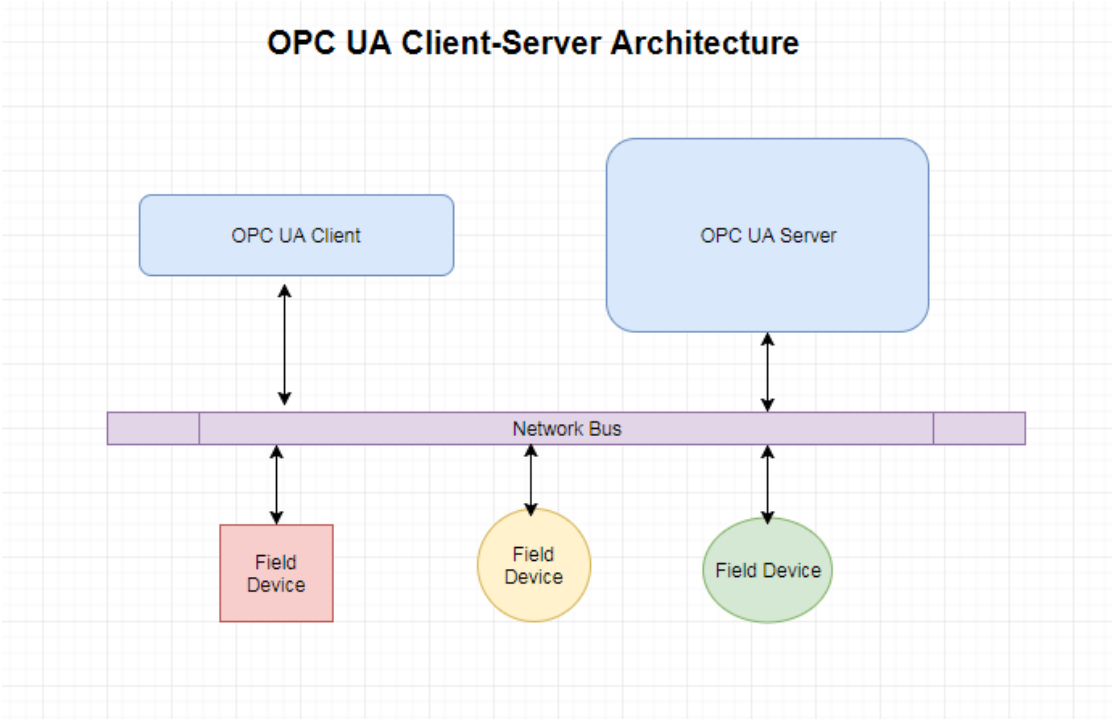

Figure 6. OPC UA Client Server Architecture

2.4.2 OPC UA Specification

OPC UA specifications are series of documents provided and maintained by OPC foundation which explain the OPC UA concepts in detail [16]. Two fundamental concepts discussed in OPC UA specifications are information modelling and transport protocols. Transport protocols are out of the scope of this work and will not be discussed any further.

Part 5 of OPC UA specification specifies the rules for defining information models and expressing them in OPC UA address space [17]. A base Address Space has been defined in [17] which include entry points and base types. User have the leverage of defining its own new types based on application specific requirements. Using the entry points and type definitions, user can build type hierarchies and abstract information models can be defined.

2.3.2.1 *Address Space*

UA Part 1 [18] defines the Address Space as "the collection of information that an OPC UA Server makes visible to its clients".

Each component of the address space is referred as a “Node”. Nodes can interact with each other via “References”. There are some base reference types such as “HasTypeDefinition” and “Organizes” to declare instances and define type hierarchies, respectively. Besides base reference types, OPC UA allows to define user-specific

reference types [17]. Address Space can be perceived as an interconnected network of Nodes connected via references.

OPC UA defines eight different type of node classes which include Object, ObjectType, Method, Variable, VariableType, View, ReferenceType and DataType. The purpose of each of these Node Classes is as follows:

A real-world entity is represented by an Object, for instance, a work-station in an assembly line. Objects are also used to organize the Address Space such is the case of base hierarchy which indeed is a collection of objects.

Object Types are the abstract representation or description of a real-world entity such as a work-cell. Each object is instantiated from an object type. An Object Type defines the structure of the objects instantiated from it i.e. sub-objects, parameters, properties it will have.

To hold data associated with an object Variables are used. Object properties are special kind of variables which cannot have any child nodes.

Each variable has a Variable Type which determines the structure and characteristics of that variable.

As evident from the name Data Types determine the type of data. Some basic data types are Numeric, String etc. OPC UA also provides the leverage of defining new data types based on user needs.

Semantics of references between Nodes is specified by Reference Types. Reference points from a source node to a target node and the reference type determines the relation between those nodes. A reference may have source and target Nodes in the same or different Address Spaces.

Methods are functions which are represented as the child nodes of the objects. A method associated with an object can perform operations defined in the method definition and may or may not return a value. For example, a robot work-station may have a method to open the door and another to close the door.

Views are used to make visible only specific parts of the address space and hide the rest. As an example, we can have different views for Production and Diagnostics. Production view will show only those parts of the address space which are relevant and rest of the parts will be invisible.

These eight Node Classes and their attributes are shown in the Figure 7.

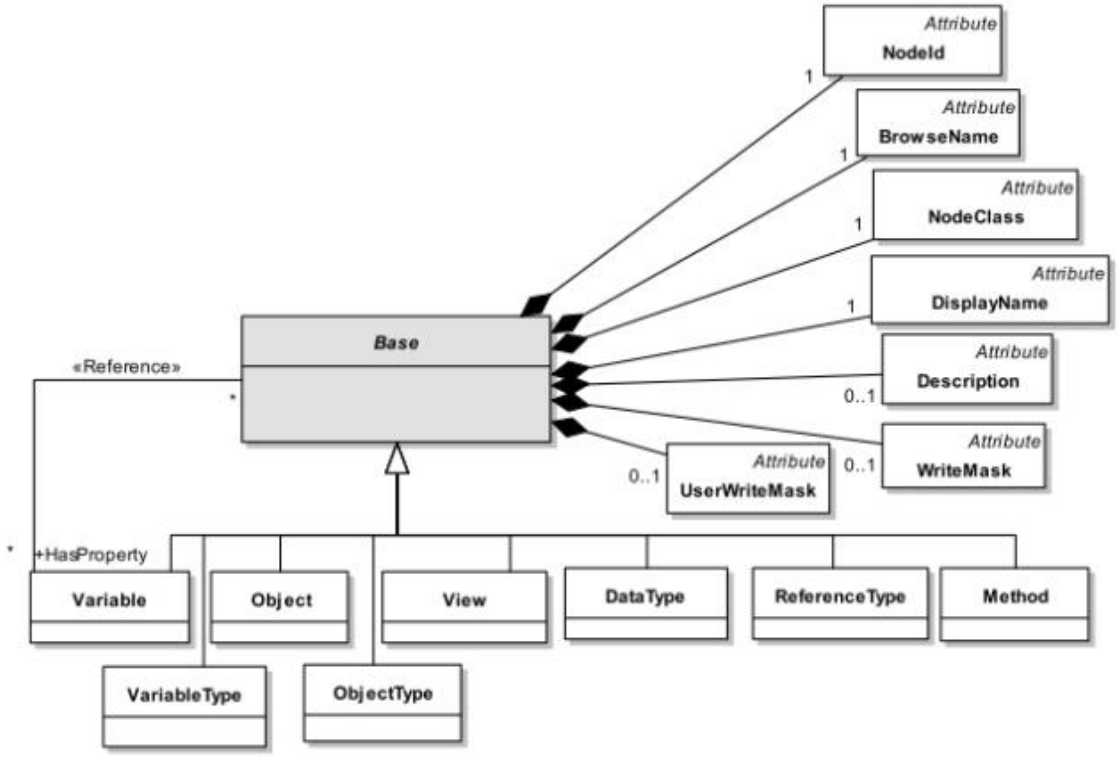

Figure 7. Node Class Hierarchy [19]

Each Node Class has a specific set of attributes associated with it. Base Class Types which is the parent of all the other Node Class is not a real class rather it is just a collection of attributes. Since every node is the child of Base Node so every node has at the least these seven base attributes. NodeId, BrowseName, NodeClass, DisplayName are mandatory attributes of each node. Each node needs to specify the values of these mandatory nodes. Whereas WriteMask, Description and UserWriteMask are optional attributes and do not need to be specified necessarily. Out of these seven attributes, a threefold attribute NodeId consisting of address space index, an enumerated identifier type and the identifier itself, is the most important and unique for every node.

Base structure of the Address Space is shown in the Figure 8 which forms the basis of the OPC UA server hierarchy. It consists of object nodes used to organize the hierarchies. One can think of these base nodes as folders in which nodes are organized based on their Node Class. Being part of every OPC UA server these nodes included in this base structure serve as the entry point to the OPC UA Address Space.

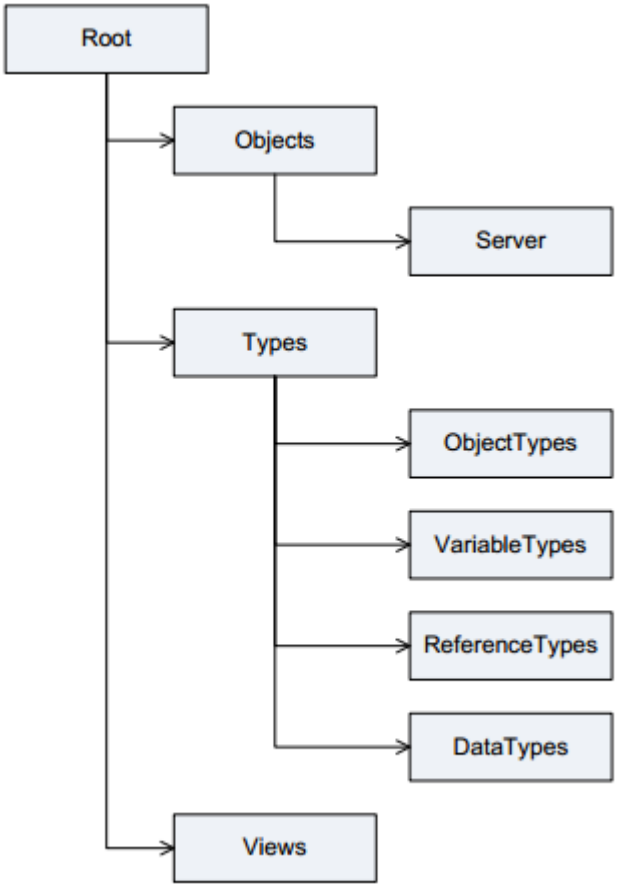

Figure 8. OPC UA Server Object Hierarchy of [20]

2.5 Eco System of State Machines

State Machines are used for representing behavior models. They consist of a finite number of states and transitions. A state is a system element which depends on previous inputs and has some impact on the following inputs. System may switch from one state to another via transitions. As evident from the name, transition defines the transition from one state to another and may typically have some transition condition associated with it. Finite State Machine (FSM) contains finite number of states per definition and suffice for modelling the considered systems for this master thesis. In the following text, FSM and their different types have been discussed.

2.5.1 Brief History of Finite State Machines

Finite state machines (FSM) is a commonly used technique to describe the interactive behavior of the systems. The very first person who suggested to use FSM for behavior modelling of graphical systems in 1968 was William Newman [21]. According to

Newman “the same action may cause a different reaction on different occasions” [21]. Based on this observation, he suggested the use of FSM because they are capable of modelling input as well as output of an interactive system keeping in view the current state of the system.

2.5.1.1 Basic State Machines Types

Mealy and Moore are two most basic types of finite state machines. Mealy state machines are names after its founder George H. Mealy who gave the idea in his paper “A Method for Synthesizing Sequential Circuits”. In Mealy state machines output is a function of input and current state. For each pair of input and state there is only possible transition [22]. In contrast to Mealy, Moore state machines presented by Edward F. Moore does not include input to determine the output. In other words, output is only a function of current state [23].

There are pros and cons of using each type and selection can be a tricky decision at times. Moore FSM are considered safer because unlike Mealy a state change is independent of input and synchronization problems can be avoided if two or more machines are interconnected. Mealy machines, on the other hand, react faster to inputs because they do not wait for the clock signal. However, according to automata theory, both Mealy and Moore FSM are interconvertible.

2.5.1.2 Complex State Machine Types

David Harel in 1984 presented the idea of state charts for behavioral system representation. Harel observed “A complex system cannot be beneficially described in this naive fashion, because of the unmanageable, exponentially growing multitude of states, all of which have to be arranged in a ‘flat’ unstratified fashion, resulting in an unstructured, unrealistic, and chaotic state diagram”. Based on this observation he concluded that a state approach for system representation can be useful if it is modular, hierarchical and well-structured [24].

For representing complex systems, Harel suggested the use of new state elements and introduced the term “statechart”. A statechart typically integrates the following concepts: state-diagrams, depth, orthogonality and broadcast communication. Depth was brought to state diagrams by using sub-states and composite-states and keeping the overall diagram structure more clear and well-structured. Different sub-state machines can operate in parallel alongside each other. Broadcast communication is possible via events and complex behavior can be described. Conditions can be associated with events and it fires only when a certain condition is achieved. Entry, exit, throughout actions, temporal logic, history states and inter-level transitions are some of the other elements of Harel state charts. Theses state charts concept provided by David Harel lay the basis for current time commonly adopted UML state machines.

UML state machines extend Harel's state machines (state charts) by introducing object-oriented principles. UML inherit most of the state elements from Harel's statechart notation and introduces a few more. Despite many similarities between UML state machines and Harel's state charts, there are some noteworthy differences to be considered while modelling a system which have been discussed in [25].

Notation: A common element in UML and Harel diagrams may have different graphical notation. For example, "A circle surrounding a smaller solid filled circle" represents a final state in UML [26] whereas a circled 'T' is used in the Harel's statechart. Executable Behavior is the most significant category of dissimilarities. A model can be compliable in both modelling notations but its executable behavior different than the expected behavior [26].

The Following table summarizes a comparison between the two types:

Property	**Harel**	**UML**
States Produce Output	Yes	Yes
Transitions Produce Output	Yes	Yes
Sates and Transitions	Yes	Yes
Depth (Composite states, hierarchies)	Yes	Yes
Parallel Substatemachines (orthogonality)	Yes	Yes
Events	Yes	Yes
History, Actions, Timeouts, Delays, Conditions	Yes	Yes
Exit States	No	Yes

Table 1: Comparison between UML and Harel's State Machines

2.5.2 OPC UA and FSM

OPC UA Specification Part 5 – Information Model provides general guidelines to model a system in terms of Finite state machine in OPC UA. The specification provides fundamental ObjectTypes, VariableTypes, RefrenceTypes necessary to model a state machine and instructs how they should be used. In OPC UA FiniteStateMachine is the subtype of StateMachineType which is the subtype of ObjectType. Besides the pre-defined state machines, OPC UA provides the system modeler leverage of defining new types as subtypes of already existing types. Likewise, new references can be defined as childs of hierarchical and non-hierarchical references in the address space. States and Transitions are represented by Objects. Once a StateMachineType has been defined in

OPC UA address space it can be instantiated according to the object-oriented principles. OPC UA provides an abstract way of modelling a state machine system. Semantics of the system can be exposed to the client by defining suitable references [27]. State Machine types defined in the OPC UA Part 5 are shown in the figure below:

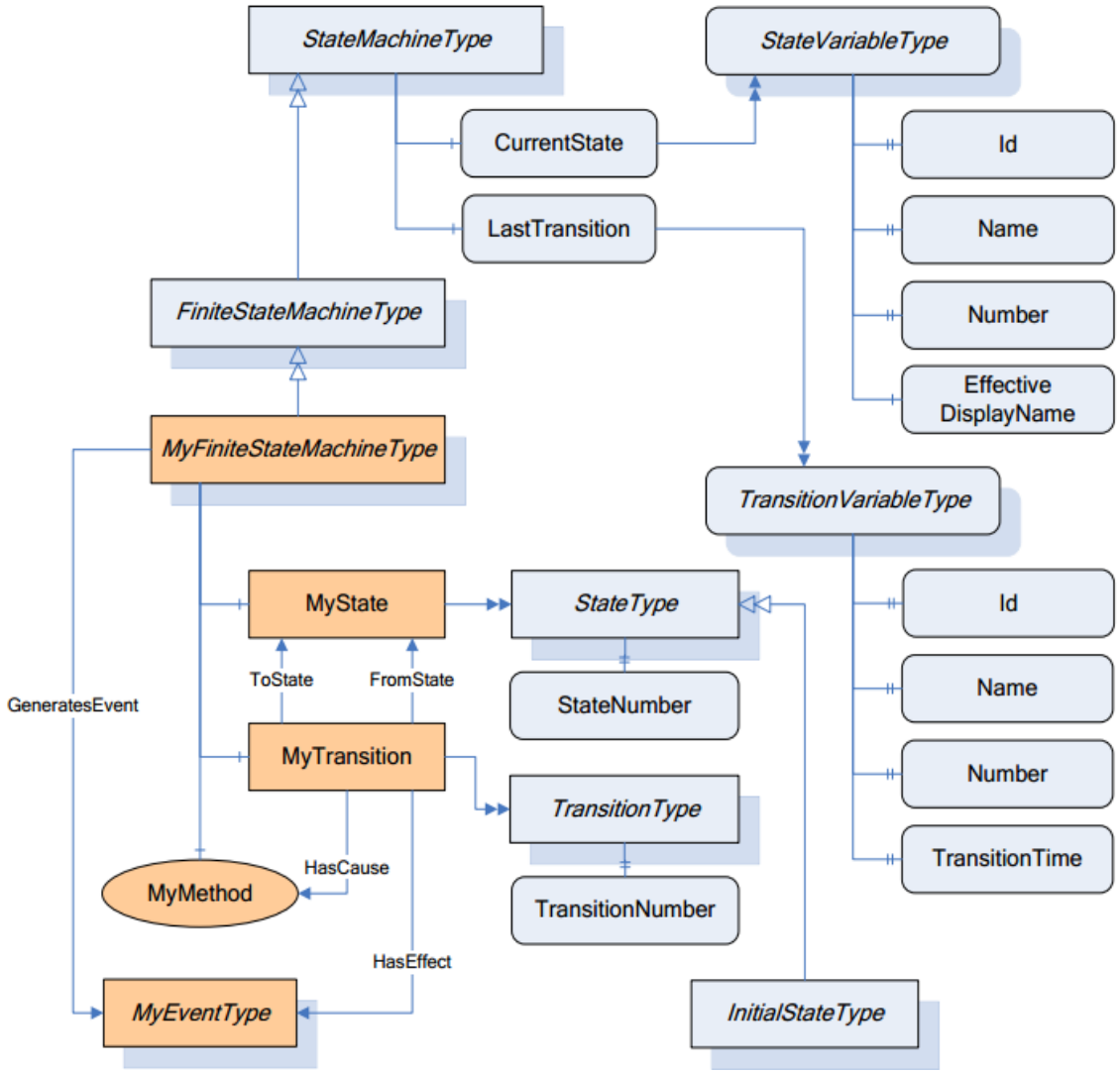

Figure 9. The StateMachine Information Model [28]

Neither the current OPC UA specification nor any companion specification provide the functionality of specifying transition conditions (Guards in terms of UML). [28] introduces OperationType, OperandRefrences and HasGuardRefrences to support guards and associate Arithmetic, Boolean and Logical conditions with Guards. Guards are like transition conditions which needs to be satisfied to fire a condition. The introduction of these operation and reference types in OPC UA server may allow one to map a UML system directly into OPC UA address space.

3. METHODOLOGY & MODELS

This chapter details the approach used for this work and the information models developed for the realization of the concept. First a generalized view of system architecture is presented which is followed by the technologies used and methodology for designing the system. In Section 3.5, the developed information models are described in detail and finally, the linking mechanism of information models at different hierarchical levels is illustrated.

3.1 System Architecture

The information models presented in this work were developed for systems which follows client server architecture as described in Chapter 2. Representation server reads user-defined information models and exposes them to Visualization layer. Visualization layer contain the clients which may subscribe to Representation Server and interact with the exposed information models. For further clarity, the overall system architecture considered for this work can be divided into four parts as shown in Figure 5:

- Visualization Layer,
- Representation Layer,
- Physical Layer, and
- Information Models.

The figure below shows how these four parts interact with each other. Representation layer contains representation server. Representation Server reads the information models defined and instantiated in the information modelling tool and exposes them to User Interface. An information model encapsulates all the information and functionality about a specific physical entity e.g. field device so that its functionality can be exposed to other layers in the system as a service. The physical layer consists of shop floor devices such as sensors, actuators, work cells, robots etc. Information about the physical design, attributes, characteristics and the operations a field device can perform are exposed to the representation layer by the information model of that entity. Once an information model of a device is defined multiple instances of that information model can be added to represent the devices having same functionality. The visualization layer provides the user interface where information models are exposed to the user. User may explore the semantics of the field devices through user interface and may invoke discrete operations as well.

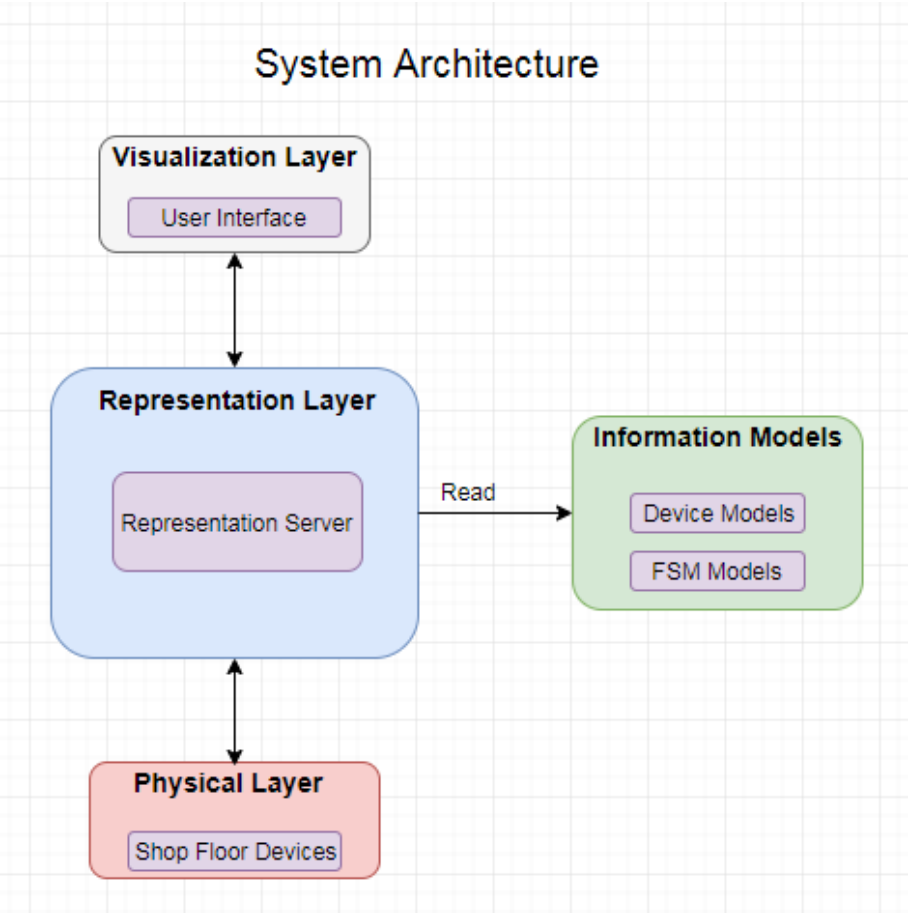

Figure 10. System Architecture

3.2 Visualization Layer

Visualization layer contains the user interface which serve as the client to representation layer in Service Oriented Architecture of Section 3.1. User interface forms the front-end of the system architecture which is used for browsing the internal structure of information models exposed in representation layer. It is also used for observing the run-time change in the values of attributes and parameters. Furthermore, it can be used for tracking the current state of systems and sub-systems on each hierarchical level.

Selection of the client which constitute the visualization layer may vary depending upon the user requirements and technologies used in representation layer. For example, if Resource Description Framework (RDF) has been chosen for representing the resources then a web-based client can be implemented using web-technologies which provides the user interface in web browser. Since the representation framework chosen for the realization of this work is OPC UA as it provides a base state machine model which suits our requirements and can

be extended, an open-source UA client UA Expert has been chosen as the client application. Details about the functionality of UA Expert are detailed in Section 4.

3.3 Physical Layer

The Physical layer may consist of ISA-88 compliant shop floor devices for instance an actuator or robotic arm etc. However, the operations and services of shop floor devices must be able to encapsulate into device information models so that it can be exposed to upper level digital system (OPC UA in this case).

OPC UA provides the capability to expose machine services into OPC UA server as OPC UA methods. So, if machine services are available as discrete operations, it is possible to expose them into OPC UA server and invoke them via OPC UA client. Machine attributes can be exposed as PropertyType. Similarly, machine parameters which can change their values over time can be exposed as BaseVariableType in OPC UA Address Space.

For further clarity, consider a robot which can rotate/move along three axes (X, Y, Z) to locate any point in space. An information model can be developed in OPC UA by mapping the information such as Angle_X, Angle_Y, Angle_Z as BaseVariables, Robot_ID as property. The move operation provided by the robot is mapped as an OPC UA method as shown in Figure 11.

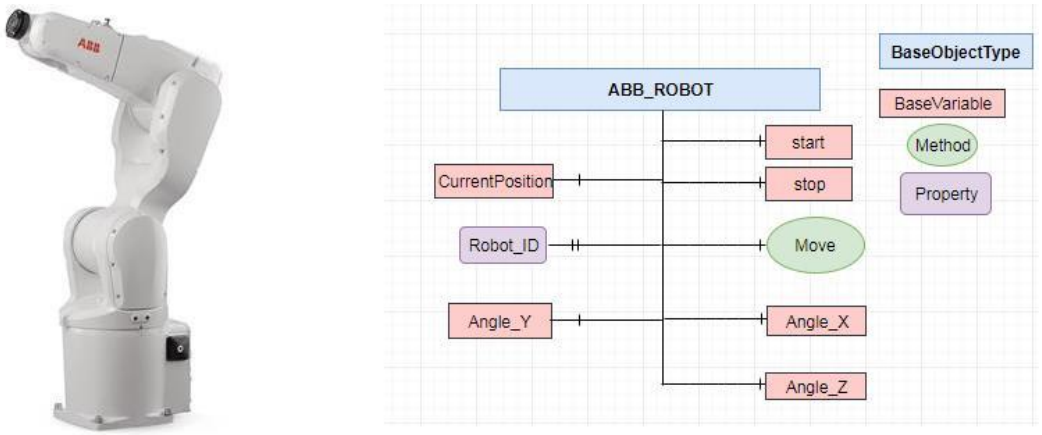

Figure 11. Robot mapped to Information Model

The physical layer used for the realization of this work follows SOA. Discrete operations of physical devices are provided as RESTful services. Device models encapsulates discrete machine level functions and expose them into OPC UA as OPC UA methods so that user may invoke them using OPA UA client UI. Device model may also subscribe to events using

the RESTful services. HTTP POST method is used to subscribe to events. As a result, an event notification is received at the URL provided in the subscription request body.

3.4 Methodology for Building Information Models

To construct the system based on information models proposed in section 3.5 a seven step methodology is presented in this section. Figure 12 gives a visual representation of the process.

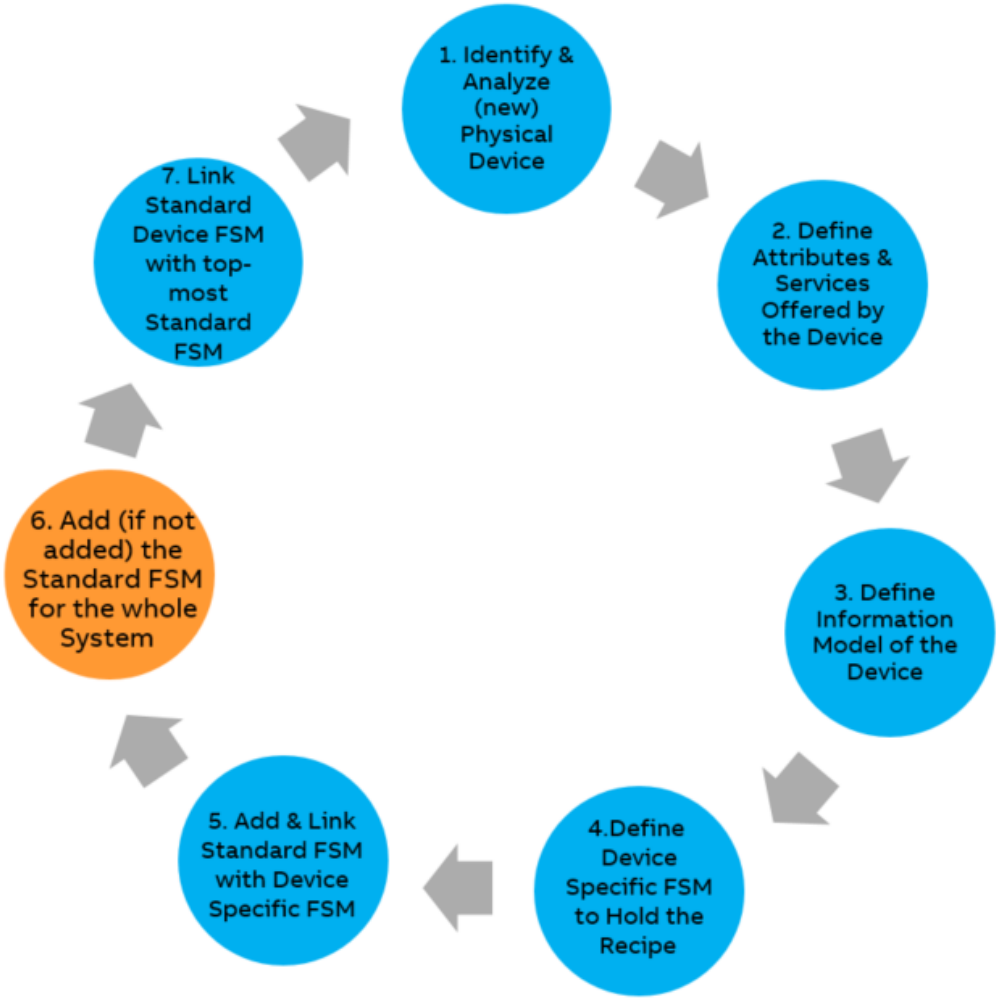

Figure 12. Methodology for building Information Models

Initially in Step 1, the physical system under consideration is analyzed and system components are discovered so that a modular system can be constructed. In Step 2, the attributes which are vital for component working and the services/operations provided by the component are identified and described. Step 3 is the definition of information model which

encapsulates the attributes and services discovered in Step 2. A Java class representing the information model of the physical device is implemented which exposes the service and functionality offered by the device in the representation server. In Step 4, a component-specific finite state machine is defined which holds the recipe for that component. This component-specific FSM is directly linked with the information model of the component and can invoke component services based on the recipe. In Step 5, a standard finite state machine is added on top of component-specific FSM of the device and linked directly with it (Linking mechanism is defined in Section 3.5.4). Standard FSM tracks the component state changes in a standardized terminology which is easily understandable engineers and line operators. To track state of the whole system a standard FSM is added on the top most level in Step 6. In Step 7, standard FSM of each component is linked with the top-most standard FSM which tracks production operation of complete line. The whole process can be repeated for physical system extension while skipping the Step 6.

3.5 Information Models

The system model aroused as a result of composition of state machine models and device models presented in this work is shown in Figure 13. The system is divided into layers. At the bottom is Physical layer composed of series units/machines defined in ISA-88 standard which collectively form a production line. On top of physical layer there is Device Integration Layer composed of configurable device information models added for each of the physical machine. Device information models encapsulate the functionality of shop-floor physical devices and expose them to upper level digital system such that their properties can be explored and their services can be invoked from the digital system.

Above the Device Integration Layer, there is a layer of machine specific finite state machines which contains the recipe for each device and directly interacts with device information models. Each device specific FSM adds behavior to its corresponding physical shop-floor device. Next upper layer is composed of Standard Finite state machines which track the states of physical devices in a standard terminology commonly used in the market, e.g. PackML [29]. Finally, the top most Standard FSM monitors the overall working of the whole production system. User can observe the state changes at different levels of the hierarchy and can issue commands to field devices using the user interface. Information models developed to realize this system are detailed in the remaining part of this chapter.

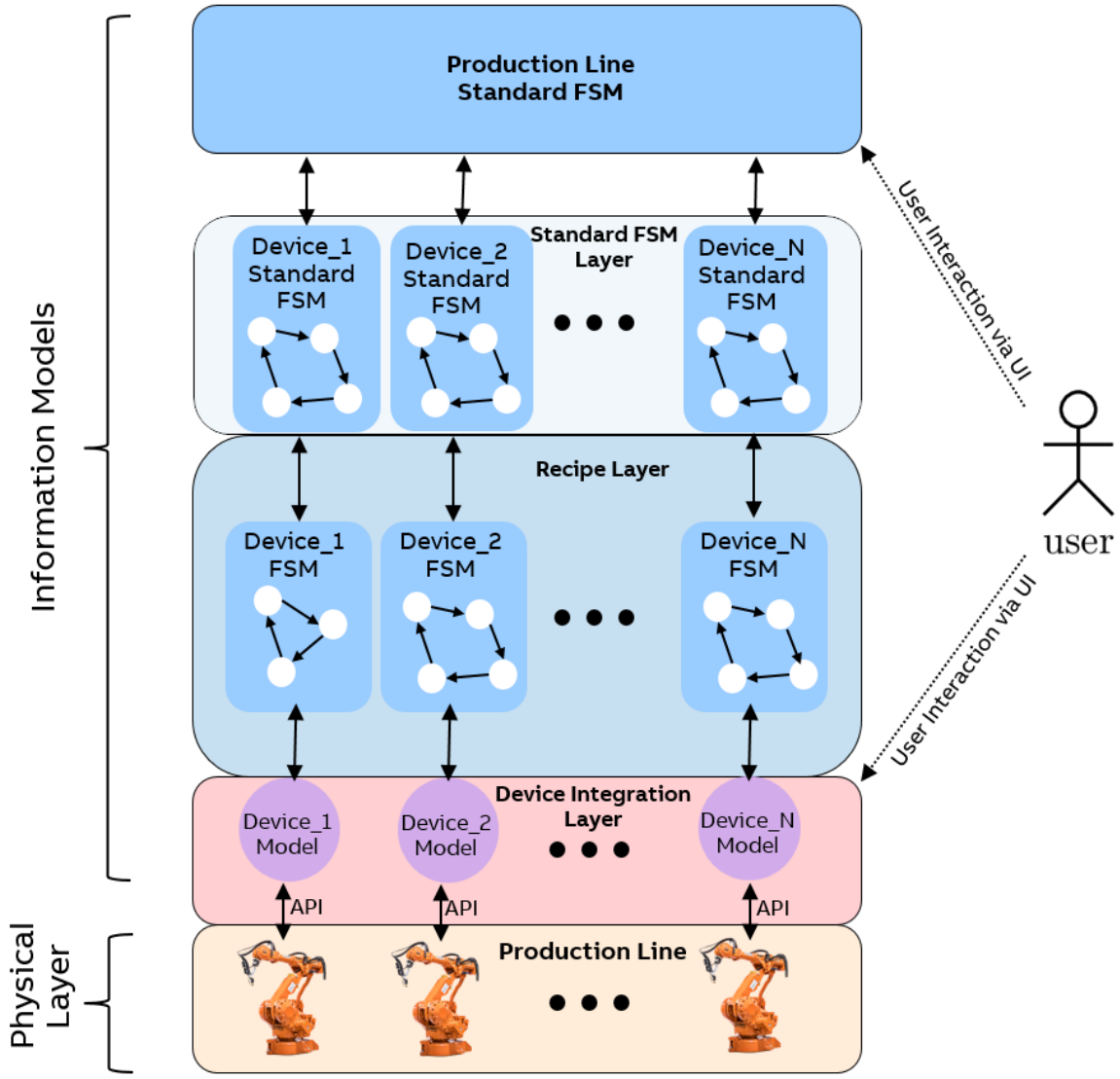

Figure 13. Information Models Hierarchy

3.5.1 PackML State Machine Type

The standard FSM chosen for this work is compliant with OPC UA companion specification for PackML. In Figure 14, the top most standard Finite State Machine and all the other standard Finite State Machines for the individual devices are instances of PackML State Machine Type information model developed for this work. PackML State Machine Type is an extension of Finite State Machine Type developed by the OPC foundation working group. Following Object Oriented rules PackML State Machine Type inherits the states and characteristics of Finite State Machine Type.

PackML State Machine Type has standard states and transitions defined in the base model of PackML standard detailed in section 2.2. At any point in time, system can be in one of these standard states. HasCause Reference is used to link transition condition with a transition. If the condition linked with HasCause Reference is satisfied, the operation linked with HasEffect Reference of that transition is performed and system jumps from one state to another. Figure 3 shows the PackML state machine type exposed in OPC UA address space.

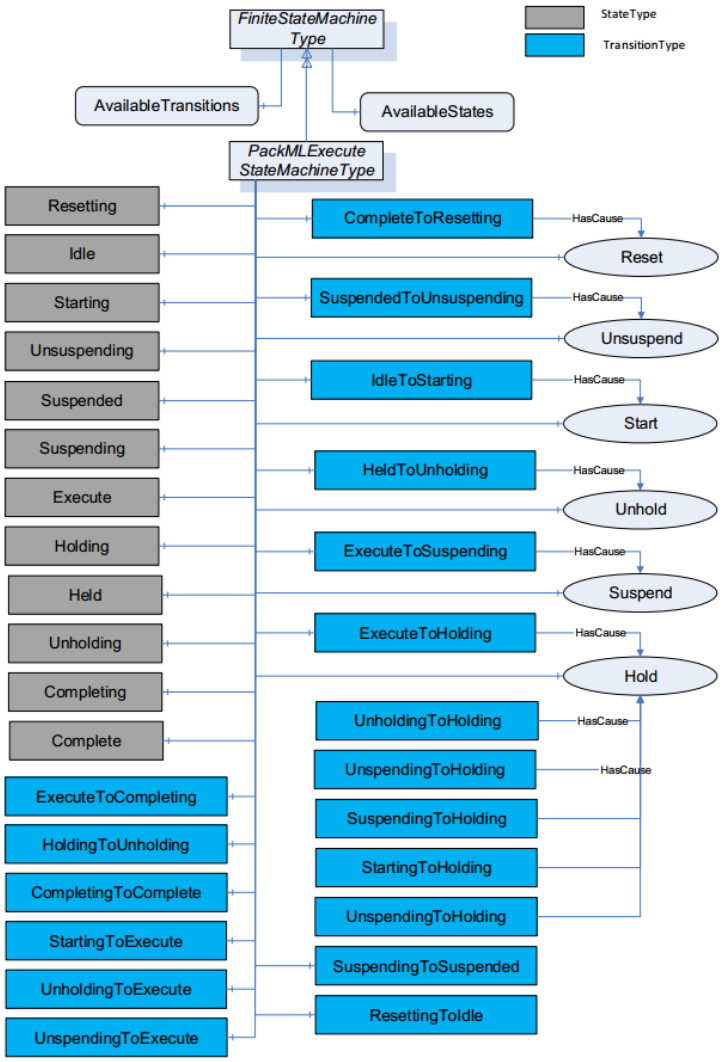

Figure 14. PackML Type [29]

3.5.2 Device State Machine Type

Device State Machine Type is a use-case specific information model defined for machine/unit keeping in view the tasks that machine is supposed to perform. Number of states and transitions of Device State Machine Type can vary depending upon what user wants to

achieve from the machine. For identical machine/units, one Device State Machine can be defined and instances can be added for each of the machine/unit. Each instance of Device State Machine Type directly interacts with the device model of the underlying physical machine/unit. It propagates any messages or commands coming from the upper level standard FSMs to corresponding device model. Likewise, it also passes any messages coming from device models to upper level standard FSMs.

3.5.3 Device Model Type

Device Model Type of unit/machine makes the digital twin of the physical shop floor machines/units. The physical lay-out, characteristics, attributes, operations and other essential information about the machine are encapsulated into the device model of that physical unit/machine and exposed to representation server. As a result, user cannot only explore the internal details of shop floor devices using the user interface but it can also invoke discrete operations of that unit/machine. In the system presented in Figure 1, each device model is directly mapped with the device state machine on top of it.

3.5.4 Information Models TypeDefinition & Instantiation

For reference implementation OPC UA is chosen because it provides a base model which suits well to implement the required state machine model. OPC UA constitutes the representation server which reads the information models as shown in Figure 10 of Section 3.4. Introducing a new information model into OPC UA server is a two-step process:

- TypeDefinition: Implement Java class over OPC UA sdk which exposes the functionality of physical device/model in OPC UA server
- Instantiation: Add instance of TypeDefinition in Address Space through XML schema or Excel tool.

OPC UA Specification Part 5 defines state machine model and its elements for tracking the state changes of system under consideration. State machine model defined in Part 5 can monitor the state changes but it cannot control the physical system. HasCause and HasEffect references are used to add semantics to state machines and no functionality can be associated with them. Unlike standard state machine model, in order to realize this work, state machine model has been implemented in a way that it cannot only observe state changes but also monitor parameters and invoke methods which allows to add controlling mechanism to state machine model. HasCause and HasEffect references have been extended by implementing some logic such that state machine transitions can be linked with transition conditions and specified methods can be invoked.

A FiniteStateMachineType.java class has been implemented by the researchers of ABB which introduces a FiniteStateMachineType (FSMType) in the address space which is extension of FiniteStateMachineType described in OPC UA specification. Transitions of this FSMType can monitor the parameters as well as method calls and capable of invoking methods of other state machines or device models on activation. For realization of this work, PackML State Machine Type (defined in section 3.5.1) and Device State Machine Type (described in section 3.5.2) have been developed by extending FSMType.

A Device Model Type has been introduced by implementing FastoryDevice.java class which encapsulates the essential attributes and services provided by the physical components. Instance of this Device Model Type are added thorough Excel tool. FastoryDevice.java and its instances added via excel are attached in Appendix.

3.5.5 Linking Information Models

Having defined the information models, next thing is to link the information models with each other so that they can inter-operate. Information models can be linked with each other using HasCause and HasEffect References. As described in the previous section, a transition-condition can be linked with a transition through HasCause reference and the operation specified in HasEffect reference of the transition is executed whenever the transition activates. So, models can be linked by HasEffect reference of one state machine transition pointing to the method of another state machine model or device model. Similarly, if a parameter of one state machine is specified in the HasCause reference of transition of another state machine model then the specified parameter will act as the transition condition and transition will be triggered whenever the value of this parameter will be true. Table 2 summarizes the role of HasCause and HasEffect references in this work.

Reference Type	Can Be Linked To	Description
HasCause	Variable/Method	Links transition conditions with the transitions. HasCause of one state machine can point at a variable/parameter of its own or another state machine.
HasEffect	Method	Triggers a method whenever a transition is activated. HasEffect of one state machine can invoke a method of its own or another state machine.

Table 2. Extended Reference Types

To illustrate linking process, consider the information model of ABB_Robot presented in Section 3.3. Suppose we define a state machine information model Robot_FSM to track the state of robot as shown in Figure 15. Robot_FSM has two states i.e. Idle and Running and

two Transitions i.e. IdleToRunning and RunningToIdle as shown in the following figure. 'IdleToRunning' transition of Robot_FSM is linked to 'start' boolean parameter of ABB_Robot through HasCause reference whereas HasEffect is pointing at Move method. Thereby, when value of 'start' parameter will be true IdleToRunning transition will activate and Move method will be invoked meanwhile Robot_FSM will change state from Idle to Running. Likewise, 'stop' parameter linked with 'RunningToIdle' transition will force Robot_FSM to jump from Running to Idle state.

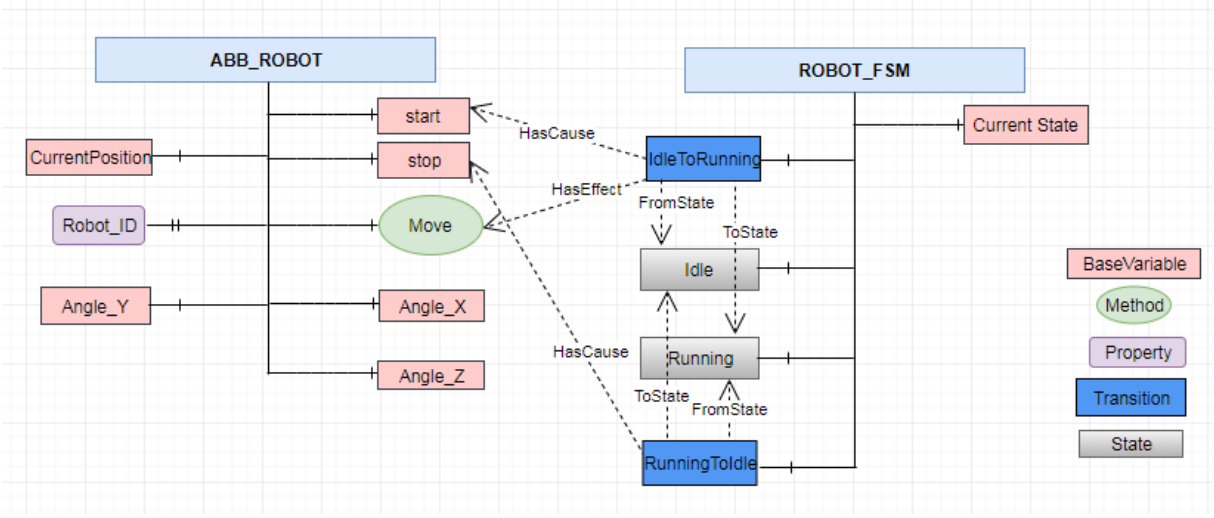

Figure 15. Linking Device Model with State Machine Model

State Machine models can be inter-connected in a similar manner. Consider, for instance, interconnecting device specific FSM with another standard FSM as shown in figure 9. As mentioned in the previous section, transitions can also observe methods and activate upon method call. DeviceSpecific_FSM is linked to Standard_FSM via HasEffect references pointing at the methods of Standard_FSM. IdleToExecute and ExecuteToIdle transitions of Standard_FSM are linked to IdleToExecute and ExecuteToIdle methods via HasEffect references respectively. As a result, when IdleToRunning transition of DeviceSpecific_FSM activates it invokes IdleToExecute method of Standard_FSM which activates IdleToExecute transition forcing Standard_FSM to change its state from Idle to Execute. Hence, one FSM is monitoring the state changes of another FSM and changing its current state accordingly. Since OPC UA is a fully meshed network of nodes, any state machine model can be linked to any other state machine model in the system by pointing at it through HasEffect or HasCause references.

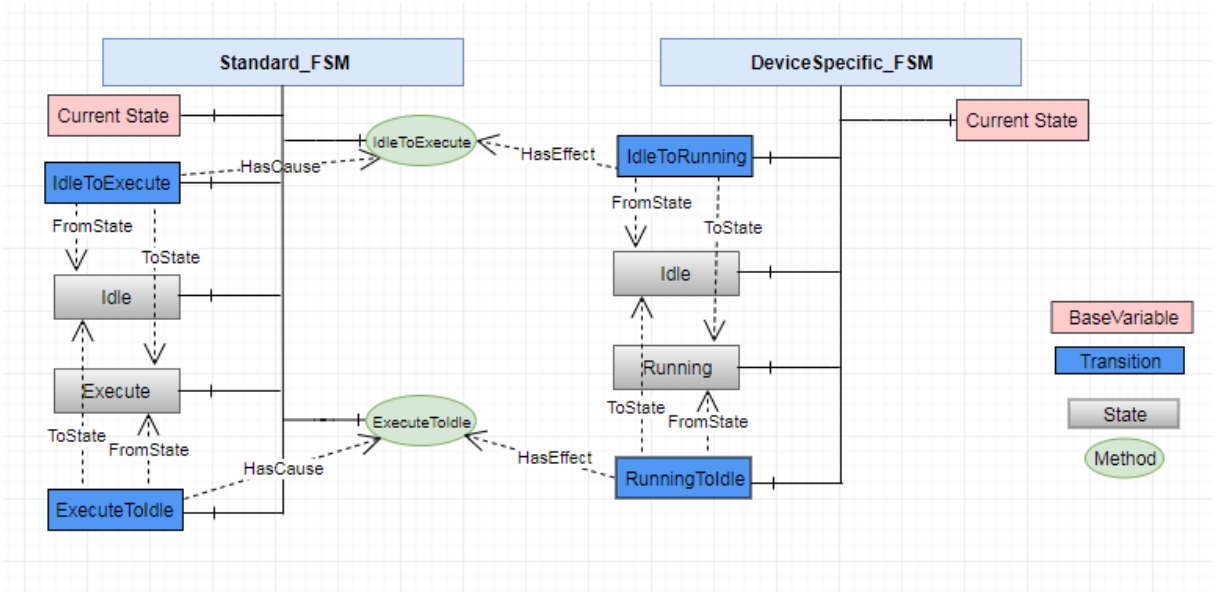

Figure 16. Inter-connecting State Machine Models

4. IMPLEMENTATION

Implementation of the concepts developed in the previous chapter has been detailed in this chapter. OPC UA has been chosen as Information modelling framework which has been described in detail chapter 2. The first section briefly explains the visualization tool used for Address Space browsing and testing purposes. Then the use-case has been described for testing the developed system has been described. In the following sections, Information Models defined for the use-case are detailed and finally, the interaction scenarios among them are discussed.

4.1 Visualization Tool

As described in section 3.2, selection of visualization tool depends on what technology has been chosen for representing information models. Since OPC UA has been chosen as the representation framework, OPC UA client constitute the visualization layer of the proposed System. Many OPC UA clients equipped with different features are available in market such as Ignition Vision, Panel Edition etc. The OPC UA client used for testing and validation purposes in this work is UA Expert. UA Expert is a general purpose graphical OPC UA client provided by Unified Automation. Windows and Linux compatible free version of UA Expert is downloadable from Unified Automation website and comes with following features:

- Data Access View
- Alarms & Conditions View
- Historical Trend View
- DI Information Model Plugin
- OPC UA Performance Plugin

UA Expert allows to connect with an OPC UA server by specifying the server URL. Once connected to the server a user can explore the information models present in that server in the Address Space pane. Figure 17 is a screen shot of UA Expert user interface. Attributes and values of those attributes of any selected node in the Address Space can be seen in the Attribute window. References of that node are shown in the reference window of UA Expert user interface. It is also possible to filter between the forward and revers references using the drop-down menu in reference window. Log Window is displayed in the bottom of the user interface and records the log generated while exploring the UA Address Space. The Centre

pane in the figure is Data Access view. Any node from the Address Space can be dragged and dropped into this DA pane and the runtime changes in the value of the attributes of these nodes can be observed. For further information about UA Expert user is referred to UA Expert documentation page [30].

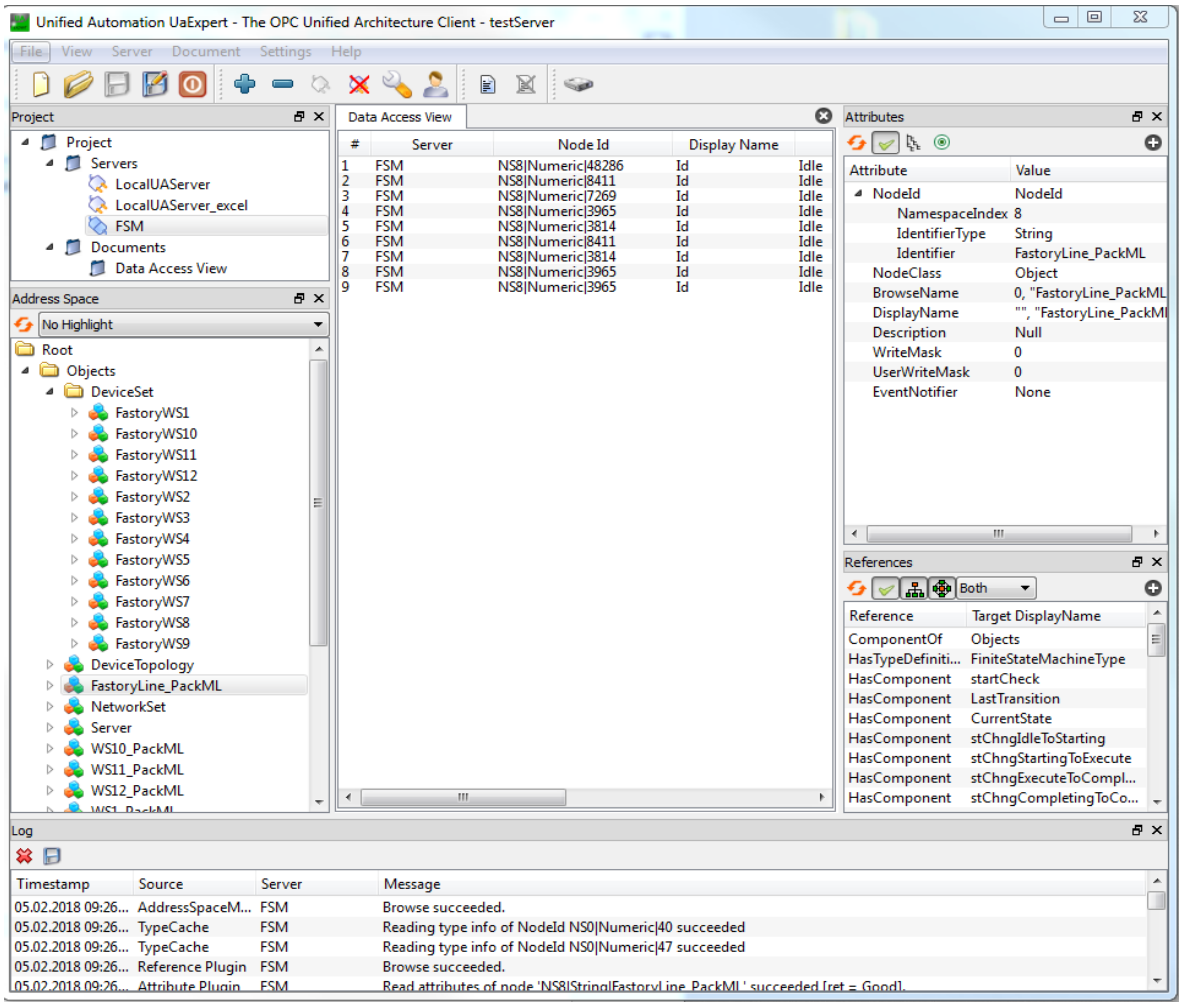

Figure 17. UA Expert User Interface

4.2 Use Case Implementation

For testing the approach developed in chapter 3, information models will be developed for a manufacturing system. Device information models for physical devices will be defined keeping in view the attributes and services provided by the components of manufacturing system. For the purpose of testing, digital twin of the real manufacturing line namely FASTory Simulator, which provides the same functionality as the real line, has been used for this work. Details about the FASTory Line and FASTory Simulator are provided in the following section.

4.2.1 FASTory Line

FASTory Line is a production line installed in TUT FAST (Factory Automation Systems and Technologies) Laboratory to be used as testing bed for research purposes by students and researchers. The line was previously being used in an industry for mobile phone production. The current version of the FASTory Line is a modified version of the actual mobile phone assembly line and emulates the real assembly process by drawing the mobile phone parts on the piece of paper.

FASTory Line has 12 work stations in total shown in Figure 18. All of the work stations are identical except work station 1 (Figure 19_c) and work station 7(Figure 19_b). Work station 1 is used to load/unload paper on the pallet whereas work station 7 is used to load/store pallets on the conveyor. Work stations 2-6 and work stations 8-12 each have a robot equipped with a pen to draw the mobile phone parts on the paper.

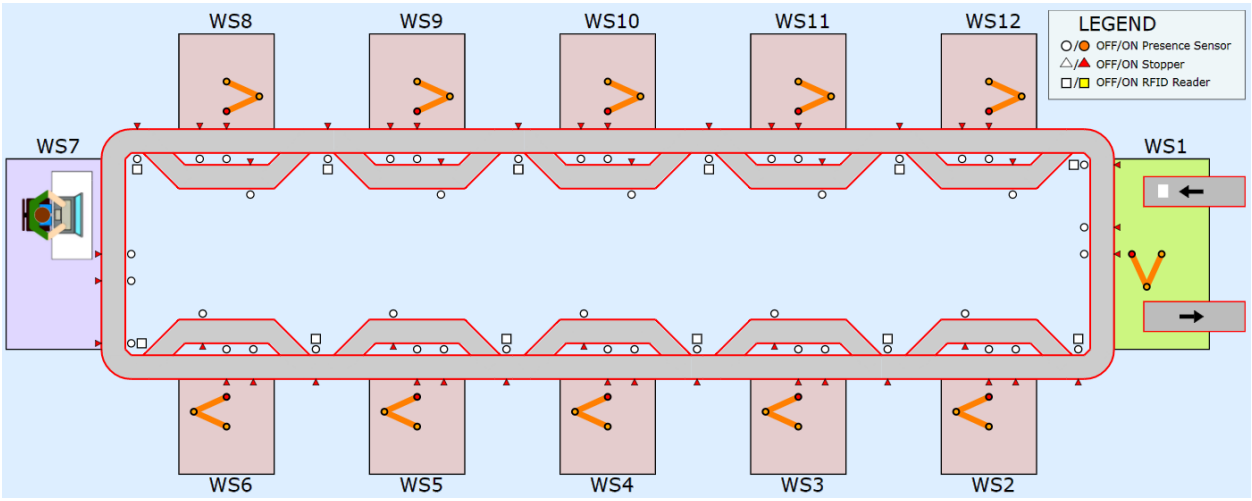

Figure 18. FASTory Line

Every drawing station has 5 zones (Figure 19_a). Each zone has an RFID reader to identify the pallet, a presence sensor to indicate the pallet presence and a stopper to stop the pallet. Pallets enter the work station at Zone 1 and can move to Zone 2 and Zone 3 if there is no pallet already being processed at these zones. Robot may draw at the pallet at Zone 3. Zone 4 is a bypass zone and can be used to either buffer the pallet or bypass Zone 2 and Zone 3 and pass the pallet to the Zone 5 which is the exit point.

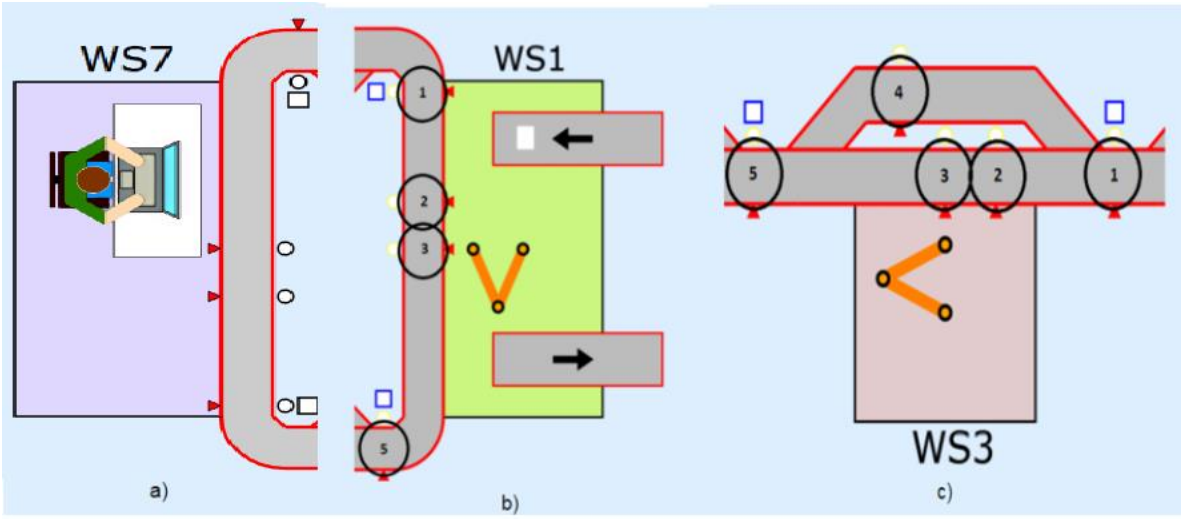

Figure 19. FASTory Work Station Types a) Pallet Loader b) Paper Loader c) Drawing Station

Every drawing station is capable of drawing three drawing three mobile phone components (keyboard, frame, screen) in three different shapes and in three different colors (red, blue, green). These different shapes are shown in Figure 20.

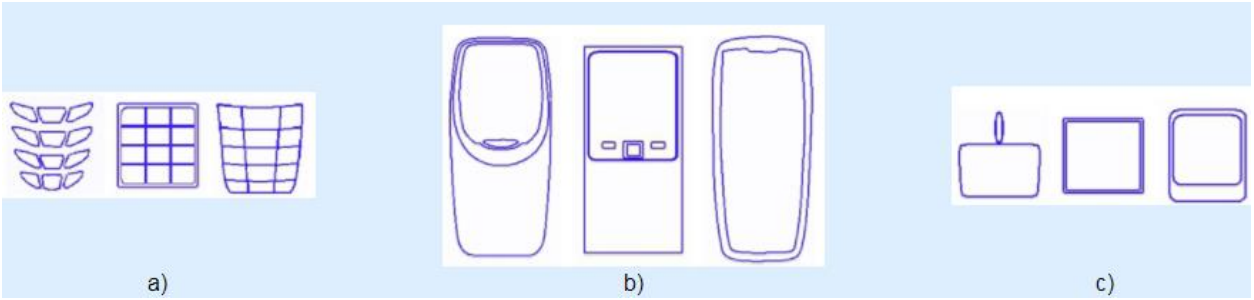

Figure 20. Key Board Styles b) Frame Styles c) Screen Styles

4.2.2 FASTory Simulator

FASTory Simulator is a digital twin of the FASTory Line developed by the researchers working on eScop project. FASTory Simulator is used by the students and researchers at TUT for research and educational purposes for testing their developed applications before deploying it to the FASTory Line. The simulator is developed using the modern web technologies and is compatible with both Windows and Mac operating systems. Simulator can be used for testing purposes to avoid the electrical or mechanical issues which occur while working the physical line. Simulator works in the same manner as the physical FASTory Line and provides similar results as the real FASTory Line. For this thesis work, Simulator has been used to test the developed system models.

FASTory Simulator is a web-based tool developed using the modern web technologies and frameworks such as Node.js, AngularJS, JavaScript etc. The Simulator allows the user to interact with it using RESTful services [31]. In this work, RESTful services has been exposed to OPC UA by developing device model for the work stations. So using the device model user can directly invoke discrete operations via OPC UA client.

Third party client applications can subscribe to the events occurring at the FASTory Simulator using the event subscription mechanism detailed in [31]. Line operations and subscription to the events is achieved using the HTTP POST method. Events are received at the URL provided in the POST method body during the method call. For further detail of the RESTful services provided by FASTory Simulator reader is referred to [32]. Figure 21 explains how RESTful client applications interact with the simulator.

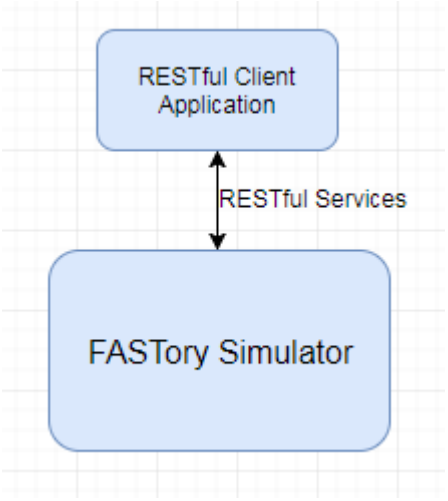

Figure 21. RESTful client interaction with FASTory Simulator

4.2.3 Interaction Model

Figure 13 in Section 3.5 depicts the general system hierarchy where standard FSMs for each device are configured on top of device specific FSM to track the states in a standard terminology. For this reference implementation PcakML state machines described in [29] have been chosen. Device Specific FSMs have been replaced with WS_FSMs and Device Modles with WS1, WS2 etc. The interaction model resulted by these use case specific replacements is shown in Figure 22. Model contains three layers of state machine models and one layer of device models. In the bottom of the model is FASTory Simulator described in Section 4.2.2. The state machine models used in these layers and their role and interaction with each other is discussed in the remainder of this section.

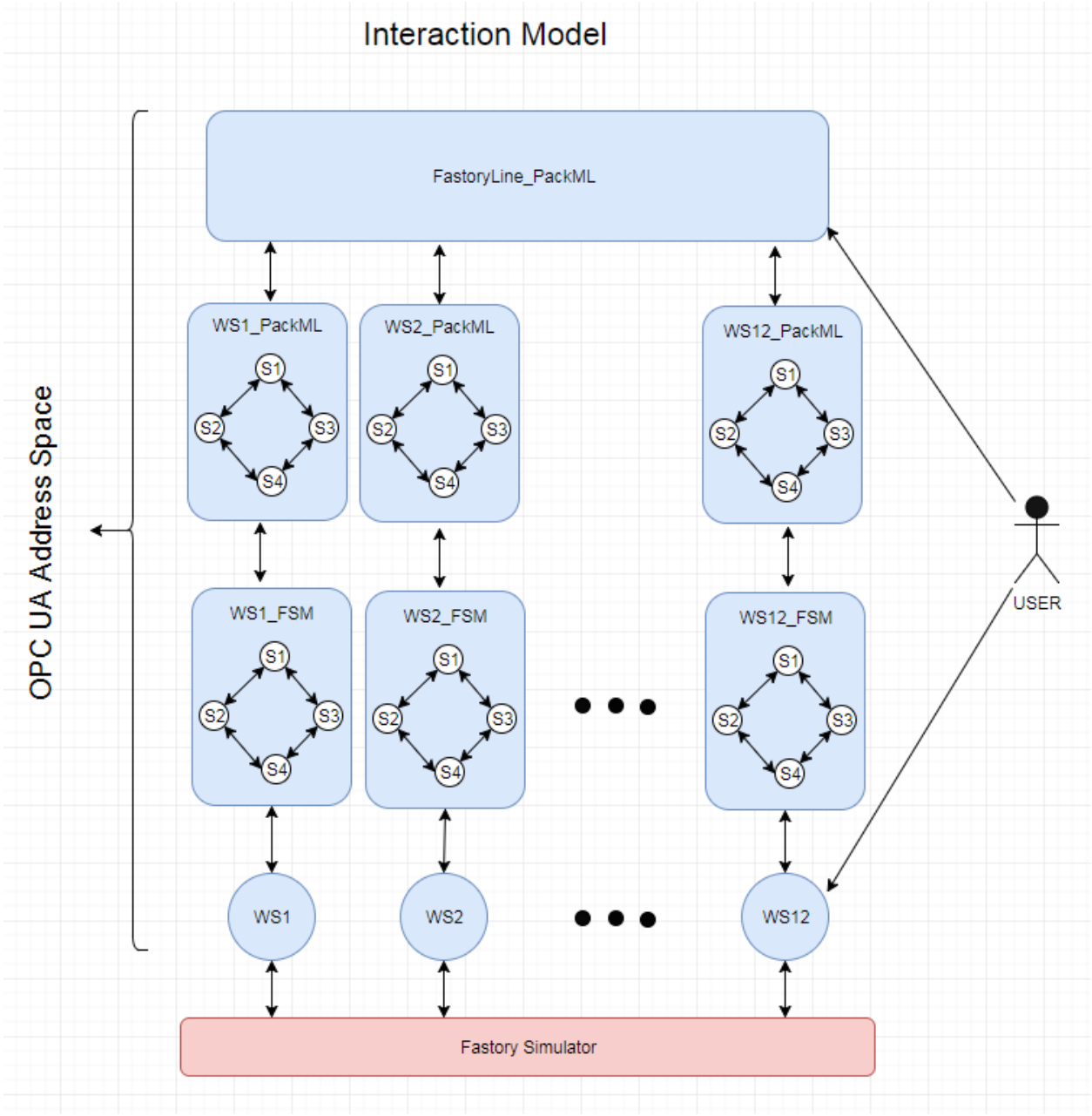

Figure 22. Interaction Model

FASTory Device Model (FASToryWS): FASTory Device Model exposes the line level operations of FASTory Simulator in OPC UA world. It encapsulates the RESTful API to perform the operations which can be performed by a FASTory Work Station. Every FASToryWS also subscribes to the events of the FASTory Simulator and notifies the OPC UA server whenever an event such as Pallet Arrived at Zone 1 or Robot Draw Operation End occurs. Figure 23 showcases device model instances exposed in OPC UA server intractable via OPC UA client.

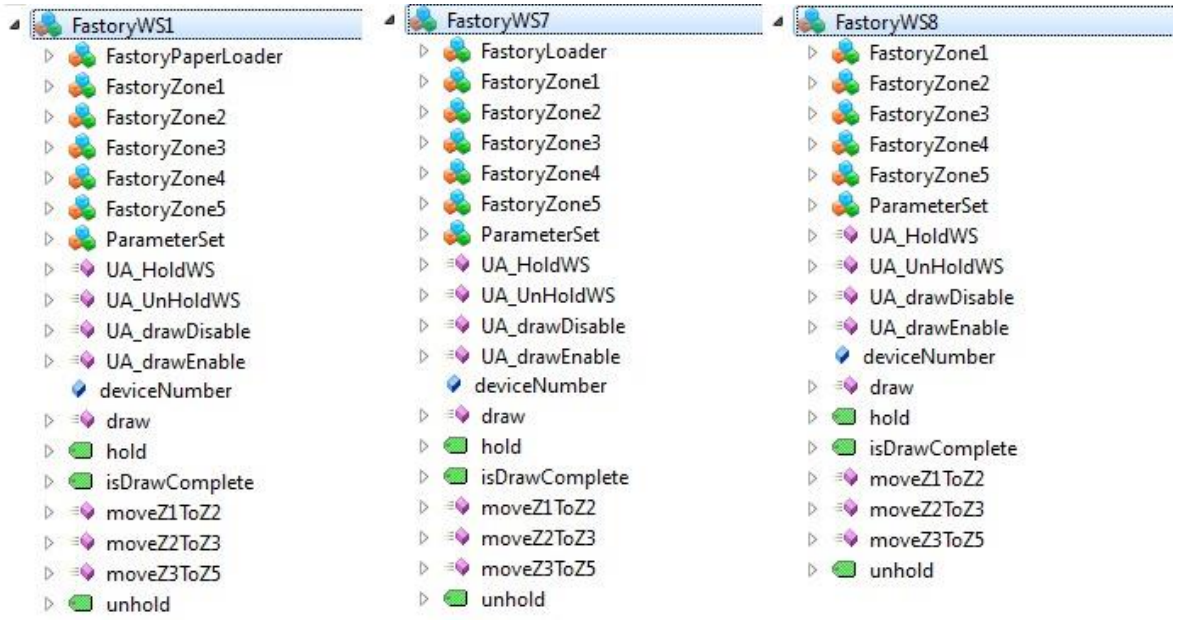

Figure 23. FASTory Device model instances exposed to OPC UA client

As detailed in the previous section, all the work stations of Fastory Simulator are identical except work station 7 and work station 1. Likewise, the device models of work station 7 and work station 1 are also different from other work stations. Work station 7 has additional component FastoryLoader added to the work station 7 device model by implementing FastoryLoader.java class. FastoryLoader is responsible for loading and storing the pallets from work station 7. Similarly, PaperLoader component has been added to work station 1 to load or unload the paper from the pallet on work station 1 by implementing PaperLoader.java class.

Work Station Finite State Machine (WS_FSM): In the system architecture suggested in this work, every device model (FASToryWS) has a Finite state machine on top of it. Every WS_FSM directly interacts with its corresponding device model by utilizing HasCause and HasEffect references provided by OPC UA framework. Transition conditions subscribe to the workstation device model parameters and invoke methods using HasCause and HasEffect references, respectively. The internal structure of WS_FSM is shown on the left in Figure 24 and OPC UA client view of the structure is shown on the right.

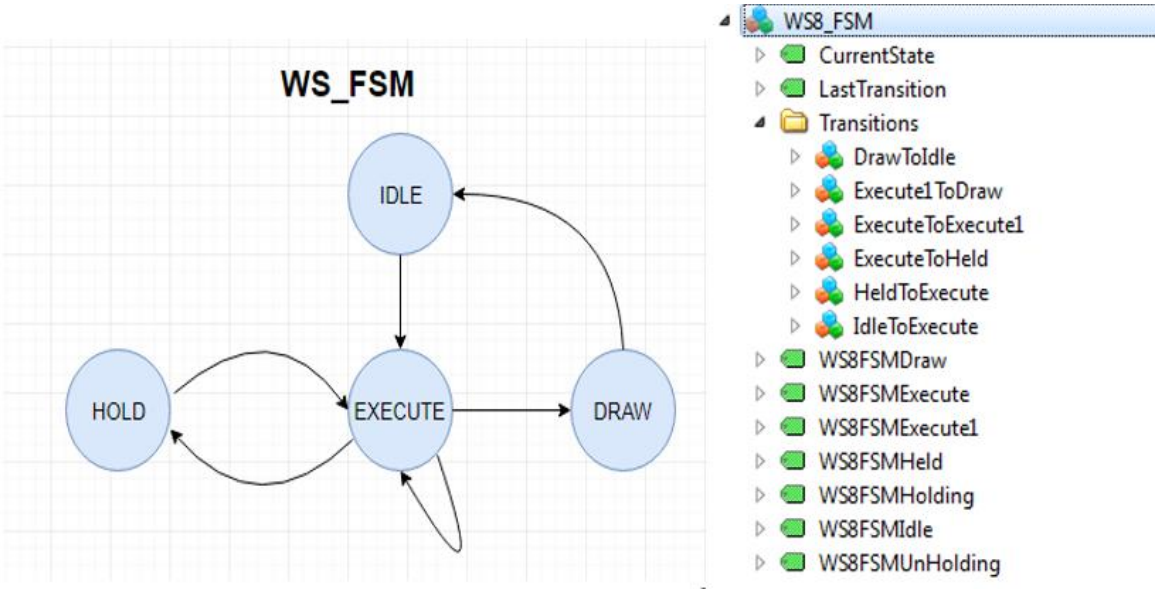

Figure 24. WS_FSM Internal Structure

WS_FSM has four states IDLE, EXECUTE, DRAW and HOLD. State machine can switch between states if the transition conditions between those states are satisfied. Transition conditions are associated with transitions using the HasCause reference and the operation to be performed when a transition condition is satisfied is invoked via HasEffect reference. For example, IdleToExecute transitions has HasCause:palletArrived_Z1 and HasEffect: movePalletZ1ToZ2 associated with it. Thereby, when pallet arrives at zone 1 WS_FSM changes state from Idle to Execute and invokes method moveZ1ToZ2.

Work Station PackML State Machine (WS_PackML): On top of WS_FSM of each work station is an instance of Work Station PackML State Machine (WS_PackML). WS_PackML is in compliance with PackML standard and has seventeen states as described in [6]. Every WS_PackML directly communicates with the WS_FSM under it using the HasCause and HasEffect references and changes the states accordingly. WS_FSM can be use-case specific and may be different for different device types (work stations) but WS_PackML will be same for all the work stations. This addition of WS_PackML allows to achieve a standardized upper hierarchy of the system. System developers may define use-case specific device models and state machines and can map them to the upper PackML state machines. So, from the abstract level, developed system is a combination of many standard PackML state machines.

FastoryLine PackML (Line_PackML): Line_PackML is the upper most state machine that maps to the overall working of the system. WS_PackML for each work station maps directly to Line_PackML. Line_PackML is also in compliance with PackML standard and has seventeen standard states described in the base model of PackML standard [5]. Line_PackML

also has PackML operations such as hold and un-hold specified in the PackML standard which can be executed through OPC UA client.

4.2.4 Interaction Sequence of System Models

A particular production plan is assumed for this use-case implementation to keep the working simple and less complex. Each work station model (device model) subscribes to the zone and robot events of FASTory Line Simulator so whenever an event occurs the respective work station is notified. Events are received in the form of JSON objects at the URL specified in the subscription request. The information enclosed in an event notification is given in the table:

Field Name	Meaning
id	A unique id of the event notification. e.g. 'Z4_Changed', 'OutOfInk' etc.
senderID	ID of the entity sending the notification e.g. 'ROB8', 'CNV10' etc.
lastEmit	Time Stamp at which the even occurred.
payload	Pay Load may contain event specific information such as PalletID.

Table 3. Event Body fields and their meanings

The device model instance of each work station is subscribed to these event notifications and will pass the critical information extracted from the event body to its WS_FSM. User starts the process by loading the pallet at Zone 3 of Work Station 7 of the Simulator either via invoking the loadPallet function from OPC UA client application or by using the FASTory Simulator Interface. WS7 receives the notification form FASTory Simulator and changes the state of WS7_FSM from Idle to Execute and moves the pallet from Zone 3 to Zone 5. Then WS7_FSM changes the state of WS7_PackML from Idle to Execute which in turn changes the state of FastroyLine_PackML from Idle to Execute. When Pallet arrives at Zone 5 of WS7 it changes the state of WS7_FSM Execute to Idle and triggers WS8_FSM by changing its state from Idle to Execute and then WS8_FSM continues its operation. This exchange of messages between the information models has been depicted through the sequence diagram in Figure 25. To put the story in a nutshell, pallet moves from work station 7 until work station 12, then paper is loaded onto the pallet at work station 1. The following work stations from work station 2 to work station 6 either draw the mobile phone part on the paper if the draw function is enabled for the respective work station or just pass on the pallet to next work station. Finally, pallet reaches work station 7 where it is removed from the Line and FastorLine_packML changes its state from Execute to Idle.

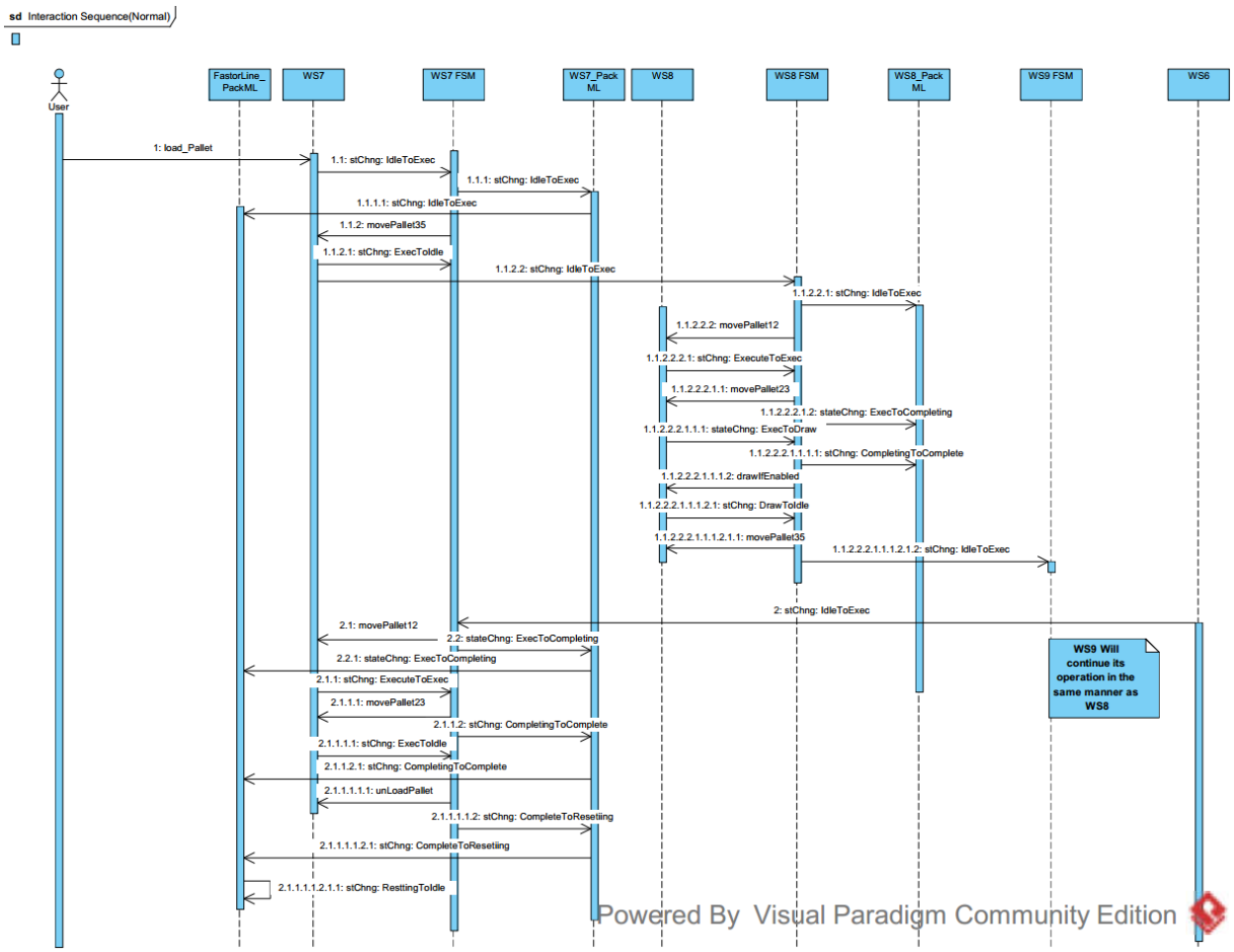

Figure 25. Interaction Sequence of overall operation

4.2.5 Downward Message Propagation

The developed system is vertically integrated in both downward and upward direction. Any message starting from the top propagates downward and ends up at FASTory Simulator. This downward propagation of messages can be illustrated by considering the Top-Down Hold-UnHold scenario. Suppose the system is running and FastoryLine_PackML is in Execute state. User changes the state of FastorLIne_PackML rom Execute to Held by invoking Hold via OPC UA client. FastorLIne_PackML then changes the state of all those WS_PackML state machines from Execute to Idle which were in Execute state. Then each WS_PackML changes the state of WS_FSM from Execute to Idle and the line stops. The sequence diagram for this scenario is show in the Figure 26. Similarly, User invokes UnHold function via OPC UA client and system goes back into the running state.

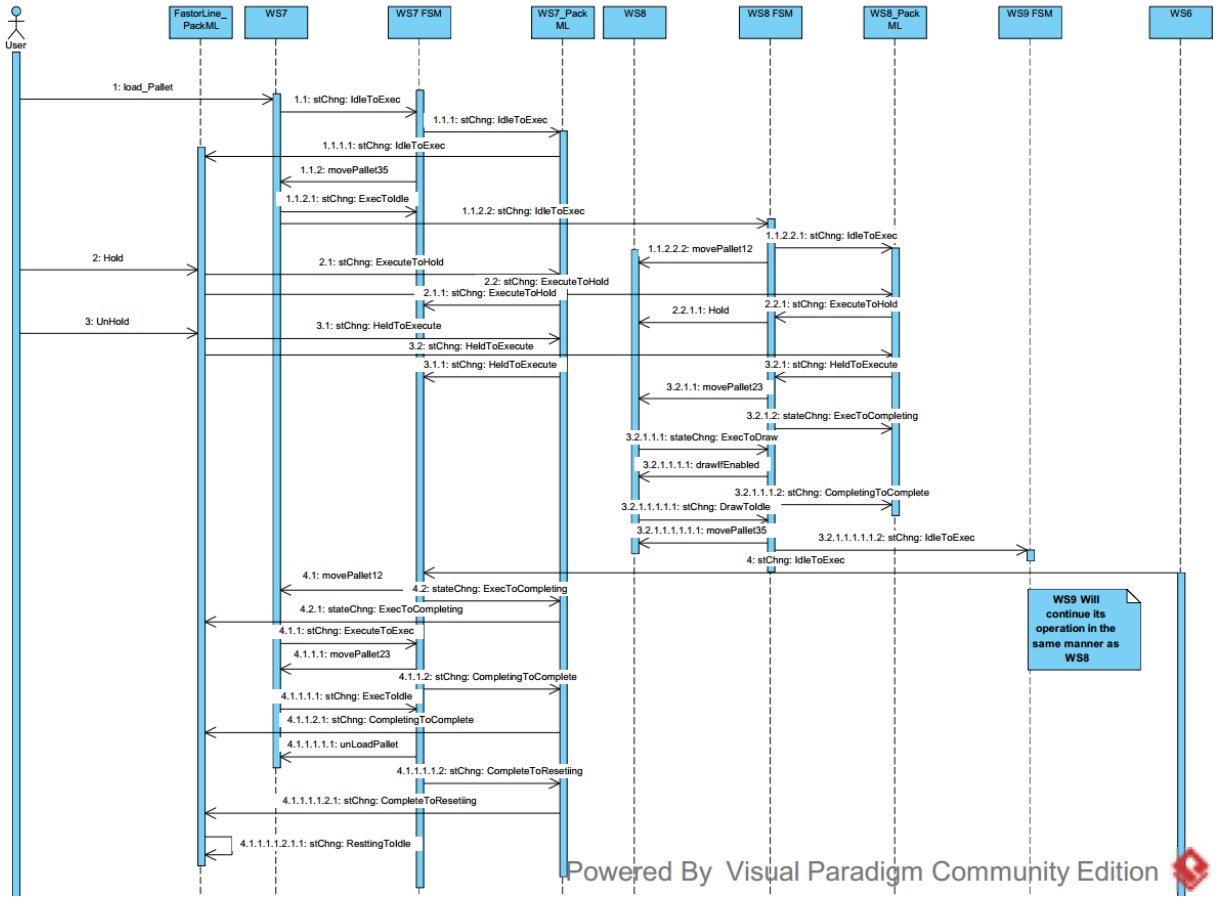

Figure 26. Sequence Diagram of Downward Message Propagation

4.2.6 Upward Message Propagation

Upward propagation of messages can be observed when user invokes HoldWS function of work station device model via OPC UA client. If the WS_FSM of the respective work station was in Execute state then it goes into Held state. WS_FSM then changes the state of WS_PackML on top of it from Execute to Held which further changes the state of top most FastoryLine_PackML from Execute to Held. Similarly, system comes back from Held to Execution state and the messages travel upward in a similar way when user invokes UnHold operation from OPC UA client. The sequence diagram in Figure 27 depicts this upward propagation of messages.

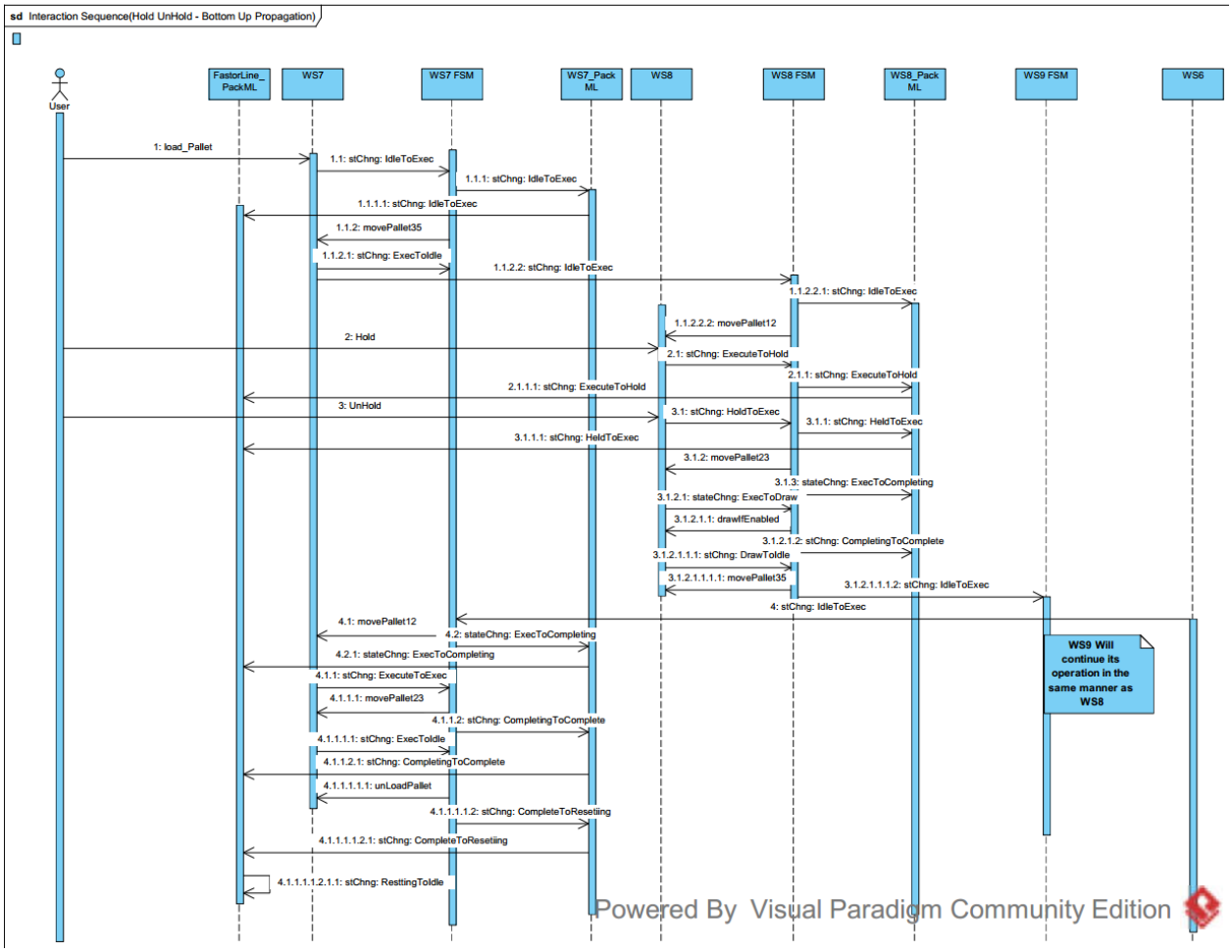

Figure 27. Sequence Diagram of Downward Message Propagation

5. Discussion & Results

The aim of this chapter is to discuss the observation and investigations during the period of this work in the context of the objectives and problems stated in chapter 1. The results obtained over the period of this research work have been discussed. It also explains how the issues and objectives were dealt with during the implementation phase of this work.

The idea behind conducting this thesis work was to aid the flexible production by building a system that will be modular, system building blocks will be configurable, that will be scalable and in which state of the system components could be tracked. For this purpose, information models and their semantics were explored in this work. The challenges offered while linking the information models together and configuring were addressed. Basic state machine model of OPC UA described in [17] was extended to suit the required needs by converting the descriptive HasCause and HasEffect references into logical references. A methodology was presented that outlines the steps to develop the system and shows its scaling capability.

Integrating the services and functionalities offered by field-level devices of manufacturing system into the upper-level digital system has been a hot-topic of research from past few years. Integration techniques mainly depends upon the digital system infrastructure that either it is service-oriented or non-service-oriented. For the implementation of this work OPC UA was chosen as upper-level digital system and has service-oriented infrastructure. Discrete services offered by the field devices were encapsulated into device information model following the Device Integration (DI) Model companion specification. Hence, all the relevant properties and services offered by the field devices were exposed as a UA models representing physical devices into digital system.

OPC UA state machine model has been used in this work to track the states of system and sub-systems but basic state machine model was not completely suitable. There was no mechanism available to specify transition conditions in base state machine model described in Section 2.5.2. The issue has been addressed by extending the basic state machine model such that state machine transitions are activated based on the parameter values they are attached to. Furthermore, state machines can also invoke methods of their own or of other state machines and device models as described in Section 3.5.5. This development paved path for the interaction mechanism between building blocks of the system.

Flexible production systems that are scalable and supposed to adapt to line-level changes occurring due to the changes in production demand or product requirements should be capable of efficient field-devices configuration. Configuration management on field level can

be time consuming and exhaustive task. In this work, field-devices can be configured from the upper level digital system. As a result, system can be scaled with ease and new devices can be configured in shot-time and with less effort. However, the current work allows manual configuration of device information models which is still time consuming. More efficient ways of configuration management can be researched in future studies.

Another objective set during the early phases of this work was to develop reusable information models. Reusable code is easier to maintain and understand, shorter development time and results into time and cost efficiency. Likewise, to gain the advantages of reusability information models developed in this work are reusable. A type definition can be defined for one kind of devices/state machines and then instances can be created as per the requirement. Hence, considerable amount of time can be saved and Address Space becomes easier to maintain.

6. Conclusion and Future Prospects

This chapter summarizes the research work done in this thesis by presenting the achievements while implementing the use-case and finally, gives an outlook of the possible future research prospects.

6.1 Achievements

This thesis meets the requirements set during the initial phases and accomplishes the objectives listed in chapter 1. The developed system is composed of modular state models and system state can be tracked on different hierarchical levels. State models are stackable horizontally as well as vertically based on the system topology. Interaction mechanism between the state machine models has been developed that allows to attach transition logic with them and also allows to invoke methods/procedures of their own and other state machine models as described in Section 3.5.5.

The developed system possesses scaling capability. Additional components can be added into the system to address the higher production demand and extra components can be removed as per user requirements. A methodology has been presented in Section 3.4 for this purpose. Field devices can be configured in the developed approach through the upper level digital system which leads to faster and easier scaling capability. This configuration process can be done “on-the-fly” which reduces the downtime, ideally, to zero.

6.2 Future Prospects

The state machine models based approach developed in this work for tracking the state of shop-floor devices and components was implemented for a real production line that demonstrates the assembly of mobile phones and tested using the simulator of that production line i.e. FASTory Simulator. Nevertheless, following prospects can be considered as proposals for future work.

In the developed approach, device information models of physical devices are configured manually by the user. The configuration process requires human effort and becomes time-consuming especially if the devices to be configured are large in number. Based on the semantics information of the physical devices, automatic generation of information models can be studied and researched as future work. Mechanisms/middleware can be developed for this purpose and thus considerable human effort required for configuration can be saved.

Due to time limitations the developed concept has been tested on one use-case, i.e. FASTory Line and, although, the implementation results are justifying the study objectives listed in chapter 1, more use cases can be implemented for an even concrete proof of concept.

Another potential future work is the addition of Health State Machine that can monitor the health state of the field device. Such a health state machine can be used to expose the health information to upper-level digital system and can also be integrated with the Standard FSM e.g. PackML for the implemented use-case. Hence, the appropriate precautionary actions can be taken before the system breakdown.

7. REFRENCES

[1] Deloitte, Industry 4.0: Challenges and Solutions for the Digital Transformation and Use of Exponential Technologies, Report, Deloitte AG. 2015

[2] O. Sauer, "Information Technology for the Factory of the Future – State of the Art and Need for Action," Procedia CIRP, vol. 25, pp. 293–296, 2014.

[3] S. Tewari, M. Mishra, (2012). "The Impact of ICT on Manufacturing Industry: An Empirical Analysis." Presented on 2012 International Conference on Communication Systems and Network Technologies.

[4] Vogel-Heuser, B.; Kegel, G.; Bender, K.; Wucherer, K.: Global information architecture for industrial automation. atp 1-2.2009, p. 108-115.

[5] "ISA-TR88.00.02: Machine and Unit States: An Implementation Example of ISA-88," International Society of Automation, August 2008.

[6] "PackML Unit Implementation Guide," The Organization for Machine Automation and Control, March 2017.

[7] "A Graphical Representation of PackML States," July 2017. [Online] Available: https://commons.wikimedia.org/wiki/File:PackML_State_Model.png

[8] "ISA88, Batch Control- ISA", Isa.org, 2018. [Online]. Available: https://www.isa.org/ISA88/. [Accessed: 26- Mar- 2018].

[9] J. Parshall, L. Lamb, Applying S88 - Batch control from a user's perspective. ISA - The Instrumentation, Systems and Automation Society, 2000

[10] "ANSI/ISA-88.01-1995, Batch Control, Part 1: Models and Terminology," the Instrument Society of America, October 1995.

[11] P. Ceruzzi, A History of Modern Computing, 2nd ed. The MIT Press, 2003.

[12] "Classic", OPC Foundation, 2018. [Online]. Available: https://opcfoundation.org/developer-tools/specifications-classic. [Accessed: 26- Mar- 2018].

[13] OPC Foundation. OPC Unified Architecture Specification, Part 4: Services. Release 22.11. 2017.

[14] W. Mahnke, S.-H. Leitner,, “OPC UA – Service-oriented Architecture for Industrial Applications,” in , ABB Corporate Research Center, 2007.

[15] W. Mahnke, S.-H. Leitner, and M. Damm, OPC Unied Architecture. Berlin: Springer Berlin Heidelberg, 2009. [Online]. Available: http: //www.springerlink.com/index/10.1007/978-3-540-68899-0

[16] "Unified Architecture", OPC Foundation, 2018. [Online]. Available: https://opcfoundation.org/developer-tools/specifications-unified-architecture. [Accessed: 26- Mar- 2018].

[17] OPC Foundation. OPC Unified Architecture Specification, Part 5: Information Model. Release 22.11. 2017.

[18] OPC Foundation. OPC Unified Architecture Specification, Part 1: Overview and Concepts. Release 22.11. 2017.

[19] OPC Foundation. OPC Unified Architecture Specification, Part 3: Address Space Model. Release 22.11. 2017.

[20] Luth, Jim. OPC-UA Architecture. Presentation slides, OPC Unified Architecture DevCon, 2007.

[21] W. Newman, "A system for interactive graphical programming", Proceedings of the April 30--May 2, 1968, spring joint computer conference on - AFIPS '68 (Spring), 1968.

[22] G. Mealy, "A method for synthesizing sequential circuits", The Bell System Technical Journal, vol. 34, no. 5, pp. 1045-1079, 1955.

[23] E. Moore, "Gedanken-experiments on sequential machines. Automata studies", The Journal of Symbolic Logic, vol. 23, no. 01, p. 60, 1958.

[24] D. Harel, "Statecharts: a visual formalism for complex systems", Science of Computer Programming, vol. 8, no. 3, pp. 231-274, 1987.

[25] M. Crane and J. Dingel, "UML vs. classical vs. rhapsody statecharts: not all models are created equal", Software & Systems Modeling, vol. 6, no. 4, pp. 415-435, 2007.

[26] OMG. UML 2.0 superstructure specification. Technical Report ptc/04-10-02, Object Management Group, 2004.

[27] OPC Foundation. OPC Unified Architecture Specification, Part 5: Information Model. Release 22.11. 2017.

[28] T. Friihwirth, F. Pauker, A. Fernbach, I. Ayatollah, W. Kastner and B. Kittl, "Guarded state machines in OPC UA", IECON 2015 - 41st Annual Conference of the IEEE Industrial Electronics Society, 2015.

[29] “OPC Unified Architecture for PackML,” OPC Foundation, June 2017. (Not Finalized – Work in progress)

[30] UaExpert", Unified Automation, 2018. [Online]. Available: https://www.unified-automation.com/products/development-tools/uaexpert.html. [Accessed: 22- Mar- 2018].

[31] W. Mohammed, ‘Encapsulation Of MES Functionalities As RESTful Web Services For Knowledge-Driven Manufacturing Systems’, Master Thesis, Tampere University of Technology, 2015.

[32] “FASTory Simulator (eScop)” [Online]. Available: http://escop.rd.tut.fi:3000/. [Accessed: 22-03-2018].

APPENDIX A -- FastoryDevice.java

```
package com.abb.model.types.festory;

import java.io.IOException;

import java.util.HashSet;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import com.abb.model.types.*;

import com.abb.nodefactory.*;

@UANode(typeNodeID = "nsu=org.basys;s=FastoryDevice")

public class FastoryDevice extends DeviceType implements UAEventHandler {

	private Logger logger = LoggerFactory.getLogger(this.getClass());

	@UADoNotClone

	private FastoryMethodFactory fastory = new FastoryMethodFactory();



	public static boolean start = false;

	public static boolean isHold = false;

	public static boolean isSuspend = false;

	public boolean startWS = false;

	public boolean draw;

	// Empty constructor for initialization via OPC UA

	public FastoryDevice() {

	}

	@UAReference
	public BaseVariable hold = new BaseVariable("hold", false);

	@UAReference
	public BaseVariable unhold = new BaseVariable("unhold", false);

	@UAReference
	public BaseVariable isDrawComplete = new BaseVariable("isDrawComplete", false);

	@UAReference
	(uaImport = true, refType = UAReferenceTypes.hasComponent)
```

```
public HashSet<FastoryZone> zones;

@UAReference
public Integer deviceNumber = -1;

private Logger LOG = LoggerFactory.getLogger(this.getClass());

@UADoNotClone
private boolean registered = false;

@UAMethod
public void UA_drawEnable() {

    LOG.info("UAMethod UA_drawEnable");

    draw = true;

}

@UAMethod
public void UA_drawDisable() {

    LOG.info("UAMethod UA_drawDisable");

    draw = false;

}

@UAMethod
public void UA_HoldWS() throws InterruptedException {

    LOG.info("UAMethod UA_HoldWS");

    this.unhold.value = false;

    this.hold.value = true;

    UANodeFactory.update(this);

}

@UAMethod
public void UA_UnHoldWS() throws InterruptedException {

    LOG.info("UAMethod UA_UnHoldWS");

    this.unhold.value = true;

    this.hold.value = false;

    UANodeFactory.update(this);

}

public boolean hasPallet(int zone) {

    try {
```

```
                LOG.info("check hasPallet:" + zone + " / " + deviceNumber);
                return fastory.hasPallet(zone, deviceNumber);
        } catch (IOException e) {
                e.printStackTrace();
        }
        return false;
}
public void move(Double from, Double to) {
        try {
                fastory.MoveCNV(this.deviceNumber, from, to);
        } catch (IOException e) {
                e.printStackTrace();
        }
}
public void draw(Double val) {
        if (draw) {// drawEnable from UA client
                try {
                        fastory.DrawFrame(this.deviceNumber);
                } catch (IOException e) {
                        e.printStackTrace();
                }
        } else {
                LOG.info("Draw not enabled for WS " + this.deviceNumber);
                isDrawComplete.value = true;
                UANodeFactory.update(this);
                LOG.debug("isDrawComplete /" + this.isDrawComplete);
        }
}
```

```
@Override
public void handle(UAEvent event) {

    if (!registered) {

            if (this.deviceNumber != -1 && this.deviceNumber != 1 && this.deviceNumber != 7) {

                try {

                    LOG.info("DrawEvent Registered at WS :" + this.deviceNumber);

                    FastoryMethodFactory.subscribeToDrawEndEvent(this.deviceNumber);

                    FastoryMethodFactory.subscribeLowInkLevelEvent(this.deviceNumber);

                    FastoryMethodFactory.subscribeOutOfInkEvent(this.deviceNumber);

                    this.registered = true;

                } catch (IOException e) {

                    LOG.warn("connection issue:" + e.getMessage());

                }

            }

    }

    if (event instanceof UAUpdateEvent) {

            UAUpdateEvent updateEvent = (UAUpdateEvent) event;

            if (updateEvent.fieldName == null) {

                if (UANodeRegistry.isInstance(this)) {

                } else {

                }

            } else {

                logger.info("value at FastoryWS" + this.deviceNumber + " was modified:" +
updateEvent.fieldName

                        + " from:" + updateEvent.oldValue + " to:" +
updateEvent.updatedValue);

            }

    }

}

}
```

APPENDIX B --Information Model Instances

A	B	C	D	E	F
Namespace	**Preferred_Name**	**Type**	**References**	**Value**	**DataType**
org.basys.Fastory.Line	**FastoryWS7**	<org.basys.Fastory.Device.FastoryDeviceType>			
	FastoryWS7/deviceNumber	<Property>		7	
	FastoryLoader	<FastoryLoader>	ComponentOf:FastoryWS7		
	FastoryWS8	<org.basys.Fastory.Device.FastoryDeviceType>			
	FastoryWS8/deviceNumber	<Property>		8	
	FastoryWS9	<org.basys.Fastory.Device.FastoryDeviceType>			
	FastoryWS9/deviceNumber	<Property>		9	
	FastoryWS10	<org.basys.Fastory.Device.FastoryDeviceType>			
	FastoryWS10/deviceNumber	<Property>		10	
	FastoryWS11	<org.basys.Fastory.Device.FastoryDeviceType>			
	FastoryWS11/deviceNumber	<Property>		11	
	FastoryWS12	<org.basys.Fastory.Device.FastoryDeviceType>			
	FastoryWS12/deviceNumber	<Property>		12	
	FastoryWS1	<org.basys.Fastory.Device.FastoryDeviceType>			
	FastoryWS1/deviceNumber	<Property>		1	
	FastoryPaperLoader	<FastoryPaperLoader>	ComponentOf:FastoryWS1		