Formal Methods for Enterprise Application Integration

Sivasankararao Kotha, T. V. Gopal

sivasankar.res@gmail.com; gopal@annauniv.edu

(1) Department of Computer Science and Engineering, Anna University, College of Engineering, Guindy Campus, Chennai, India

***Abstract*— In the past few years, there is an enormous growth in enterprise computing. Large enterprises have more than thousands of applications that work in tandem to support enterprise business processes. The enterprises typically adopt internet technologies. Enterprise Application Integration (EAI) provides the option to integrate the different applications to enhance the functionality and sharing of the information among different applications. Integration increases the automation of business process and significantly reduces the redundancy of data functionality and the IT infrastructure cost like maintenance, management and operational.**

Presently, the Formal Methods use mathematical models for verification and analysis of the Software life-cycle. In this paper, the focus is on the possible formal levels of Enterprise Application Integration and proposed a theoretical, computational and process oriented collaboration model for integrating formally. Explained different formal method technologies for EAI. This research will address and propose solutions of the different enterprise applications while integrating them in terms of mathematical formalism.

***Keywords*—Autonomous Robotic System, Complex and Control System, Enterprise Application Integration, Formal Methods, Formal specification, Formal Verification, Process Model, Petri Net, Z Specification Language.**

I. Introduction

The traditional meaning of enterprise is the company or organization for commercial purpose. In the IT world, the meaning of the enterprise is different and is often fuzzy [19]. The enterprise is an organization that follows some characteristics like size or location of the company, software, and hardware applications which are used for the enterprises, and management of the enterprise.

Integration is not a new methodology in the field of Information and Communication technology and has been used for the past 50 years. The early systems were designed and developed without concerns of change in technology and the requirements were estimated based on the context. Technologies such internet and e-commerce are mandating the integration of the internal systems to satisfy the customer's needs. The data and functionalities of different applications need to be shared. Hence, there is a need to integrate the applications even if they run on different platforms for different purposes. EAI is one of the emergent frameworks to facilitating the integration of different applications.

Formal methods are based on mathematical techniques and notations used for describing, analyzing and specification of properties in software systems. These mathematical techniques are based on predicate logic, set theory, relation, functions and graph theory [4]. Based on the model of software in the life cycle, we will get a result from requirement analysis in the form of informal language. The specification is the phase of transforming the informal to formal which satisfies the requirements. The phase from design to program corresponds to the development of the code. The verification and validation are the two basic principles in the development of the system. There are many tools and methods for verification and validation of the system. The formal verification, formal specification languages, fundamental concepts of model checking and theorem proving are discussed in this paper.

In this paper, Section 2 gives the details of EAI. Section 3 provides the different types of EAI, while Section 4 and Section 5 gives the fundamentals of the formal methods and their impact in software engineering. Details of Formal Methods and some industrial scenarios are discussed in Section 6 and Section 7. Section 8 explains the case study with one of the specification language. The proposed work of

research is discussed in section 9. Finally, concluded the paper in section10.

II. Enterprise Application Integration

Every Enterprise strives to improve the productivity, reduce cost and improve the efficiency business process. It is possible with the effective combination of Independent Systems, Data Exchange and Data Sharing between all processes of an enterprise. Enterprise Application Integration (EAI) is developed as a solution bridge the gap between business enterprises and IT sector to interact across different platforms.

Enterprise Application Integration (EAI) is the process of integrating the different applications that were developed by using different technologies on various platforms. There is a need for an architecture that provides for inclusion, interactivity, and context amongst all the stakeholders in the enterprise system.

One of the main challenges in the enterprise applications is interoperability [23]. There is a need to develop new methodology and framework for implementing the EAI. That implemented framework need to be the best fit for enterprise and it has to satisfy the requirement specification of the enterprise. New technologies like cloud computing and Internet of Things (IoT) are evolving in the market, so new integration methodologies are needed for such platforms.

The main aim of EAI in the enterprise is to ensure that all of its applications work together as though it is a single function. Model-driven development is a good methodology.

There are different criteria which must be considered when we integrate the applications. Reliability of target services, technology selections of different applications, extensibility, data format, data coupling and so on. For the question "Which integration approach best addresses which of these criteria?" there are different integration methods. Usually, no particular integration method which satisfies all the different criteria and constraints.

III. Types of EAI

There are different levels of EAI which depends on many factors including company size and budget, type of industry, integration and/or project complexity and so on.

The four well known levels of integration [17] are given below,

- Data Level
- Application Level
- Method Level
- User Interface Level

A. Data Level

It extracts data from one database to update another data base. Sometimes the extracted data may be transformed before updating the other database [3].

The best example of the data level is ETL (Extract, Transform, and Load) tool which can extract, transform and load data from different data sources and store in a common repository called as a data warehouse. Fig.1 represents the data level integration between the data sources.

Low cost and low risk are the main benefits of this level. Because, we are not modifying any code in the existing applications and no expenses to developing, testing and deploying new versions of the applications. The main drawback of this approach is to maintain the database design to understand the operations of the data generated by this approach.

Fig.1. Data-Level Integration

B. Application Level

This is the level of integration it consists of different interfaces which provided by the packaged applications to access the information between packages. Generally, this kind of integration is done as follows.

- Extracting the information from a source application through a provided application interface.
- Convert the data in the understandable format of the target application.

- Transfer the information to the targeted application.

"Message Broker" approach is the most common usage to implement this kind of integration [19]. In this approach, it controls the flow of information through hub and bus frameworks.

Fig.2 shows the application level integration. The interfacing between the different applications is relatively easy due to applications interfaces are provided by the applications. This is the main advantage of this level of integration. The disadvantage of this approach is the cost of the message broker.

Fig. 2. Application Level Integration

C. Method Level

Method-level integration is similar to application level interface but at a lower level of granularity. The main idea behind this level does not even share the functions but to share the different methods used in this function. Reusing approach is emphasized in this level of interface [29]. So, all other enterprise applications need to implement the same methods can use them without having to rewrite it. Fig.3 denotes the method level integration between different methods.

This integration level can be done with a lot of technologies like Java RMI, CORBA, and DCOM and so on. The emerging trend in implementing this approach is Web services [30]. The methods are shared easily by using web services.

The ability to reuse business logic and sharing the methods make this approach very suited for EAI. This method supposes the modification of existing applications to allow the sharing at such a low level.

Fig.3. Method Level Integration

D. User Interface Level

It is used for bundling applications into one by using their user interfaces as a common point of integration. It consists of replacing existing user interfaces of legacy systems and the Windows, Icons, Menus, and Pointer based interfaces of recent applications with standardized interfaces. User interface level integration is less expensive and less flexible than other approaches. The code of the existing applications is not modified [7]. The given below fig.4 represents the user interface level integration between two legacy systems.

Fig.4. User Interface Level Integration

In this paper, three more levels in enterprise application integration are proposed. They are,

- Architecture Level
- System Level
- Formal level

E. Architecture Level

Every component of the enterprise uses different data formats, various programming languages, and even different operating systems with a standard interface. Service Oriented Architecture (SOA) [6] is the best model to integrate the various components at the architecture level.

It integrates different services from different vendors, independent platforms, and different technologies [15]. The main advantages of this level integration are Loose Coupling, High scalability, and flexibility. It enables interoperability across heterogeneous systems.

F. System Level

System level integration is the process of combining different systems and software to behave like one physical and functional system. The system level integration is more important in the modern internet world. Many systems and applications are interconnected to different systems and sometimes it needs to connect already deployed systems. Fig.5 shows the system level integration.

ESB (Enterprise Service Bus) approach is the type of integration approach in which all subsystems will be communicated to other subsystems. The main benefits of this level integration are,

- More efficiency
- More profitability

Fig. 5. System Level Integration

All levels of EAI including the proposed levels are shown in the figure 6.

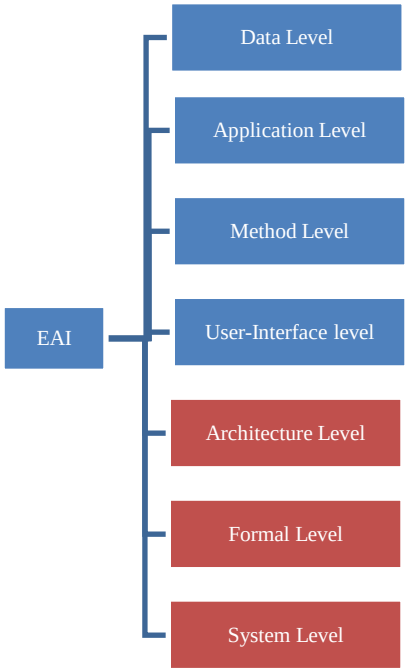

Fig. 6. Total level of EAI

Formal level of enterprise application integration will be discussed in the next section. There is a lot of research going on in the Enterprise Application Integration [EAI]. EAI is a very complex integration mechanism because of its distributed nature. Moreover, the development process and requirements of applications change rapidly. A new methodology and framework are needed to develop and achieve the meaningful EAI to integrate even the applications built for cloud computing and IoT.

Data security and privacy is another challenge in the EAI. Presently, there are not many comprehensive efforts in maintaining data security and privacy. EAI is not a simply integrating the applications built on different platforms. It is also to facilitate cooperation between different processes and business customers. The core challenges in EAI are as follows. EAI today demands the ability to integrate millions of sensors and computers, big data analytics to wireless networks. There is a need for theories and formal proofs to verify the design and development of EAI.

IV. Formal Methods

Formal methods can be present in all phases of a software project. The Project manager will decide when these methods should be used in different phases to detect more defects. Most of the time we can ensure that to get defect free software. Usually, software systems almost always contain some errors, even after rigorous testing in development and user acceptance. Some surveys saying that minimum 3 to 20 errors may occur from every thousand lines of code at the time of software putting into service and after finishing the normal testing and user acceptance testing[13]. When the developer has used formal methods, even in part of development, it becomes easy to identify the defects easily, quickly and thoroughly than the traditional methods.

Formal methods are categorized into two categories model–oriented and property–oriented approaches [31]. In a model-oriented approach, the system's behaviour is defined by creating a model of the system in terms of mathematical structures like tuples, functions, sets, sequences, relations and so on. But, in property-oriented approach, the behaviour of the system is indirectly defined by expressing its properties in the form of a set of axioms which satisfies the system. The Examples of popular model-oriented specification techniques are Z [28], CSP, and CCS and so on. Model-oriented

approaches will use in future software life cycle process also because changes of some specification will effect on the entire specification of the system. Property-oriented approaches are suitable for requirements specification because they can be easily changed. They specify a system as a conjunction of axioms and you can easily replace one axiom with another one.

V. Impact of the Formal Methods on Software Systems

Reliability is one of the major challenges in the software and sometimes it is very critical for the people. When a failure occurs, it leads to disasters where people die or large sums of money are lost. Such software is very complex and it is not easy to verify the correctness. It is often full of bugs and results in delay, over cost and many usability problems. The best and famous example is the Ariane 5 rocket explosion in 1996 that was due to a software bug (a data conversion of a too large number). To overcome such problems, it has been suggested to use formal methods in software, especially in the safety critical system [5, 14].

Formal methods help us to discover the errors in the early stage of the life cycle in the software development process. It reduces the overall cost of the project. Formal methods will provide the kind of evidence and give the solid reason for trust in the product. It is needed in heavy industries like avionics, telecommunications, and automobiles and so on where half of the projects fail. Formal methods for end-to-end flow control in EAI are necessary.

The benefits of formal methods include a decrease in the rework process, automatic verification of some properties and most importantly finding errors in early stages. Formal Methods can also be used in reverse engineering for the model and analyze existing systems. Formal methods do not give any guarantee to the correctness of the system but can be used to increase the level of correctness.

The primary reason for failures in software engineering is unstable requirements and specifications [11, 31]. The formal methods use mathematics to structure and analyze the requirements and specifications in a manner where changes can be systematic. Formal methods help the engineers to construct more reliable systems.

VI. Basics of Formal Methods

A. Formal Specification

A specification is said to be formal if it is expressed as a notation or formal specification language in which the notations are based on mathematical logic and set theory. The formal specifications are used in the analysis and design phases to record the requirements and design decisions in the development lifecycle [10].

It can be used to predict the behaviour of the product before implementation by using some formal analysis and test cases.

The formal specification is characterized as being abstract and a refinement of requirements analysis phase of the development lifecycle. A specification is created in the requirement analysis phase and it should describe the requirements of the software system to be implemented. The mathematical notation must prove the properties the formal specifications in order to verify and validate them.

Verification is only concerned with the correctness of a product with respect to its specification [18]. Another question is whether the specification correctly describes the problem to be solved. Validation is the act of investigating the latter. Specifications properties can be validated formally by using some mathematical statements. It can also be validated by testing when the specifications are executable form.

B. Formal Specification Languages

A specification language is a formal language which is used in the requirement analysis phase and design phase of the software design lifecycle to describe the system. Many specification languages exist that are suited for different kind of systems and different contexts. Each formal language has its own mathematical framework. It consists of notations of models and notation of statements that can be used to express the

properties of models that must be satisfied by the statements.

A specification can be a syntactic presentation of a model in which case it is said to be model-oriented specification language. This model is created using well understood mathematical entities such as functions and sets. Ex: B [26], VDM [22], Z [28] and so on.

A specification can be a syntactic presentation of a collection of statements or properties in which case it said to be property oriented specification language. This language is constructed using logical axioms, operations and their relationships and temporal logics. Example: CASL, Maude, and cafeOBJ and so on. For specification of concurrent systems, there is a famous language called CSP and CCS, it is constructed using process algebra.

C. Formal verification

Verification is the act of investigating whether the product is correct i.e. satisfies its specification. When mathematics is used for verification process it is called as formal verification. During the verification of the system design, a large number of theorems are proved. All these proofs are informal (not in mathematical form) and keeping track of the details of the proofs and interrelationship among the various theorems can be overwhelming. To ensure high safety, security and reliability proofs must be carried out with a high degree of precision. When the formal specification is used in the design and requirement phases, it is possible to use formal verification technique. Formal verification uses mathematical logic [33].

The advantage of formal verification is considerable in any context to find the defects before the system is implemented. The major approaches of formal verifications are theorem proving and model checking.

1) Theorem Proving

It is mathematical logic and proof for a statement to be true. The statement is known to be true and is said to be theorem when the act results in a proof. If a proof is not found, one cannot conclude that the statement is false.

There are many ways to prove that implementation satisfies the specifications. The proof can be either manual or automated. It is not suitable for large software and hardware systems. Proofs can also be interactively checked by the machine based on the steps provided by the human. They can give a guarantee of correctness if the theorem prover is correct. Automated theorem prover can solve the problem within a reasonable time. It is faster than interactive proving. Many problems cannot be solved automatically [10].

This approach needs large manpower to prove small theorems. It is difficult to prove large and hard theorems. It is usable only by the experts. It requires a deep understanding of both system design and methodology. There exist many programming tools which construct formal proofs [4] some of them are given below.

Based on the proof checker Meta Math and Mizar so on are the programming tools in that it can check automatically whether the suggested proof is actually a correct proof of the given theorem.

Based on the interactive theorem prover PVS, Isabella, HOL, ACL, and COQ so on are the tools that construct a correct proof by interactively. This is a most common form of tools.

Based on the automated theorem prover SPASS and Prover 9 so on are the programming tool that automatically searches for a proof of the given theorem. These tools are most difficult to create as it is and computationally hard to find a proof.

2) Model Checking

Model checking is the automated approach to verify that a model of a system satisfies a formal specification of requirements to the system. In this approach, the models describe how the state of the system may evolve over time. Tools that automatically perform model checking are called model checkers.

The process of the model checker is to create a model of the system as a first step and to formalize the requirement by obtaining a formal specification of the system. The model checker returns information about whether the model satisfied the property of the specification. Model

checking is a promising technique for the improvement of software quality. A model of a program consists of states and transitions, and a specification or property is a logical formula.

A model checker is a tool which automatically performs the model checking. Many of model checkers exist. Each model checker has a different specification language for expressing the model and expressing the properties [20] some notable examples of model checker are SPIN, NUSMV, and SAL.

Here we can compare the model checking and theorem proving, Model checking is fully automated and much easier and faster to use than the theorem proving. Model checking can't be used for generalization of generic parameter system.

VII. Impact of Formal Methods

A. Impact of the Formal Methods on Avionics

Formal Methods are being incorporated in the aircraft and spacecraft software design and verification process [27] Pete Manolios, USA, focuses on integrated modular avionics, verification cost, and system integration. Marc Pantel, FRANCE, focuses on developing embedded systems in avionics and safety requirements. Guillaume Brat, NASA, USA, focuses on sound, precise and scalable static analyses of flight control system [9].

There are two important methods for verifying the avionics formally. Deductive Method and Abstract interpretation based static analysis. The first deductive methods are based on a hoare logic concept with tools of Caveat and Frama-C. The second technique is based on abstract interpretation with automated tools like Astree, aiT, Stackanalyzer, and Fluctuat.

B. Impact of formal methods on Automobile

The automobile industry is rapidly changing from a mechanical industry to one driven by innovation in electronics and embedded software. Many new safety and convenience features are being designed in the presence of traffic and whether condition, driving skills level, road condition, formal methods are needed.

C. Impact of formal methods on Virus-malware

Researchers in academia and industries are beginning to develop anti-virus technologies founded on formal methods of analyzing programs. These methods with mathematical foundations have mostly been developed for optimizing compilers and more recently for hardware and software verification.

Some formal models [43] of virus transformation are developed and tested the against virus attacks. Proposed [44] some nonlinear mathematical models to study the role of antivirus program in the computer network and analysed. Present malware detection tools are operated by searching for pattern matches with respect to signatures of known malware. These detectors generally in capable of identifying newly released malware. We need a formal method to investigate the malware detection and behaviour.

D. Impact of Formal Methods on Telecommunication

Telecommunication services are defined as a service that is provided by the public switched telephone network. The main system requirements of telecommunication services are to communicate between user and devices and interaction between two communication systems with one another.

Some characteristics of telecommunication services are [8] concurrency, distribution, relativity, code size, complexity, large-scale environment, reliability, availability, and interoperability so on. The telecommunications industry has developed many standards for fulfilling the requirements and characteristics [2].

Mostly SDL (Specification Description Language), Z and PROMELA [42] are used in the telecommunication industry.

E. Impact of formal methods on Defense

The use of Formal Methods is mandatory for certain classes of military software. The major problems are the integration of different components and subsystems, such as radar, electronic support measures, navigation, communication and mission data processing.

Colored Petri Nets [40] [41] are one of the formal specification language and graphical oriented modelling language for the design, specification, and verification of distributed systems. One of the key technical challenges of defense is to develop high assurance safety critical cyber system.

F. Impact of formal methods on Robotics

An autonomous system is an artificially intelligent entity which is interacting and make a decision by means of inputs and the robotic system is a physical entity that interacts directly with the physical world. So we consider Autonomous Robotic System as a machine that uses Artificial Intelligence (AI) technology and physically interacts in and out with the real world. Autonomous Robotics is a very complex, hybrid system and combination of both hardware and software. Nowadays, Autonomous Robotics is being used tremendously in many fields like Driverless cars, Pilotless Aircrafts, Industries and acts as Assistants In many domestic and international organizations.

The common challenges of autonomous robotics systems are heavily dependent on software control, exact decision making, and deployment in critical scenarios, along with it requires strong verification methods to deal with all mentioned challenges. Here the concept of Formal Methods is needed which is a mathematical-based technique, for both verification and specification of systems, to ensure the correctness and evidence for the robotic systems.

Most of the formalisms used to verify or specify the robotic systems. We can divide the formalisms like set-based, automata, logics, process algebras, etc. Set-based formalism like the Z and B-method are representing a system using set theory logics and capturing the manipulation of data. [35] Describe a formal reference model of a self-adaptive system called FORMS by using Z specification language. [36] Proposed a Jaza animator by using Z specification language and java debugger to check the run time monitoring system. This is used as a model for a robot assembly system in the NASA ANTS project [45].

Petri Net and Finite state automata are the formalisms to specify the behaviors of the system. [40] In that they mentioned the Petri Net logic to capture the robot plans. The Petri Net models are used to analyze the deadlock and resource usage of robots.

Temporal logics formalisms are the most relevant formalisms for autonomous robotic systems to analyze and verify the systems. [37] Proposed some rules and assumptions by using Linear-time Temporal Logic (LTL) to verify the autonomous pilotless aircraft. [38] Represent the model by using the Probabilistic Temporal Logic (PTL) for safety rules and the environment of the autonomous robot assistants. These verification and specification results improve the confidence in the safety of the robot's high-level decision-making-

VIII. Case Study

In this section, we present a case study with two applications. First application had explained the details of the faculty name and their contact number. The second application includes the address of the faculty with some authentication procedure.

The objective of the first application (Michael Butler 2001) is to construct the telephone directory system in the university to maintain the faculty names and their contact numbers. The specifications or requirements are like,

– A faculty may have one or more telephone numbers

– Must be able to add new faculty and/or new entries

– Must be able to remove faculty and/or existing entries

– Must be able to query the system for a faculty or number

The system behaviour is implemented by Z [32] [28] schemas. The schemas are described as abstract specifications and it is also implemented as concrete designs with added details to provide sufficient confidence about

the specification before the coding. Our Z schema must satisfy the above mentioned requirements.

The below schema represents the creating the *PhoneDir* for storing the faculty name and contact number.

PhoneDir

$faculty : \mathbb{P}\ person$

$directory : person \leftrightarrow phone$

$\text{dom}\ directory \subseteq faculty$

Phonedir′

$Faculty' : \mathbb{P}\ person$

$Directory' : person \leftrightarrow phone$

$Dom\ directory' \subseteq faculty'$

ΔPhoneDir

$Faculty, faculty' : \mathbb{P}\ person$

$Directory, directory' : person \leftrightarrow phone$

$dom\ directory \subseteq faculty$

$dom\ directory \subseteq faculty'$

ΞPhoneDir

$\Delta PhoneDir$

$Faculry' = faculty$

$Directory' = directory$

The schema *Addmember* represents the adding of faculty name in the directory. It will check whether the faculty already existed in the directory or not. If it does not exist it will be added to the faculty data and it will modify the faculty and directory automatically. The next schema *Facultyexists* will execute if faculty already existed.

Addmember

$\Delta phonedir$

$name? : person$

$rep! : MESSAGE$

$name? \notin faculty$

$faculty' = faculty \cup \{name?\}$

$directory'' = directory$

$rep! = \text{“}Ok\text{”}$

Facultyexists

$\Delta phonedir$

$name? : person$

$rep! : MESSAGE$

$name? \in faculty$

$rep! = \text{“}Faculty\ already\ exists\text{”}$

The below schema *AddEntry* represents the adding the entry of the faculty name and phone number into the directory.

AddEntry

$\Delta phoneDir$
name?:*person*
number?:*phone*
rep!:*MESSAGE*

---

name?$\in$*faculty*
name?$\mapsto$*number*?$\notin$*directory*
directory′=*directory* $\cup$ {*name*?$\mapsto$*number*?}
faculty′=*faculty*
rep!= "*Ok*"

Removing the faculty and removing the total entry were represents in the below schema.

Removefaculty

$\Delta phoneDir$
Name?:*person*
rep!:*MESSAGE*

---

name?$\in$*faculty*
faculty′$\in$*faculty*\{*name*?}
directory′={*name*?}$\lhd$*directory*
rep!= "*Ok*"

RemoveEntry

$\Delta phoneDir$
Number?:*phone*
Name?:*person*
rep!:*MESSAGE*

---

name?$\mapsto$*number*?$\in$*directory*
directory′=*directory*\{*name*?$\mapsto$*number*?}
faculty′=*faculty*
rep!= "*Ok*"

We can retrieve the faculty details and finding the contact number of the particular faculty by using given below schema.

Findnumber

$\Xi phoneDir$
name?:*person*
number!:$\mathbb{P}$*phone*
rep!:*MESSAGE*

---

name?$\in$*faculty*
name?$\in$dom *directory*
number!=*directory*$(\!|${*name*}$|\!)$
rep!= "*Ok*"

Findname

$\Xi phoneDir$
Number?:*phone*
name!:$\mathbb{P}$ *person*
rep!:*MESSAGE*

---

number?$\in$ran *directory*
name!=*directory*′$(\!|${*number*}$|\!)$
rep!= "*Ok*"

The above schemas (Application 1) represents the storing of the faculty name and contact number of the particular faculty.

The second application saying that storing the address of the particular faculty with their contact number. Schema of second application is showing in the below schema.

Addrdirectory

dir: *PHONE*$\rightarrow$*ADDRESS*

Addressbook

Address:$\mathbb{P}$ *ADDRESS*
dir:*PHONE*$\rightarrow$*ADDRESS*

---

dom *directory*$\subseteq$*Address*

Addressbook′

Address′:ℙ *ADDRESS*
Addrdirectory′:*PHONE*→*ADDRESS*

dom *directory*⊆*Address′*

Δ*Addressbook*

Address,*Address′*:ℙ *ADDRESS*
Addrdirectory,*Adddirectory′*:*PHONE*→*ADDRESS*

dom *directory*⊆*Address*
dom *directory*⊆*Address′*

Ξ*Addressbook*

Δ*Addressbook*

Address′=*Address*
Addrdirectory′= *Addrdiectory*

The schema *Addressbook* is the main directory in which can store the details of address along with contact number. *Addaddress* represents the adding of address. The address will add automatically if it does not exist already, otherwise, we will get the message like address already exist. The schema of the above explanation represents like,

Addaddress

Δ*Addressbook*
Addr?:*ADDRESS*
rep!:*MESSAGE*

Addr?∉*Address*
Address′=*Address*∪{*Addr*?}
Addrdirectory′=*Addrdirectory*
rep!= " *Ok"*

Addressexist

Δ*Addressbook*
Addr?:*ADDRESS*
rep!:*MESSAGE*

Addr?∈*Address*
rep!= "*Address alerdy exist"*

The given schema represents that adding total entry into the directory *Addressbook*. Here also we will get the already existing message once it already exists, otherwise, it will add in to the directory.

Addaddressentry

Δ*Addrdirectory*
Addr?:*ADDRESS*
Number?:*PHONE*
rep!:*MESSAGE*

Addr?∈*Address*
number?→*addr*?∉*Addrdirectory*
Addrdirectory′=*Addrdirectory*∪{*number*?→*addr*?}
Address′=*Address*
rep!="Ok"

Entryexist

Δ*Addrdirectory*
Addr?:*ADDRESS*
Number?:*PHONE*
rep!:*MESSAGE*

number?→*addr*?∉*Addrdirectory*
rep!= " *Entry Alreay Exist"*

The schema *FindAddress* in which we can retrieving the address by using the contact number.

FindAddress

Ξ*Addrdirectory*
number?:*PHONE*
Addr!:*ADDRESS*
rep!:*MESSAGE*

---

number?$\in$dom *Addrdirectory*
Addr!=*Addrdirectory* (|*number*?|)
rep!="*Ok*"

The total schema of linking address to the phone number is,

Link address to number $\triangleq$ Addaddressentry $\cup$ Addressexist $\cup$ Entryexist

The 'Z' schema is very relevant and easily to understand. The static and dynamic characteristics of a system are described by Z schema. The states, relationship between states, some possible operations, and relationship those operations are can tell as static and dynamic characteristics of the system.

Consider, every application a separate development project. There are many reasons to introduce the new applications or projects with extended features i.e., maybe cost effective for development and maybe requirement specification. Now a days the pragmatic of software engineering become very expansive. It either has to chuck the old application or software and it includes some new features to improve the application.

Most of the time information security problem will happen at the end of execution of the application. In this case study, there are some issues while authenticating whether the given address is correct or not. The enhanced done in this application to improve the authentication that linking the Aadhar Card to each and every faculty member by using their address. So that the issue in this case study can easily recover.

The below Z schema represents the linking of Aadhar Card with faculty address. *Aadhardirectory* is the main directory of this application.

Aadhardirectory

dir: *Address*$\leftrightarrow$*Aadhar*
MESSAGE ::= "*Ok*" | "*Already Exist*"

The *Addaadharentry* schema is using for adding the Aadhar number and Address. *Findaddress* represents retrieving the address by giving input as an Aadhar number.

Addaadharentry

Δ*Aadhardirectory*
Anumber?:*AADHARNUMBER*
Addr?:*ADDRESS*
rep!:*MESSAGE*

Addr?$\in$*Address*
Aadhardirectory′=*Aadhardirectory* $\cup$
{ *Addr*?$\rightarrow$*Anumber*?}
rep!= "*Ok*"

FindAddress

Ξ*Aadhardirectory*
Anumber?:*AADHARNUMBER*
Addr!:*ADDRESS*
rep!:*MESSAGE*

---

number?$\in$dom *Aadhardirectory*
Addr!=*Aadhardirectory* (|*Anumber*?|)
rep!="*Ok*"

Like this, every software must authenticate the data which can enter into the application. Not only in terms of data, even must verify in terms of the product also. Whatever verification or

authentication is to be needed, must be done before itself or while giving the specification in the top level of the process. Finally, we can identify the authenticated and non-authenticated persons separately.

Integration plays a key role in this case study to get the complete product and can access easily.

Because of formal specification, it appears that it is a seamless integration here we are not calling the additional module and not built other version of the software but it works seamlessly. The advantage of having formal specification is second application is more or less similar to the first one with something more. In the field of software engineering, we faced a lot of troubles many times because of this type of problems. The formal specification will definitely improve the way we are dealing with evolving software specification. It appears easily connectable. If only all application developers are used this formal specification within less span of time we would say that they all similar. We don't have worry about what they are used, but by the formal specification, all application developers would think that all applications are same under a product. This is the way software specification evolves for every integrated application.

The application development method has become very dominant because we are working on shorter life cycles. In this case study, the address book data is integrated with the existing phone number of faculty. Data level integration is playing the key role in this work and consequently all levels of integrations like method level, applications level, architecture level, formal level and so on.

To integrate different applications, one has to consider all levels of SDLC [Software Development Life Cycle] and levels of EAI [Enterprise Application Integration] like data level, application level, architecture level and system level and so on.

Every legacy system has been facing serious constraints in working with different modern applications. The cost of a working archived / legacy application is much higher than the changing the application. It becomes to extend the functionalities of existing projects. Here the third party maintenance plays a key role in maintaining the projects. Even though the project is under maintenance, because of high cost and more drying the project, the entire project is scrapped. Enterprise Application Integration plays a dominant role in this situation once the application has retired. Buy one more application or thinking about second application if the first application had retired is a common solution. But some sunken work in the first application can be used in the second application.

This case study would deal with the importance of application integration and evolving software specification in which the applications are developed by formal specification language like Z. However, there are some limitations in Z formalism as an integrator in this case study.

IX. Proposed Work

The authors are working on several other case studies and how they map to the proposed levels as in Figure 6.

From the past five decades bringing the practice of software in terms of application development, integrating applications and formal methods together. The fitting into levels and mapping case studies to different levels is a challenge. There is a need for factoring the hardware architecture for supporting the formal verification of performance.

A grammar which will decide the level as in Figure 6 can result in automated verification and

proving. Context Free Grammar [CFG] [34] plays a key role in controlling functionalities of the system by using grammatical rules and it is preliminary for understanding the way formal proofs in computing are constructed. Eventually a generic formalism such as Z will be used for specifying the evolving complex systems. The integration of CFG rules and Z notation will increase the modelling power of the complex system [25]

A better understanding of the behaviour of the system is very important. Behaviour of the each object and system will change based on the requirements. Commonly, interaction diagrams are used for representing the behaviour of the system and model. Sequence Diagram and Collaboration Diagram are the two ways of representation of system behaviour [24].

It is difficult to get a complete ordering of events to match the hierarchy or levels. Hence getting the correct sequence of actions to formally prove is difficult [21] thus the Sequence Diagram becomes one of the drawbacks of Unified Modelling Language (UML) while verifying formally. In complex or large systems we can able to develop many sequence diagrams. Process Model is one of the alternate method for sequence diagram. That is the reason we are going to process view model. It deals with dynamics aspects of the system, explains the system processes and how they communicate with each other in the run time environment [12]. It addresses the concurrency, performance, scalability and so on. It is a group of activities in which all activities must work together towards a common goal.

Collaboration Model is the one which has a group of roles and connectors with some attributes [24]. It mainly defines the achieving of the goal not how could accomplish that. Actions will be done quickly and problem-solving will be faster and more effective. Reuse is one of the main usage of the collaboration model [16]. A design which is implemented by collaboration can apply in various situations also. Collaboration Model may need Service Level Agreements, Protocols, Non-Disclosure Agreements, Intellectual Property and such support systems.

Computational, Process and Collaboration (CPC) is a relatively new research area for integrating the different applications throughout the levels of enterprises and software systems formally.

Open System Interconnect [OSI] is a very successful approach in the layered architecture for Computer Networking [1]. The cost of ownership is very high in the integration. Early researchers developed request broker or resource broker models in which they connected some objects and components using Service request and Service Oriented Architectures. The emergence of IoT and Bring Your Own Device [BYOD] have been even more challenging in the implementation at the Operational levels.

X Conclusion

Enterprise Application Integration (EAI) mainly focuses on the design and implementation of the enterprise solutions. The demand for integration has motivated the rapid growth of technologies, usage of internet and development of applications with new platforms like Cloud Computing, Internet of Things [IOT] and Artificial Intelligence technologies and so on. Basics of the EAI and Formal methods including usage of formal methods in industry scenario are explained. Some research directions like Context Free Grammar, UML, Collaboration and Process Models on these topics are indicated. It is proposed that every application should satisfy all levels of EAI and must be satisfied phases of software life cycle while integrating the applications. One case study with Z specification

language representing the proposed method to integrate different applications formally is included in this paper.

References

[1] Andrew, S. Tanenbaum., David, J. Wetherall., Computer Networks, 5th ed., *Prentice Hall, Pearson Education*.,2013.

[2]Ardis, M.A., Formal methods for telecommunication system requirements: A survey of standardized languages, *Springer Annals of Software Engineering,* 3(1), 157–187, 1997.

[3] Bettino, E., Ferrari, E, XML and data integration, *IEEE Internet Computing*, 5(6), 75 – 76, 2001.

[4] Bjorner, D., Havelund, K.,40 years of formal methods: Some Obstacles and Some Possibilities, *Springer*, 8442, 42–61, 2014.

[5] Bowen, J., Stavridou, V., Safety-Critical Systems, Formal Methods and Standards, *Software Engineering Journal*, 8:189-209, 1993.

[6] Chen, X., Xu, H., One service-oriented architecture based enterprise application integration platform, *9th International Conference on Advanced Communication Technology*, Volume 1, 746-751, 2007.

[7] Daniel, F., Yu, J.,Benatallah, B., Understanding UI integration: A survey of problems, technologies, and opportunities*, IEEE Internet Computing,* 11,59-66,2007.

[8] Dietrich, F., Hubaux, J. P., Formal methods for communication services: meeting the industry expectations, *Journal of Computer Networks*. 38(1), 99-120., 2001.

[9] Feron, Eric., Formal methods for aerospace applications: *In Formal Methods in Computer-Aided Design (FMCAD),*IEEE. 3-3.

[10] Haxthausen, An Introduction to Formal Methods for the Development of Safety-critical Applications, *Technical University of Denmark*, Lyngby, Denmark, 2010.

[11] Hussain, S., Dunne, P., Rasool, G.:Formal Specification of Security Properties using Z Notation, *Research Journal of Applied Sciences, Engineering and Technology*, 5(19), 4664-4670,2013.

[12] Inoue,K., Ogihara, T., Kikuno, T., Torii, K.: A formal adaption method for process descriptions, *ACM 11th international conference on Software engineering*, 145-153,1989.

[13] Ioana Rodhe, Martin Karresand.: Overview of formal methods in software engineering, *FOI, Swedish Defence Research Agency*, 2015.

[14] Jonathan Lockhart, Carla Purdy, Philip Wilsey.: Formal Methods for Safety Critical System Specification, *IEEE 57th International Midwest Symposium,* 201-204, 2014.

[15] Jujian, Z.:Apparel enterprise application integration model based on service-oriented architecture, *ICAL'09*, *IEEE International Conference on Automation and Logistics*, 1374-1377,2009.

[16]Kusumasari, T.F., Supriana,I., Surendro,K., Sastramihardja,H.:Collaboration model of software development, *International Conference on Electrical Engineering and Informatics (ICEEI),* 1-6,2011.

[17] Laftsidis, A.: Enterprise Application Integration. *IBM Sweden*,2003

[18] Laurent, O.: Using formal methods and testability concepts in the avionics systems validation and verification (V&V) process*, Third International Conference Software Testing, Verification and Validation (ICST),* 1-10, 2010.

[19]Linthicum,D.S. Enterprise application integration. Addison-*Wesley Professional*, 2000.

[20] Michael, Butler. : Introductory Notes on Specification with Z, Department of Electronics and Computer Science. University of Southampton.2001.

[21] Minhas, N.M., Qazi, A.M., Shahzadi, S., Ghafoor, S.:An Integration of UML Sequence Diagram with Formal Specification Methods—A Formal Solution Based on Z, *Journal of Software Engineering and Applications*, 8(08), 372-383, 2015.

[22] Muller, Andreas.: VDM: The Vienna Development Method, *Research Institute for Symbolic Computation (RISC),* Johannes Kepler University Linz, Austria,2009.

[23] Payton, J., Gamble, R., Kimsen, S.:The opportunity for formal models of integration,*2nd International Conference on Information Reuse and Integration*, 2000.

[24] Rumbaugh, J., Jacobson, I., Booch, G.:Unified modelling language reference manual, *Pearson Higher Education*, 2004.

[25] Sabir, N., Ali, A. Zafar, N. A., Linking finite automata and formal methods enhancing modelling power for complex systems, *IEEE International Conference on Computer Science and Information Technology*. *ICCSIT*.58-63.2008.

[26] Schneider, S., The B-Method: An Introduction. *Cornerstones of Computing*, Palgrave, 2001.

[27] Souyris, J., Wiels, V., Delmas, D., Formal Verification of Avionics Software Products, *FM '09 Proceedings of the 2nd World Congress on Formal Methods*, Eindhoven, Netherlands.532–546, 2009.

[28] Spivey, J. M., The Z Notation, Reference Manual. Programming Research Group, *International Series in Computer Science Prentice Hall International* (UK) Ltd.1992.

[29] Staab,S., Benjamins, V. R.,sheth, A., Web services: been there, done that? , *IEEE Intelligent Systems,* 18(1), 72 – 85.2003.

[30] Vinoski, S., Integration with Web services, *IEEE Internet Computing*, 7(6), 75-77. 2003.

[31] Vojislav, B. M, Dusan, M. V., Formal specifications in Software development: an overview, *Yugoslav Journal of Operations Research*, 1, 79·96. 1997.

[32] Woodcock, J. C. P., Davies, J., Using Z: Specification Refinement, and Proof". *Prentice Hall International*. 1996.

[33] You, J., Li, J., Xia, S., A survey on formal methods using in software development, *International Conference on Information Science and Control Engineering,* IET, 1–4.2012.

[34] Zafar, N. A., Khan, S. A., Alhumaidan, F., Kamran, B., Formal Modelling towards the Context Free Grammar, *Life Science Journal*, 9(4).988-993.2012.

[35] D. Weyns, S. Malek, and J. Andersson. FORMS: a FOrmal Reference Model for Self-adaptation. In *Autonomous Computer*., page 205. ACM, 2010.

[36] H. Liang, J. S. Dong, J. Sun, and W. E. Wong. Software monitoring through formal specification animation. *Innovation System Software Eng*., 5(4):231, 2009.

[37] M. Webster, M. Fisher, N. Cameron, and M. Jump. Formal Methods for the Certification of Autonomous Unmanned Aircraft Systems. Volume 6894 of LNCS, pages 228–242. *Springer*, 2011.

[38] P. Gainer, C. Dixon, K. Dautenhahn, M. Fisher, U. Hustadt, J. Saunders, and M. Webster. CRutoN, Automatic Verification of a Robotic Assistant's Behaviours. In *Form. Methods Ind. Crit. Syst.,* volume 10471 of LNCS, pages 119–133. Springer, 2017.

[39] V. A. Ziparo, L. Iocchi, D. Nardi, P. F. Palamara, and H. Costelha. Petri Net Plans: A Formal Model for Representation and Execution of Multi-robot Plans. In *Autonomous Agents Multiagent System*., volume 23 of AAMAS, pages 79–86, 2008.

[40] Bowden,F.D., Petri Nets and their Application to Command and Control Systems, *Defence Science and Technology Organization Canberra* Australia. 1996.

[41] Jensen, Kurt. A brief introduction to coloured petri nets. In *International Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, pp. 203-208. Springer, Berlin, Heidelberg, 1997.

[42] Zave, Pamela. Formal description of telecommunication services in Promela and Z. *NATO ASI SERIES F COMPUTER AND SYSTEMS SCIENCES* 173, 395-420, 1999.

[43] J. Morales, P. Clarke, Y. Deng, and B. M. Golam Kibria, Testing and evaluating virus detectors for handheld devices, *Journal in Computer Virology,* vol. 2, pp. 135-147, 2006/11/01 2006.

[44] J. B. Shukla, G. Singh, P. Shukla, and A. Tripathi, Modelling and analysis of the effects of antivirus software on an infected computer network, *Applied Mathematics and Computation*, vol. 227, pp. 11-18, 1/15/ 2014.

[45] https://attic.gsfc.nasa.gov/ants/