Mustafa Kamal

UI Construction for a Web-Based IDE on an Industrial IoT System

School of Science

Thesis submitted for examination for the degree of Master of Science in Technology.

Espoo 24.7.2017

Thesis supervisor:

Prof. Petri Vuorimaa

Thesis advisor:

Mika Luotojärvi M.Sc.

Aalto University
School of Science

AALTO UNIVERSITY
SCHOOL OF SCIENCE

ABSTRACT OF THE
MASTER'S THESIS

Author: Mustafa Kamal

Title: UI Construction for a Web-Based IDE on an Industrial IoT System

Date: 24.7.2017 Language: English Number of pages: 7+85

Department of Computer Science

Professorship: Digital Media Technology Code: T-111

Supervisor: Prof. Petri Vuorimaa

Advisor: Mika Luotojärvi M.Sc.

ABB as one of the leading power and automation company is connecting millions of electrical devices and systems to industrial internet of things called ABB Ability^{TM}. ABB Ability^{TM} is refining the measured real-time data with calculations to additional soft sensors signals and Key Performance Indicators (KPIs) at the various levels from the system edges to central cloud. The engineering of the calculations requires web based Integrated Development Environment (IDE) that provides good developer experience for the subject matter experts to be productive in their work. This thesis aims to construct a user interface for ABB calculation engine that will help subject matter expert to work on the calculation engine more efficiently. As the output of this thesis work, a web-based IDE is developed on top of ABB's internal front end dashboard framework. The developed user interface lets user to operate the calculation engine more efficiently and follows their natural work flow.

Keywords: User Interface, User Experience, Javascript, Integrated Development Environment, Internet of Things, Parser, Single Page Application

Acknowledgments

It is impossible for me to finish this thesis alone. It takes a lot of supports and contributions from many people to complete the whole work. I would like to use this opportunity to articulate my gratitude to all people who have helped me throughout this thesis work.

First, I would like to thank my thesis supervisor at Aalto University, Professor Petri Vuorimaa, who has given me so much guidances and directions about how to proceed with the whole master thesis process.

Many thanks for my thesis advisor at ABB, Mika Luotojarvi, for giving me a chance to do my master thesis at ABB with a very challenging and interesting topic. I also want to thank all my other colleagues at ABB for all the help during the thesis work period.

Big appreciation also goes to my scholarship sponsor, LPDP. Without its support on both funding and administration, I would not be able to pursue this further degree education.

I also want to thank my family for all the endless supports during my master study abroad. Lastly, I want to express my gratitude to Allah the almighty god for all the blessing He has been giving me in my life.

Helsinki, 08.18.2017

Mustafa Kamal

Abbreviations

AST	Abstract Syntax Tree
CRUD	Create, Read, Update, Delete
CAGR	Compound Annual Growth Rate
CSS	Cascading Style Sheet
DAI	Data Abstraction Interface
DFA	Deterministic Finite Automata
DOM	Document Object Model
FSA	Finite State Automata
FSM	Finite State Machine
GUI	Graphical User Interface
HCI	Human-Computer Interaction
HTML	Hyper Text Markup Language
IDE	Integrated Development Environment
IIoT	Industrial Internet of Things
IoT	Internet of Things
IoT-GSI	Global Standards Initiative on Internet of Things
ITU	International Telecommunication Union
ITU-T	ITU Telecommunication Standardization Sector
JSON	JavaScript Object Notation
KPI	Key Performance Index
NFA	Non-deterministic Finite Automata
OPC	Open Platform Communications
OPC DA	OPC Data Access
OPC HDA	OPC Historical Data Access
PC	Personal Computer
REST	REpresentational State Transfer
SASS	Syntactically Awesome Style Sheets
SDK	Software Development Kit
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
UI	User Interface
UX	User Experience
W3C	World Wide Web Consortium
WSDL	Web Service Definition Language
XHR	XmlHttpRequest

Contents

Chapter 1

Introduction

1.1 Background and Motivations

Internet of Things (IoT) technology has been gaining attention of experts all over the world. IoT enables anything to be connected to the internet. This connection will make it possible for any objects to communicate with one another. There is an ever increasing trend to incorporate IoT technology in various field.

The increasing trend of IoT implementation is also happening in the industrial manufacturing. According to a research report published by World Economy Forum, industrial internet is indeed transformative [28]. It is more and more important for any industrial devices to be connected to the internet. This connection will make it much easier and more efficient to process the raw data, which is retrieved from all those industrial devices.

ABB as one of the leading power and automation company is connecting millions of electrical devices and systems to industrial internet of things platform called ABB Ability$^{\mathrm{TM}}$. ABB Ability$^{\mathrm{TM}}$ is refining the measured real-time data with calculations to additional soft sensors signals and Key Performance Indicators (KPIs) at the various levels from the system edges to central cloud. This data refinement process is done by the subset system of ABB Ability called calculation engine. All configurations of the calculation engine are stored on the proprietary real time database engine made internally by ABB. The calculation engine would then read any configurations that has been stored in the database and execute the calculation based on those configurations. Results of the calculation can be stored back to the database for end users consumption.

1.2 Problem Statement

Prior to the newly developed calculation engine, ABB already has a calculation engine Software Development Kit (SDK). With this SDK, the calculations in ABB Ability platform are developed with Microsoft Visual Studio and a calculation framework that provides execution environment for the calculations. This approach has a drawback. It requires direct access (remote desktop) for the user to the back-end server and to its resources. This is not an acceptable arrangement especially from the cyber security perspective.

The new calculation engine was built to solve the security problem. The new approach provides client-side development of the calculations and contains two independent parts: a calculation engine (back-end) and client-side editor (front-end), which should have some level of IDE-like functionality. It allows the possibility of creating multiple client-side editors with the same server-side back-end. ABB already has two client-side editors. But these editors are not really IDEs and still have several limitations.

Without reliable client-side IDE, the only way to work with the new calculation engine is to directly fill the database tables of this calculation engine with the preferred configurations. This is a tedious work, especially when it comes to writing the calculation script. End users of this calculation engine within ABB have so far just used Microsoft Visual Studio to write their calculation code. After that, they would copy and paste the code directly to the database. This work flow is inefficient and slow. Especially, when they need to debug their code. This hinders the efficient utilization of the engine.

1.3 Objectives

The objectives of this thesis is to design a better user interface for the calculation engine. Another objective is also to provide a better experience for the end users of this calculation engine. With better UI/UX, it will be easier for the users to operate the engine. This will results in a more efficient utilization of the calculation engine. More efficient usage of the calculation engine also means faster and more productive work on analyzing all the raw data from the existing IoT system.

The user interface itself is expected to be in the form of a web based application. This web based application need to be built on top of the existing Javascript-based ABB's front end system. It will need to have some level of IDE-like functionality.

1.3.1 Research Questions

With that objectives in mind, here is the list of the research question for this thesis:

- **What are the requirements of the web based IDE for ABB Ability Calculation Engine?**
  This first question deals with the requirement of such system. In depth requirement analysis will need to be done as the foundation of the development of the web based IDE later on.

- **How should the web based IDE be designed and developed?**
  This second question deals with the development of the web based IDE. It investigates the choice of technology, the structure of the code, the layout for the UI and other technical consideration regarding the development.

1.4 Research Methodology

The research methodology for this thesis consists of two approaches: literature study and constructive research. Literature study is used for the theoretical background and constructive research is used to answer the main research questions.

1.4.1 Literature Review

This thesis explores the theoretical background by conducting literature study on various related topics. The study follows the guideline of systematic literature review in software engineering, which was written by Kitchenham in 2007 [50]. The goal of this literature study is to get a better and holistic understanding of the current state of researches in the field of industrial internet of things, user interface design and web application development in general. With this literature study, this thesis identifies, analyzes and synthesizes available relevant researches to the topic related to it [51]. The topics, which were investigated, include:

- Internet of things.
- Industrial internet of things.
- User interface design and study of human-computer interaction.
- Web application development.
- Parser and automata theory.

1.4.2 Constructive Research

To answer the main research question, this thesis uses the constructive research methodology. Constructive research is a research approach where a solution to the problem statement is constructed. It implies developing an artifact that solves a problem with the goal of formulating new knowledge about how to solve, understand, explain or model the problem [17]. The output of this methodology is called a construct. The construct could be a theory, algorithm, model, software or framework. The construct would then be evaluated by some benchmark test on the construct which has been built. In the case of this thesis, the construct is the web based IDE application, which will be used to configure a calculation engine on an industrial IoT system.

The constructive research approach consists of six phases [48][58]:

1. Choose a practical relevant problem to be solved with research potential.
2. Acquire a broader and more holistic understanding of the topic.
3. Build an artifact as a solution construct to the problem.
4. Demonstrate that the solution works.

5. Formulate the theoretical body of knowledge based on the presented solution.
6. Evaluate the pertinence of the solution.

This thesis incorporates all the steps. The practical problem in this case is how to present a better user interface to operate a calculation engine for industrial IoT system. The second step is done by doing the systematic literature review as well as assimilating ABB's industrial IoT system architecture. This includes gathering information about the relevant researches and understanding the existing ABB's code base. The third step is the one which takes most time. The artifact is built in the form of an actual front-end system, which can be used by the end users at ABB. The development of the artifact is also demonstrated, evaluated and reiterated. The last step is examined in the discussion chapter.

1.5 Thesis Structure

This thesis is structured as follows:

- The first chapter (1) contains the introduction of the thesis. The introduction consists of background and motivation of thesis along with the objective of the thesis. Next, the research questions are defined. It is then followed by explanation about the research methodologies used to answer the research questions.
- The theoretical background is split into four chapters. First theoretical chapter (2) examines the current state of research on industrial internet of things. The second part of the theoretical background is written in chapter 3, which discusses about user interface design. The third theoretical chapter (4) is about web application technology, which is being used as the foundation of building the web based IDE application. The last theoretical chapter (5) will examine the concept of parser and automata theory which is a fundamental concept for an important part of this thesis.
- The next chapter (6) is called ABB Ability. It contains explanation about the existing ABB Ability platform to which this thesis work is built upon. This way, there would be a clearer context for the remaining chapters.
- Chapter 7 is about requirements. It contains in-depth requirement analysis of the project, which is the object of the thesis work.
- After the requirements are clarified, comes the biggest part of the thesis. That is the implementation and development of the artifact in this constructive thesis work. The whole process is written in chapter 8
- Chapter 9 is discussion. This chapter contains evaluation of the thesis work and the possible future work based on this thesis work.
- The last chapter (10) is the conclusion.

Chapter 2

Industrial Internet of Things

This chapter discusses internet of things in general and also specifically industrial internet of things. In this chapter, this thesis will examine the characteristic and general potential of internet of things. This thesis will also look for latest researches in the field of internet of things. Chapter 2.1 will discuss about internet of things in general and chapter 2.2 will discuss specifically industrial internet of things.

2.1 Internet of Things

Definition of IoT

Internet of Things (IoT) is a term to define electronic interconnection between objects. The term was first coined by Kevin Ashton of Procter & Gamble in 1999 [7]. The definition has been standardized in 2013 by Global Standards Initiative on Internet of Things (IoT-GSI). It is defined in ITU-T Recommendations Y.4000/Y.2060 as "A global infrastructure for the information society, enabling advanced services by interconnecting (physical and virtual) things based on existing and evolving interoperable information and communication technologies" [67].

By that definition, the "things" in the context of IoT, which is often referred as *smart objects*, could be anything. It could be physical or virtual objects as long as these objects are augmented by network capabilities [53]. The physical things could be in the form of small gadgets like key chain, wristwatches, eyeglasses and various electrical equipments. It could also be in the form of big items like refrigerator, cars, industrial robots and buildings. Examples of virtual things are software and multimedia content. They could also be considered as part of the IoT network as long as they can be stored, processed and accessed [67].

The electronic interconnection between all the smart objects will enable them to interact with each other. This interconnection is much more complex than just a machine-to-machine interconnection. In fact, IoT has been seen as the extension of machine-to-machine communication [6][42]. In the IoT context, the interconnection is not limited only between two machines. It is instead a complex interconnection between multiple machines, which are connected to the global internet infrastructure.

One important thing to note is that IoT is not a single monolithic technology [52]. It is instead an umbrella term for collection of technologies, which work together to form the entity of what we call as IoT. The advance of those underlying technologies is what drives IoT to gain momentum in the recent years. Silicon chips that keep shrinking by size over time along with the decrease on price allow for creation of small sized embedded systems with more computing power to run basic calculations. Wireless networking technology that keeps finding a way to become faster and faster has made it possible for all the small device to be connected with ease [14].

Benefits of IoT

The complex inter-networking of things will make it possible for all those objects to collect and exchange their internal data. This exchange of data could then be used as a way to sense or control the object remotely through the network. As a result, data collection could be enhanced. It could be done much more frequently, it could even be done in real-time [52]. It will also improve access and control of devices. A good example in this case is home automation [35]. Smart home has been proven to give high level of ease and convenience for its resident to access and control the devices in and on the home.

Impacts of IoT

IoT, especially through the use of big data analytics, definitely has a big impact on our life. Riggin and Wamba (2015) wrote a paper which discusses about the creation of research framework for the topic of IoT impact. They split the impact on four level: society, industry, organizational and individual [71].

At the individual level, IoT has huge influence on our daily life. It helps us by automating many day-to-day tasks such as reordering groceries, tracking body weights and house monitoring [81][35]. All of those automated tasks increase individual productivity.

Looking from organizational perspective, IoT has been proven to have a big impact on increasing business efficiency. Improvements on the data collection, data handling, access and control of the smart objects are the factors behind it. Varian Medical System has seen 50 percent reduction in mean time to repair their connected devices because of the implementation of IoT. As a result, they successfully reduced customer service costs with 20 percent fewer technician dispatches worldwide [79]. Cisco with their IoT solution has helped its customer, Stanley Black & Decker, Inc., to increase Overall Equipment Effectiveness (OEE) on their router production line by 24 percent among many other improvements [15]. ABB's connected robot saves time and money for Injection Technology Corp. It reduces man-hours by 45 percent and also reduces cycle time by 23 percent [37].

Industry-wise, adoption of IoT could also potentially transforms various industries. Bandyopadhyay and Sen (2011) discuss in their article how IoT could impact automotive industry, telecommunications industry, health care industry and many others [8]. In general, the impact that IoT has on industry is mostly derived from its impact on the organizational level. IoT increases business efficiency, which in turn drive fundamental change in the industry as well.

In the broader societal sense, IoT also has various impacts. The advent of smart city could be seen as something that is fueled by the growth of IoT adoption [47]. Jeremy Rifkin in his book, The Zero Marginal Cost Society, goes even further by inspecting how IoT will affect human civilization. He describes in detail how IoT is speeding humanity to an era of nearly free goods and services. This precipitate the rise of global Collaborative Commons as well as diminishing the impact of capitalism over time [70].

Trends of IoT

In general, IoT has been seen as an emerging technology that will be disruptive for business, industry and society at large. Gartner, in its 2016 Hype Cycle report, has listed IoT as one of the emerging technologies in which organization must track to gain competitive advantage [33]. US National Intelligence Council, in its report, has listed IoT as one of six technologies that will be disruptive to our future [16]. The Google web search popularity for "Internet of Things" keywords also shows continuous sign of growth.

Many marketing consulting and technology companies have made forecast on the adoption of IoT. In 2016, IHS made a forecast that the installed base of IoT devices will grow from 15.4 billion devices in 2015 to 30.7 billion devices in 2020 and 75.4 billion in 2025 [57]. In 2015, McKinsey made an estimation that the IoT has a total potential economic impact of \$3.9 - \$11.1 trillion per year in 2025, which would amount to 11 percent of the world's economy in 2025 [63]. Bain & Company Inc, in its report, expects that annual revenues for IoT vendors could exceed \$450 billion by 2020 [11]. Gartner forecasts that 8.4 billion connected "Things" will be in use in 2017, increasing 31 Percent from 2016 [32]. IDC made a prediction that the global IoT revenue will reach \$7.065 billion by 2020 from \$1.927 billion in 2013, gaining a Compound Annual Growth Rate (CAGR) of 20.04% [59]. Ericsson forecasts that by 2022, there will be a total of around 29 billion connected devices worldwide, in which 17.6 billion of it is in the form of wide area IoT and short range IoT [12].

It is hard to make an apple-to-apple comparison between all the forecasts of IoT. Those forecasts seem to be vary widely. It is because each of the institution has their own definition of IoT and also their own unique method of forecasting. But despite the fact that the forecast of either the adoption rate and market growth of IoT is so divergent, it still shows that IoT is growing nonetheless.

2.2 Industrial Internet of Things

Industrial Internet of Things (IIoT) is the application of the IoT technology to the manufacturing industry. Industrial internet has been defined by Industrial Internet Consortium as an internet of things, machines, computers and people, enabling intelligent industrial operations using advanced data analytics for transformational business outcomes [56].

IIoT is invariably included as one of the most important component of Industrie 4.0 trend. The term Industrie 4.0 was first introduced by the German government in the strategic planning for their manufacturing industries [10]. The term was later popularized by representatives of German industry leaders, researchers, industry associations and unions [44][69]. The German term has now been spread in a more global sense into Industry 4.0. Industry 4.0 is the next industrial revolution, which is happening right now [38]. The first industrial revolution is mechanization of the manual labors. Second industrial revolution is electrification especially in the form of development of assembly line. Third industrial revolution is digitization with the introduction of computer technology. The fourth industrial revolution is the interconnection of all the digital industrial device to form a wide network of industrial things [21].

Benefits and Potentials of IIoT

IIoT offers many benefits for businesses from various industries. It improves connectivity, increases efficiency and pushes further scalability. All those improvements can lead to savings in time, resources and costs for the business. Further, higher efficiency in the industry will indirectly impact the economy at large as well.

The most common benefits that has been tapped by the industry from IIoT technology is in the case of predictive maintenance. Predictive maintenance is defined by Joseph Patton in his book, Maintainability and Maintenance Management, as measurements that detect the onset of system degradation (lower functional state), thereby allowing causal stressors to be eliminated or controlled prior to any significant deterioration in the component physical state [68]. IIoT technology provides a more versatile way to conduct predictive maintenance. With IIoT technology, system degradation can be detected even before it happens. Utilization of big data analytics can determine if a degradation is likely to happen based on all the raw data from the sensors. Predictive maintenance has been known to make a saving of up to 12 percent over scheduled report, reduce overall maintenance cost by 30 percent and eliminating breakdown by 70 percent [76].

Another obvious way IIoT could help business is by bridging a data gap from the factory floors up to the executive offices. With IIoT technology, raw data from sensors and devices can be processed in real-time. The processed data could then be analyzed and presented to the executive in a decision support system. This way, IIoT could aid the business decision making process to be faster and more rigorous [49].

IIoT also has the potential to open up a new revenue channel by allowing the development of innovative business models. Accenture in its white paper, has reported several cases of business which has been implementing a novel business model based on IIoT technology [18]. Michelin helps truck fleet managers reduce fuel costs and allows them to pay for tires on a kilometer-driven basis. CLAAS, an agricultural machinery company, moved into a service based business model by letting farmers to operate their equipment on autopilot. They also give advice to farmers on how to improve their crop productivity.

Challenges of IIoT

With all the benefits and potentials offered by IIoT, it also comes with its own set of challenges. Survey conducted by World Economic Forum shows that two of the biggest hurdles for IIoT are interoperability and security [28].

There are various IIoT device out there manufactured by different companies. Without a standardized protocol, those devices would not be able to talk to one another, subsiding the whole point of IIoT. To solve this problem, Industrial Internet Consortium has published the Industrial Internet Reference Architecture Technical Report. This resource, represents broad industry consensus, built to drive product interoperability and simplify development of Industrial Internet systems that are better built and integrated with shorter time to market and, at the end, able to better fulfill their intended uses [43].

Security is also an obvious issue. Advent of IIoT means that more data would be connected to the wider network, either a private network within the organization or the public internet itself. This puts the data at risk. For this issue, Industrial Internet Consortium has published The Industrial Internet Security Framework (IISF). This collaborative project is the most in-depth cross-industry-focused security framework comprising expert vision, experience and security best practices [73].

Current State of IIoT

Many innovative manufacturing companies have already implement some level of IIoT technology in their whole business process [61]. Cisco announced four global IoT Centres of Excellence at CES 2014. Shell has been implementing an IIoT solution for their oil field which they call The Smart Field. General Electric has offered a smart grid solution called GridIQ which depends heavily on IIoT technology. ABB has long offered IIoT solutions for its various clients. Now, they are moving into a unified IIoT solution called ABB Ability$^{\text{TM}}$, which is an integral part of this thesis [60].

On the public sectors, many governments have already started initiative to ignite the growth of industrial IoT. Germany inspires many other countries with its industrie 4.0 initiative [10]. USA, under president Barack Obama, has launched Advanced Manufacturing Partnership program. The program set smart manufacturing as one of the areas of interest for potential future investment [3]. Many Asian countries have also started their initiatives. Singapore's Smart Nation program includes support for IIoT initiatives in the country [75]. Taiwan initiates Productivity 4.0 as the answer to industry 4.0 movement [80]. Chinese government has a program called Made in China 2025 with the goal to comprehensively upgrade Chinese industry. One of the strategy is to incorporate IIoT technology into Chinese manufacturing industry [62].

Industrial internet growth momentum could also be seen from the establishment of Industrial Internet Consortium (IIC). This consortium was founded in March 2014 to bring together the organizations and technologies necessary to accelerate the growth of the Industrial Internet by identifying, assembling and promoting best practices. Their member includes small and large technology innovators, vertical market leaders, researchers, universities and government organizations [2].

Chapter 3

User Interface Design

The main task of this thesis work is to build a UI for ABB's calculation engine. For this purpose, a thorough examination of theoretical framework for UI design process needs to be done. The first sub chapter (3.1) will define what is UI to give a clearer scope of what we are discussing in this chapter. Next sub chapter (3.2) will examine what constitute a good UI and what factors contribute to how well a UI is and how do we measure it. After that, sub chapter 3.3 will dig into the topic of systematic UI design and development process from the literature. Lastly, chapter 3.4 investigates existing researched approaches to evaluate a UI.

3.1 User Interface Definition

Soren Lauesen in his book, User interface design: a software engineering perspective, defines User Interface (UI) as the part of the computer system that we can see, hear and feel [54]. Wilbert O. Galitz expands this definition. Beside being the part of the computer system that we can see, hear and feel, UI is also the part where we can talk to, or otherwise understand or direct [30].

UI has two components: input and output. Input component is where the user gives any command to the computer. In most computer systems, it is done via keyboard and mouse. Output component is where the computer shows its computation result to the user. In the traditional computer system, it is mostly through the monitor screen and/or speaker.

UI design is one of the most important topic in the field of Human-Computer Interaction (HCI). HCI has been defined by Association for Computing Machinery (ACM) as "a discipline concerned with the design, evaluation and implementation of interactive computing systems for human use and with the study of major phenomena surrounding them" [39]. The scope of HCI research is not only limited to the common usage of standard personal computer by an individual. The context of human in HCI does not only point to a single individual, it reaches to other scopes such as a group of people or sequence of users. The context of computer in HCI also does not limited to a single PC workstation. Computers in the modern world are everywhere and have many forms. Computers can be in the form of big machineries. There are computers in many modern vehicles. A smartphone can also be considered a computer. A small embedded system is also another form of computer. Lastly, the context of interaction in HCI has a wide meaning of any kinds of interaction between a user and computer. HCI is a multi-disciplinary field which combines the study of computer science, engineering, psychology, graphic design and many others.

3.2 User Interface Quality Factor

There are many factors that define a good UI. Many parties have tried to formulate some set of principles on designing high quality UI. Wilbert O. Galitz in his book, The Essential Guide to User Interface Design: An Introduction to GUI Design Principles and Techniques, has derived list of general principles that he took from various UI design guides of companies and organizations such as Microsoft, IBM and Open Software Foundation. The list is as follows [30]:

- **Aesthetically pleasing**. A UI should be aesthetically pleasing by aligning a good composition of layout, colors and graphical element. It will help the system to be more inviting for the user.
- **Clarity**. What is included in the scope of clarity of a UI here is the visual elements, functions, metaphors and wording. They all need to be clear, understandable, simple and unambiguous.
- **Compatibility**. A good UI should provide compatibility to the user, the task and the product. It needs to appropriate to the needs of the users. The organization of the system should follow the flow of task at hand. It should have some level of compatibility to an existing similar system that the user might already use.
- **Comprehensibility**. A good UI should be easily learned and understood.
- **Configurability**. One sign of an exceptional UI is its ability to be easily customized and personalized to ignite higher sense of belonging from the user.
- **Consistency**. A system should maintain consistency across its all views. Similar components should have a similar look and operate similarly. The function of an element should not change and the result of its action should always yield the same result.
- **Control**. The user must have control of the interaction with the UI. Any result should come from explicit user request and it should be provided as quickly as possible.
- **Directness**. A system should provide direct ways to accomplish task.
- **Efficiency**. A good UI should minimize the tasks needed by the users to operate it.

- **Familiarity**. Similar to the compatibility point on the user, a UI should mimic the user's behavior pattern and utilize familiar concept.
- **Flexibility**. A system should have some level of flexibility to accommodate different types of users.
- **Forgiveness**. A system should be able to tolerate the potential error that could be made by the users.
- **Predictability**. Users will naturally predict what would happen throughout his operating of the system. The UI should take into account this user's prediction.
- **Recovery**. A system should allow the user to go back to some point on the state of the UI. It should make sure that they would never lose their work for any reason.
- **Responsiveness**. A good UI is those which could provide immediate response to actions that are taken by the users.
- **Simplicity**. A simple and straightforward UI is one sign of a good UI.
- **Transparency**. The inner working of an interface should be transparent to the users, letting them to focus on their task or job.
- **Trade-Offs**. Many of those principles above have a high chance of conflicting with each other. A good UI should manage a good balance of trade-offs between all those principles.

Speaking about UI quality, one can not avoid the topic of usability. It could be concluded that a good UI equal to high usability. Vice versa, a bad UI indicates low usability. There is an ISO standard, which gives a definition for usability. In that standard, usability is defined as "The extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency, and satisfaction in a specified context of use" [45]. According to Soren Lauesen, usability consists of six factors [54]:

- **Fit for use**. This is functionality factor. How good is the system in terms of providing good means of its function.
- **Ease of learning**. How easy it is to learn the system become another factor to usability.
- **Task efficiency**. If it takes less time and less work for a user to achieve a task, it means that the system has high task efficiency.
- **Ease of remembering**. A system, which is easy to remember, means it is easier to operate.

- **Subjective satisfaction**. User personal satisfaction also contributes to the whole point of system usability.
- **Understandability**. A system with high usability means that the system is easy to understand.

All those principles apply for UI in general. Be it UI for desktop application, mobile application, web application or any other kinds of platform. There are of course some specific principle for each type of UI. In the case of web based application, especially those, which is considered rich internet application, there is also some set of principle on designing for this type of UI. Bill Scott in his book, Designing web Interfaces: Principles and Patterns for Rich Interactions, formulates six principles in this matter [74]:

- **Make it direct**. What it means is that the system should be able to respond directly to the user's interaction. Examples of design pattern on this principle includes in-page editing, drag-and-drop and direct selection.
- **Keep it lightweight**. A good rich internet application should avoid the problem that usually exists in desktop based UI: tool clutter. One example of design pattern on this term is contextual tools. What it means by contextual tools is a set of tools, which only appear on some context. Either only when it is hovered, or being put directly on a particular area on the page near to a related content, or something similar to right click menus on desktop application.
- **Stay on the page**. There was a time when web applications relies on the need to refresh and/or open a new page as a result of a particular action. It was cumbersome and broke user flow. Now, with modern web technologies like Ajax, we can avoid that problem. It has now became a trend among web application developers to build a single page application, which does not need any page refresh at all. Modern web application should go with this principle to minimize page refresh.
- **Provide an invitation**. Invitation in this context is the prompts and cues that help guiding users through an interaction.
- **Use transition**. Transition in a UI catches the user attention. A good UI should utilize it without overdoing it.
- **React immediately**. A good UI should provide immediate feedback to the user upon any interaction that they have started.

3.3 User Interface Design Process

There are many attempts to conceptualize a systematic approach to UI design process. Each attempt has their own pros and cons. Many of those attempts have similarities on parts of their process.

One classical approach is called iterative design. Iterative design is a methodology where the process of prototyping, testing, analyzing and refining is done in a cyclic manner. In this approach, the usage of the ready-made UI is used as a research object towards evolving the UI itself into its better form. This approach will allow developers to identify any usability issues that could exist on the UI. There are three core steps on this methodology [65]:

1. **Task analysis**. The first step is to do user task analysis. This needs to be done as the basis for developing the prototype later on.
2. **Prototype development**. The next step is to immediately build the prototype based on the result of the task analysis.
3. **Usability testing**. After the prototype is ready, a quick usability testing is then conducted to see whether the UI has good usability or not. The result of this process will be used to refine the prototype.

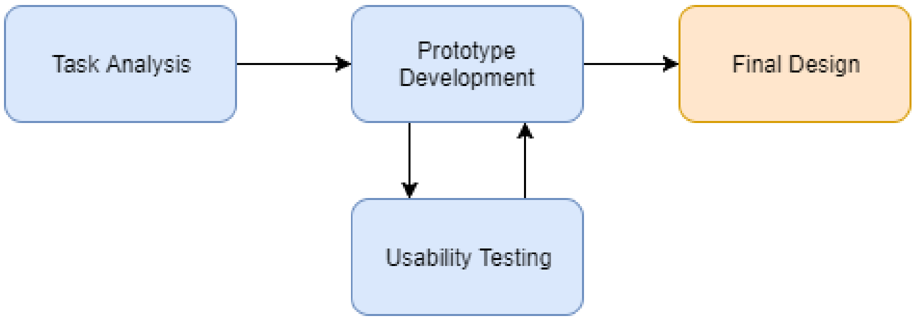


Figure 3.1: Iterative design diagram

Wilbert O. Galitz in his book, The Essential Guide to User Interface Design: An Introduction to GUI Design Principles and Techniques, formulates a more detailed process on designing UI. The process comprises of 14 steps. Those 14 steps are as follows: [30]

1. **Step 1 Know your user or client.**
   In the first step, we need to identify our users. Who are they? What are their level of skills, knowledges and experiences? What is the characteristics of their needs? All these questions need to be observed from the users.
2. **Step 2 Understand the business function.**
   After a thorough understanding of the user, the next step is to determine the business function. For this purpose, we need to do task analysis to inspect the user tasks and activities that is going to be the requirement of the UI.
3. **Step 3 Understand the principles of good screen design.**
   Before starting to develop the UI, it is important for the designer to understand the hardware constraint of the UI. The developer needs to know what kind of screen he is going to build the UI into. The designer needs to know at least the basic principles of designing a UI on various types and sizes of screen.

4. **Step 4 Develop system menus and navigation schemes.**
   The first part of designing the UI is to develop the menus and navigation schemes. This is important especially on a large and complex system. A good navigational system would greatly help the user to navigate through the UI. It will also help designers on their work by becoming the progress point of reference throughout the work.

5. **Step 5 Select the proper kinds of windows.**
   The definition of windows in this context is an area of the screen, which contains the UI. This step is especially important for complex system, which needs multiple windows to operate each part of the system.

6. **Step 6 Select the proper device-based controls.**
   For standard system, which uses regular PC, the control device is usually keyboard and mouse. But those two options might not be the best choice for some other type of system. A cash machine might only need a keypad or touchscreen. Drawing software would be best if it has support for stylus and/or drawing tablet. Modern smartphone manufacturers are now beginning to emphasize the usage of microphone as device control by letting users to speak out the command to a personal assistant software.

7. **Step 7 Choose the proper screen-based controls.**
   In contrast to device-based control, a screen-based controls are the elements on the screen that could perform as control mechanism for itself or other elements on the UI.

8. **Step 8 Write clear text and messages.**
   Writing a good wording for the UI is a step in UI design process that is often overlooked. A clear and meaningful words, messages and text can help increases the system usability.

9. **Step 9 Provide effective feedback, guidance and assistance.**
   Every user interaction on the UI should be given immediate feedback in some way. This way the user can have a sense of control on the UI. A good UI must provide this immediate feedback. It is also important for a UI to provide guidance and assistance along the way, especially on the part of the UI which is quite complex.

10. **Step 10 Provide effective internationalization and accessibility.**
    A system that is targeted for wide target users needs to pay attention to internationalization and accessibility. For example, a UI should take into account layout for language that use right-to-left script. Another example is providing good accessibility for blind people by developing a system in accordance to accessibility guidance from screen reader application.

11. **Step 11 Create meaningful graphics, icons and images.**
    Nice looking visual elements like graphics, icons and images can function as

attention grabber for the user. They will ease the user on operating the UI. Although, the usage of such visual elements needs to be done carefully so that it would not be distracting for the users.

12. **Step 12 Choose the proper colors.**
    Similar to the function of graphics, color also helps draw attention from the user. It can help creating a better structure of information on the screen. On the other hand, if used improperly, it could also has a bad impact on the system usability.

13. **Step 13 Organize and layout windows and pages.**
    At this point, all the necessary elements that are needed on the UI have been prepared. The windows have been defined, wording and text have been written, visual elements have been prepared, color schemes have been chosen. After they are all ready, it is time to organize everything in their respective places on the UI in a good organized layout.

14. **Step 14 Test, test, and retest.**
    Once the UI has been developed, it is important to do a thorough testing throughout the UI, This testing process needs to be done iteratively alongside the refinement of the UI itself.

3.4 UI Evaluation

In order to build a really good UI, a thorough evaluation needs to be done. We have previously mentioned about usability testing as one of the steps in iterative design. Usability testing is one of the most well-known approach to evaluate a UI. This chapter will examine four common UI evaluation techniques: usability testing, heuristic evaluation, software guidelines and cognitive walkthroughs [46].

Usability Testing

Usability testing is the most well-known method of evaluating a UI. It is a well-studied testing framework that provides systematic way to gather user's judgment on the UI. JS Dumas and J Redish in their book, A practical guide to usability testing, list five characteristics of usability testing [22]:

1. The main goal of the test is to increase the usability of the UI.

2. The participant of the test must be real users.

3. The tester must do real tasks.

4. The facilitator should observe and record what testers do and say.

5. Finally, result of the test observation should be analyzed to identify usability problems and suggest some changes to fix the problems.

Heuristic Evaluation

Heuristic evaluation is an approach to evaluate a UI by involving usability experts. In this method, several usability experts will be invited to have a look at the UI. Those experts would then be asked about their opinion about the UI based on their knowledge and experience in the field of usability. These experts will naturally base their evaluation on some set of rules. These rules can be in the form of usability guidelines and principles. The evaluator might already have their own principles or they might acquire it from another trusted sources [66].

Software Guidelines

This approach works by letting the UI developers to do the evaluation. They will be given a guideline document. This guideline contains recommendation about how should the UI be designed. The guideline could contain some rules about layout, colors, font styling and/or any other design related recommendation. The UI will then operate the UI and compare every aspect of the UI to the guidelines. More matching between the UI and the guideline indicates high usability. This approach alleviates the need for an expert as in heuristic evaluation method [46].

Cognitive Walkthroughs

Cognitive walkthroughs method takes into account the fact that most users prefer to learn a UI by directly using it to accomplish a task rather than following some manual guidelines. This method has two phases. The first phase is called preparatory phase and the second phase is the analysis phase. In the first phase, the evaluator will prepare four things that will be used as input for the next phase [78]. Those four items are:

- **User population**. It is important to clearly define the user population of the test. In this part, the evaluator is enforced to write down the exact target users of the test.
- **The tasks**. This is the description of the tasks that need to be done by the users/testers.
- **Action sequences of the task**. This contains detailed sequence of actions that the users/testers need to take to accomplish the task.
- **The UI**. Finally, the UI itself is going to be the object of the test. In this case, the UI can be in the form of live prototype as well as paper draft.

After those four items have been prepared, the second phase can be started. The testers will be told about the task that they need to accomplish during the task along with the sequence of action to achieve that task. The tester will then proceed with the UI and try to follow the sequence of action. During this step, the evaluator observes the actions done by the tester and then tell and evaluate their story about the actions that was taken by the users. Wharton et al. suggest these four questions to construct the story [78]:

- Will the user try to achieve the main task by executing action on the sub task?
- Will the user notice that the correct action to achieve some task is available?
- Will the user acknowledge the correct action with the effect that they try to achieve?
- Will the user perceive that they are progressing toward achieving the task if they took the correct action?

Chapter 4

Web Application Development

This thesis work is required to be built on top of the existing ABB front end dashboard system. This system will be explained later in-depth on chapter 6. The system that they have is in the form of a web application. More specifically, it is a Javascript based single page application. Thus, it is imperative to investigate the topic of web application development as well.

4.1 Web Technologies Overview

This section contains a brief overview of web technologies that are being utilized in this thesis work.

4.1.1 HTML5

HTML is short for Hyper Text Markup Language. HTML5 is the latest version of HTML. HTML is the main language for creating web pages and web applications. Web browsers will immediately render the HTML file that it receive. A standard HTML file contains multiple HTML elements. HTML elements is an individual component of an HTML page. Each element is written with the name of the element enclosed with with angle brackets. Optionally, each element could also have its own attributes. Each HTML element has different functions and will be rendered differently in the browsers. HTML5 comes with various new features including new elements such as <progress> to show progress bar and <video> to show video on the browser [41].

4.1.2 CSS 3

CSS is short for Cascading Style Sheet. CSS3 is the latest version for CSS. CSS is the language that is being used to give a style to any XML based language including HTML. The style includes colors, layouts, fonts and animations. CSS can be written both inside the HTML document and separately in its own independent file [26].

The syntax of CSS comprises of two components: selectors and declaration block. Selectors contain the syntax to select an element in the XML document that are being styled. The declaration block contains list of styling that are being applied to the selected elements.

SASS

SASS is CSS Preprocessor. It is basically an extension of CSS. It adds several functionalities, which do not exist on CSS such as variables, nested rules, mixins and inline imports. All those functionalities increase the efficiency of writing the stylesheet. SASS has two different syntaxes. The first syntax is called SASS and the second is called SCSS. SASS relies on indentation to separate selectors and declaration block. SCSS use similar syntax to CSS, it uses braces to enclose a declaration block [72].

4.1.3 Javascript

Javascript is a high-level, weakly-typed, dynamic, interpreted programming language. Ever since its birth, Javascript was primarily used as a client-side language. But since the advent of nodeJS, the usage of Javascript as a server side language starts to increase. As a client-side language, Javascript was used to provide interactivity on a web page.

Javascript specification is based on ECMAscript specification. At the time of this thesis writing, the latest version of ECMAscript is ECMAscript 2017. The latest version of ECMAscript comes with various new features that could help developers to code their application more efficiently. Some of the most popular features includes: arrow functions, template literals, promises, classes and modules [23].

4.1.4 Ajax

Ajax is short for Asynchronous Javascript and XML. The term was first coined by Jesse James Garrett in 2005 [31]. Ajax is not really a single monolithic technology, it is actually a set of client side technologies working together to provide asynchronous call for a web application. The most important underlying technology behind Ajax is XMLHttpRequest (XHR). XHR is a Javascript API that lets client to transfer data to a web server without the need to refresh the web page. This characteristic provides boost in terms of user experience. It leads many major website in the world to implement Ajax. This development contributes to further advancement of Javascript.

4.1.5 Websocket

Websocket is an internet protocol that enables two way communication between a client and a remote host that has opted-in to allow the communication between the two parties [27]. The API for websocket protocol has been standardized by W3C [40]. Websocket protocol is an independent protocol, which relies on TCP connection. By opening a two way communication channel between the client and the server, websocket allows for a more rapid interaction between the two parties. This is the reason why websocket has become the main technology for real time application.

4.2 Web Application Architecture

Len Bass et al. in their book, Software Architecture in Practice, define software architecture as the set of structures needed to reason about the system, which comprises of software elements, relations among them and properties of both [9]. This definition applies to any kinds of software application including web based application.

Web based application works on top of web platform, which relies heavily on internet protocol. This characteristic gives general flavor to the architecture of web based application. The architecture of a web application is often characterized by its composition of elements between the server side system and the client side system. The following sub chapters will explain two different types of web application architecture with regard to its composition on client-side and server-side system.

4.2.1 Client-Server Architecture

In this architecture, the client makes the request to the server. The server responds to the request, executing any command that needs to be executed based on that request, and then generates an HTML page. This HTML page will then be sent back to the users to be loaded on their browser [29]. All the logics and data manipulations are done on the server. This way the purpose of the client side is just for presentation. This is why this architecture is sometimes called thin-client architecture.

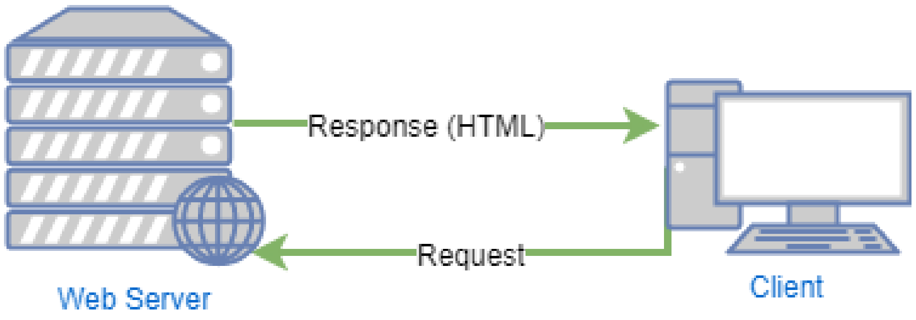


Figure 4.1: Client server web application architecture schematic

This is the most traditional architecture of a web application. This architecture was once the most dominant architecture of web application. Prior to emergence of various new web technologies, the only way of building a web application was with this architecture. It utilizes the basic protocol of the web, HTTP, along with the standard HTML, CSS and Javascript. Usage of Javascript in this architecture is mostly only for eye-candy purpose.

4.2.2 Service Oriented Architecture

Service Oriented Architecture (SOA) is an architectural approach where software is built as composition of independent services. The services are provided through a published and discoverable interface. The application component will maintain connection with the services through a messaging protocol over a network [25].

In the case of web application, SOA is implemented with web service. There are many technologies that can be used in order to build a SOA web application. Some of the most popular are SOAP, WSDL and REST [24].

Practically, in a SOA web application, the client will first make the request to retrieve the application from the server. The server will then respond by sending the whole application to the server. This application usually comes in the form of a single HTML page and some Javascript files. After the client receive the application, all operations of the application can be done on the client side. Logics can be dealt on the client side. Changes of data is communicated to the respecting web service and handled by the Javascript code. The most popular format to transfer messages between web service and the client is JSON. In some cases, this architecture can also be called thick or fat client because the client handles much more than only presentation.

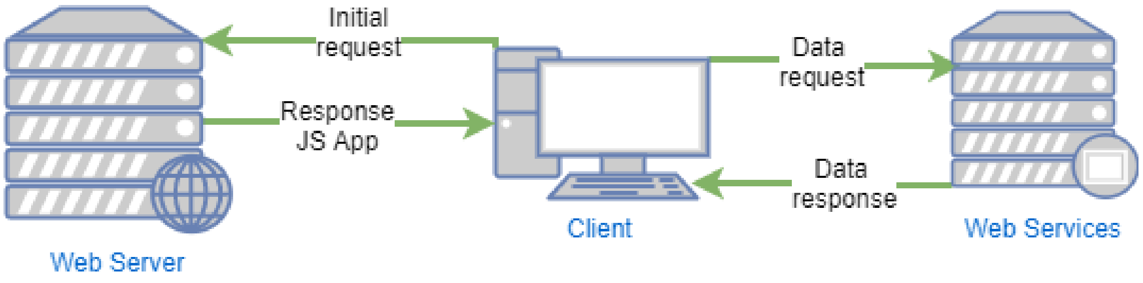


Figure 4.2: Service oriented web application architecture schematic

Chapter 5

Parser and Automata Theory

One important part of this thesis work is a well integrated code editor inside the whole IDE user interface. This code editor needs to be able to provide smart auto completion so that it will make it easier and more comfortable for the end user of this system to add new calculation code. Thus, a parser is needed to give the suggestion list on the auto completion feature. To develop the parser, several design decisions have been made (more on this on chapter 7). One of the design decisions is to build a client based parser which will read the input code and build a parse tree out of it. This chapter inspects the theoretical background of parser development.

5.1 Overview

Dick and Ceriel in their book, Parsing techniques, a practical guide, define parsing as the process of structuring a linear representation in accordance with a given grammar [20]. They intentionally kept the definition abstract to allow wide interpretation. It is because parsing technique that they discuss in their book can actually be applied in various fields and for various types of input data such as natural language sentence, sequence of geographical strata, a piece of music and, of course, a source code of a computer program.

For the context of this thesis, parsing is narrowly defined in computer science terminology. Parsing is the process of examining strings of tokens, which come from a source code and determine whether each individual token obeys certain syntactical structure. This syntactical structure refers to explicit convention in the syntactic definition of the programming language [5].

With that definition of parsing, we can infer that a parser is a computer program, which reads source code in a programming language as the input string and checks whether the source code complies with the syntactic definition of the programming language.

Parser can be used in various ways depending on the type of the input. In the case of natural language input, the parser is usually in the form of a standalone application, which extracts some information from the text or builds a grammatical tree structure of the sentences. For usage of reading markup language like XML and HTML, parser comes in the form of a file reading tool, which reads the markup text and builds the hierarchical structure of the file. In the case of HTML, the parser is built in the browser and would generate the DOM to be presented as web page. For most computer programming languages, parser is included in the compiler or intrepreter software. The following chapters will focus deeper into parser for a computer programming language.

A parser can either be written manually or generated automatically by a parser generator based on the supplied grammar. Manual parser construction ensures good error recovery and it is more flexible in terms of manipulating the parser execution. But, it takes a lot of work to develop. Parser generator alleviates this problem, a parser can be generated in minutes given the supplied grammar. There are already many ready-to-use grammar files for various well-known programming languages. The downside is that they are quite complex and rigid giving less freedom to the developer or the user of the parser to customize the inner working of the parser.

5.2 Parser in a Compiler

The way a machine executes a computer programming language is by translating that language into machine readable commands. This process of translation is called compiling and it is done by a special computer program called compiler.

According to Aho and Ullman, in their book, The Theory of Parsing, Translation and Compiling, compiling process in general can be divided into six steps [5]. These steps are not rigid. Practically, some steps might be done in one single steps. Some other steps might be split into more steps. Those six steps are as follows:

1. **Lexical analysis**. Process of turning stream of characters in tokens.
2. **Bookkeeping**, or symbol table operations. Process of managing list of identifier existing in the source code along with its detailed information.
3. **Parsing or syntax analysis**. Process of determining whether the token matches the valid syntactic structure of the language.
4. **Code generation** or translation to intermediate code. Process of generating a lower level intermediate code (assembly).
5. **Code optimization**. Process of optimizing the generated intermediate code.
6. **Object code generation**. Process of converting the intermediate language into executable machine language.

The first four steps described above is called compiler front end and the rest is called compiler back end. Compiler front-end is part of the compiler, which analyzes the source code and then builds an intermediate representation of the source code. An intermediate representation is a representation of a program between the source code and target languages. To build the intermediate representation, the compiler would utilizes the two results of source code analysis in the compiler front end phase: symbol table and abstract syntax tree. Generally, both symbol table and abstract syntax tree are produced and maintained by the parser.

5.2.1 Symbol Table

Symbol table is a data structure, which stores every entities on the source code (variables, methods, classes etc) along with its detailed information such as its declaration, type and position on the source code. For most programming language, symbol table is built for every scope definition on the source code. This is because identifier with the same name could be declared on various scopes on the source code. For example, i and x are often used as a generic variable name for quick purpose like for-loop iteration and this could be used in many parts of the source code [4].

Depending on the programming language, a symbol table maybe utilized in the following ways [5][4]:

- To check for the semantic correctness of a source program.
- To be used as a resource for generating code.
- To determine the scope and location of an identifier.
- To provide some useful data on interactive debugging session.
- To be referred on the creation of a diagnostic report, either during or after execution of the application.

The data structure used on a symbol table could be anything from a simple list, binary tree to hash table. But implementation of most major programming language compiler uses hash table. This is because symbol table needs to be rapidly read and written during the analysis process. Hash table provides the fastest way to do those operations. The key in the hash table would be the name of the identifier and the value would contain all kinds of information for that particular identifier [4].

5.2.2 Abstract Syntax Tree

The main output of parsing is parse tree. Parse tree in computer science terminology is often called concrete syntax tree. A parse tree is a rooted hierarchical tree data structure that represents the syntactic structure of an input string [13].

Abstract syntax tree, or often abbreviated to AST, is a form of parse tree, which represents the syntactic structure of a source code. It is different to parse tree in a way that abstract syntax tree does not store all the unnecessary and non-essential information from the input source code [55]. For example, parentheses of an if expression in some programming language is treated as non-essential, unlike the expression itself. In parse tree, these parentheses would still be stored.

Robert Harper describes abstract syntax tree as an ordered tree whose leaves are variables and whose interior nodes are operators whose arguments are its children [36]. In this case, variable refers to a generic piece of syntax from some programming languages. This format of variables could differ from one language to another. Hence, abstract syntax trees can be categorized based on each programming

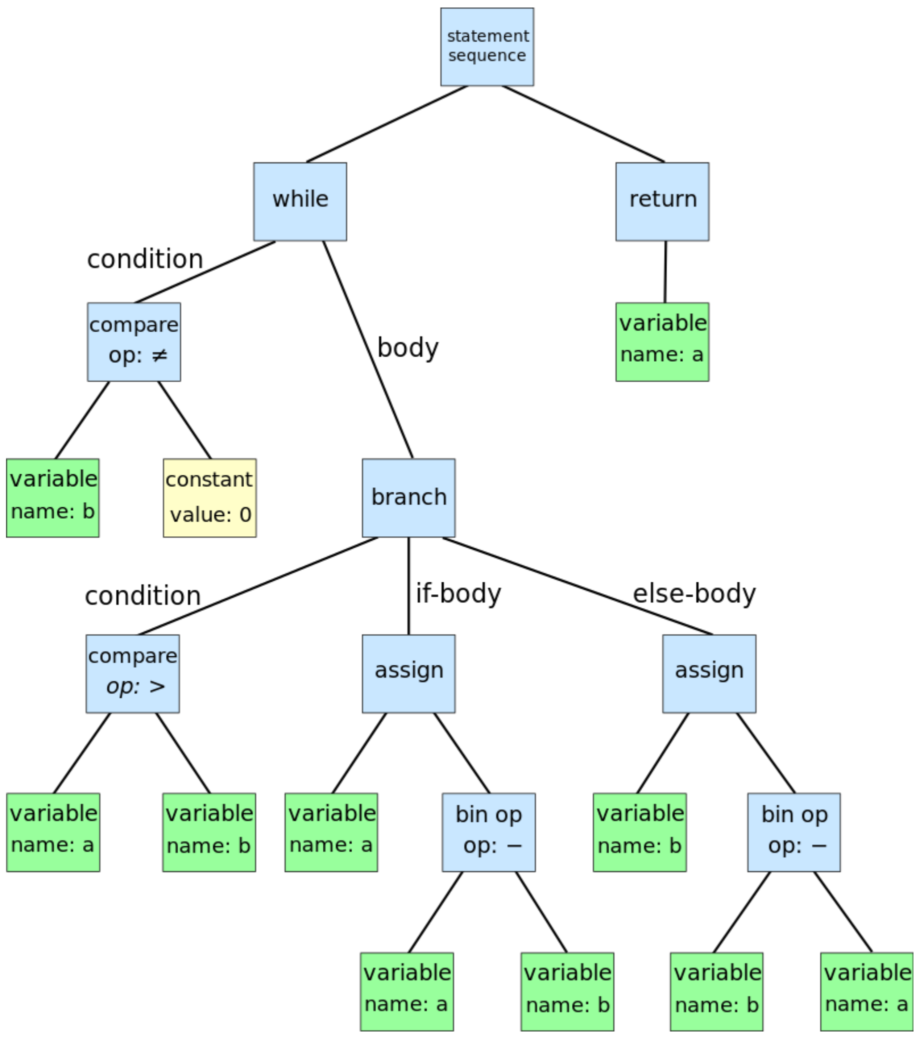


Figure 5.1: Example of an abstract syntax tree [19]

language syntactic categories. One example of syntactic distinction between programming languages is in terms of expressions and commands. For language with a certain syntactical expression, each interior node represents an operator whose children of it represent the operands [4].

5.3 Parser Types

In terms of how the input can be derived from the grammar, a parser can be differentiated in two big categories [20][4]:

- **Top-down parsing**
  Top-down parsing is an approach to parsing by finding the left-most derivations of an input stream. This approach constructs the parse tree starting from the root and creates nodes along the way down the parse tree in order. One of the most common approach to top-down parsing is recursive descent parsing.

Recursive descent parsing works by constructing the parse tree from the top using recursive procedure. There are two kinds of recursive descent parsing:

1. Back tracking
   Recursive descent with backtracking is a top-down parsing technique that works by trying each production rule by doing back tracking. The parser would then determine which production rule to use. This parsing technique may require exponential time because of the back tracking process through the production rule. This exponential time could still occur even if they terminate.
2. Predictive parsing
   In contrast to the previous technique, predictive parsing does not need back tracking. It has the capability to predict, which production rule to use to replace the input string. The way this parsing technique predicts is by using a look-ahead pointer. Look-ahead pointer is a pointer, which refers to the next token. With this approach, predictive parser can only accept the kinds of grammar, which is called LL(k) grammar. This technique employs use of stack and a parsing table. Both of them contains an end symbol $ to specify empty stack and that the input is consumed. In the parsing process, it will look into the parsing table to make decision on the input and stack element combination. Unlike in the recursive descent with backtracking parsing, predictive parsing will only have one production rule to choose. As a result, there is a possibility that no production rule matches the input string. This will cause the parsing procedure to fail.

- **Bottom-up parsing**
  In contrast to top-down parsing, bottom-up parsing starts with the initial input and then tries to rewrite it to the start symbol by moving up through the grammar rule tree. It starts from the leaf node of the parse tree and traverses up through the parse tree until it reaches the root node. This parsing technique is also called Shift-Reduce Parsing. The name comes from the fact that it works in two steps as follow:

  1. Shift step
     This step refers to the shifting of the input pointer to the next input symbol. That next symbol is called the shifted symbol. This symbol represents a new node on the parse tree and will be pushed onto the stack.
  2. Reduce step
     This second step happens when the parser finds a complete grammar rule on the right hand side of the production rule. It will then replaces it to left hand side of the production rule. At this point, the top of the stack will contain a handle. The stack would then be reduced by doing pop function on it, which replaces the handle with the left hand side non terminal symbol.

5.4 Parsing Process

There are two different approach of parsing process. The first approach is called scannerless parsing. Scannerless parsing is one approach of parsing where the lexical analysis and syntactic analysis are done in one step [77]. The other approach breaks the two processes into a pipeline. The two steps in parsing process is more common for general programming language. Those steps are lexical analysis and syntactic analysis.

Lexical analysis is the process by which the stream of input source code is tokenized. Every characters in the code would be read and and be broken into words and/or symbols. This process is done by a lexer. In the case of a programming language, the tokens would be all identifiers, keywords, constants, strings and parentheses and all the operator symbols. Lexical analysis works in two step process. The first step is scanning and then immediately followed by the second step: evaluating. The scanning step is usually based on a finite state machine. The scanner would scan through all the input characters while maintaining state for each scan. With this state, it determines the possible sequence of characters that can still be counted as one token. The second step is evaluator. After the scanner finds the stopping characters for a token, it will then pass it to the evaluator to determine the type of this token.

Syntactic analysis is the process of parsing itself. In this second step, each tokens will be checked if it conforms the allowable expression. This conforming process is often done by referring to a context-free grammar. The parser would run each token through the grammar component and define to which grammar rules it is comply to. It will then built the abstract syntax tree based on this token checking process through the grammar.

All parsers can be generalized to work by utilizing two components. The first components deals with making substitution and recording a parse tree. The second component works as a control mechanism that decides the next move for the parser. In the field of computer science, the substituting component is called non-deterministic automaton [20].

5.5 Non Deterministic Finite Automata

Non deterministic Finite Automata (NFA) is a subset of Finite-State Machine (FSM). FSM, or Finite-State Automata (FSA), is an abstract machine which has a finite number of states and the state will change when there is an external input coming to the machine [34]. The term "non-deterministic" is used on this FSA because this automaton can have several possible moves in response to an input. The choice of the next move is not predetermined. This is different than Deterministic Finite Automata (DFA), where each state can only have one possible next move depending on the input [64]. A DFA is actually a special subset of NFA.

NFA is chosen for the implementation of parser for a reason. The non-deterministic property of a finite state machine contributes to a substantial efficiency in terms of processing a computer program through an automaton. It makes it possible for us to write the code in higher level language. What parser does is to construct a parse tree from a high level language source code using NFA [64]. The parse tree would then further be compiled into a deterministic automaton which can be executed by the machine.

NFA works in a parser by managing three elements. The first element is the input string as the object of the parsing process. The second element is the (partial) parse tree which keeps updated along the way until the parser reach the terminal state. And the last element is the internal administration which handles the information being sent from the parsing of the input string toward the parse tree [20].

Chapter 6

ABB Ability^{TM}

This thesis work is built on top of ABB Ability^{TM} platform. To give a clearer context, this chapter will explain some aspects of the existing ABB Ability^{TM} platform, which are connected to this thesis work. This whole chapter is referenced from unpublished internal ABB document. Chapter 6.1 will provide the general description of what is ABB Ability^{TM} and ABB Ability^{TM} platform. The rest of the chapters will explain parts of ABB Ability^{TM} platform, which are connected to this thesis work. Chapter 6.2 will explain about cpmPlus History or sometimes just referred as "History" in short. Chapter 6.3 describes cpmPlus View or just "View" in short. The last chapter (6.4) contains description of ABB's calculation engine.

6.1 Overview

ABB Ability^{TM} could be seen as a complete portfolio of ABB digital solution. ABB digital solution itself is the kind of solution that lets ABB deliver value to the customers through digital and connected technology. So far, ABB already has installed base of more than 70 million connected devices, 70,000 digital control systems and 6,000 enterprise software solutions [1].

All those devices, control systems and enterprise software solutions are currently mostly disconnected with each other. They are all deployed at the customer site. They work in their own silos of business unit. There is no common way to integrate and visualize all of the information that could be collected. This is the driver to the development of unified ABB Ability^{TM} Platform.

ABB Ability^{TM} platform is a set of common, integrated and standardized digital enabling technologies. These set of technologies are used to build the ABB Ability^{TM} solutions. These technologies exist at the device, gateway, and cloud level. It makes it possible for ABB to develop solutions more quickly, reliably, and securely. Solutions developed on top of the ABB Ability^{TM} platform will be more integrated and work in a consistent manner. Future development of ABB digital solution from all business units will always be built on top of ABB Ability^{TM} platform.

ABB Ability^{TM} platform has many components included with it. Here are list of those components which are related to this thesis work:

- cpmPlus History
- cpmPlus View
- cpmPlus CalcEngine

6.2 cpmPlus History

cpmPlus History is a proprietary time-series database engine built by ABB. More specifically, it is a process historian software, which comprises of several independent but integrated software technology components. All those components work together to provide a highly scalable software platform to build products and systems for manufacturing process industries and utilities. Many ABB solutions are utilizing cpmPlus History. This is a list of example solutions that are based on the cpmPlus History:

- Power plant information and condition monitoring systems.
- Data logger for various high speed data producers, such as variable speed drives, power distribution devices, analyzers, and vibration monitoring systems.
- Process data warehouse systems.
- Overall equipment effectiveness (OEE) reporting systems.
- Electricity distribution automation and SCADA historian systems.

6.2.1 Features and Functionalities

The core of cpmPlus History is a relational database system called RTDB. This RTDB is equipped with built-in columnar features for optimized time series signal processing and storage. cpmPlus History implements the standard data models for process information, events/alarms, and equipment model. Equipment model is a dynamic modeling tool for application specific data models.

Beside the core historian functionality, cpmPlus History also contains a full functional platform to build solutions from data acquisition, processing, and storage to analytics, public interfaces, and visualization. Here is a list of some functionalities offered by cpmPlus History:

- Data acquisition connectivity from control systems and devices.
- Input stream processing with alarm detection, automatic aggregation functions, and integrated event notification to the users of the data.
- Data storage with redundant system configurations.
- Engineering tools and APIs for integration with remote engineering such as DCS or SCADA system.

- SDKs and hosting services for analytical applications.
- Public interfaces for bidirectional data access including OPC DA, HDA, and UA servers, and ODBC/SQL.

6.2.2 Data Abstraction Interface (DAI)

In term of system architecture, ABB Ability^{TM} platform implements a data abstraction interface on its digital solution. This data abstraction interface will act as a bridge between the client layer and the data provider and storage layer. Client layer is where all the applications reside. Data provider and storage layer contain various types of data and storage engine, including ABB's own RTDB. The DAI is domain-independent middle ware technology. This interface hide the details of the actual data storage to make the client data access transparent and independent from the data storage. This interface uses secure websocket for communication between clients and servers as well as in system-to-system communication. DAI provides means to transform the data structures from various data sources to a unified object model and publish them in the standard interface. For example, what constitute as a "table" in traditional relational database model is called "class" in this DAI. Table rows will be mapped to "instance". Table column is called property.

6.3 cpmPlus View

cpmPlus View is a web based front end dashboard system. It is a tool to build application user interface. cpmPlus works as a single page application. It is built with HTML5 and vanilla Javascript with internally developed framework. cpmPlus View API provides asynchronous data access from web browsers to the data sources that are connected to the cpmPlus Data Abstraction Interface (websocket server). It has two main modules: dashboard editor and SDK.

6.3.1 Dashboard Editor

Dashboard editor is a tool to build and layout a dashboard using ready-to-use widgets without much need of programming. With the dashboard editor, the user can just drag and drop widgets of their choice and configure them however they want. The client widget library contains basic business graphics and samples on how to implement your own or to embed third party widgets.

Widgets

The dashboard editor already has several default widgets that can be used. The default widgets are categorized into five categories:

- **Dashboard basics**
  This category contains some generic widgets with various purpose that might

be needed to construct the dashboard. Below is the list of widgets of this category:

  - **Dashboard**. Every dashboard can function as an independent widget on another dashboard with the help of this dashboard widget.
  - **Horizontal splitter**. This widget is used to create a splitting point between widgets horizontally. With this splitter, widgets that is attached to this splitter can be resized horizontally.
  - **Vertical splitter**. Similar to the horizontal splitter but this one allows for vertical resizing.
  - **HTML**. This widget has two modes. On the first mode, it can show any HTML script that is put by the users. The other mode is that it will act as an iframe if the user provides a link to the external source for the frame.
  - **Label**. This widget is meant to be generic label.
  - **Panel**. This widget acts as generic panel that can be used for various purposes.
  - **Top bar**. This is a default top bar, which contains standardized top bar for various ABB application. This widget contains a logo of ABB and can be inserted with a navigation menu.
  - **Theme selector**. This widget allows users to switch the theme of the dashboard.

- **Controls**
  This category contains widgets that function as control mechanism on the dashboard. Below is the list of widgets of this category:

  - **Button**. This widget acts as generic button for various purposes.
  - **Input**. This widget acts as generic input text for various purposes.
  - **On/off slider**. This widget acts as an on/off switch. Similar function to HTML checkbox.
  - **Single property editor**. This is a highly robust widget. This widget can edit any property of a class on the database. The format of this editor adapt to the type of the property. If the property being set on this widget is of text data type, this widget will show an input text. If the property is in a form of enumeration, this widget will show a selector and so on.
  - **Time bar**. This widget shows time bar, which can be used in conjunction with any other widget to show time constrained data.
  - **Time input**. This widget allows user to input time value in a more convenient way than regular input text.

- **Data browsing**
  This category contains widgets for browsing data in various ways. Below are some of the important widgets of this category:
  - **Data list**. This is one of the most important widgets in the dashboard editor. This widget lists data from a class in complete tabular format. It allows user to add, edit and delete instance directly from within the table.
  - **Property grid**. This widget shows list of properties of a class in grid format, user can edit the value of each property from this grid.
  - **Property list**. This widget shows list of property of a class in list format.
- **Data visualization**
  This category contains widgets for visualizing data in various ways. Below is the list of widgets on this category:
  - **Chart**. With this widget, user can show some data from the database in chart format.
  - **Gauge**. With this widget, user can show a data from the database in gauge format.
  - **Legend**. This widget acts as legend for any other visualization widgets.
  - **Pie chart**. This widget shows pie chart based on the selected data from the database by the user.
  - **Radar chart**. This widget shows radar chart based on the selected data from the database by the user.
  - **Ruler**. This chart shows ruler as guidance for other data visualization widgets.
  - **Status widget**. This widget shows customizable status based on some configuration and data source from the database.
  - **Value**. This widget shows value of an instance property of a class.
- **Navigation**
  This category contains widgets that deal with navigation throughout the dashboard. Below is the list of widgets of this category:
  - **Navigation button**. This button is used to direct the user to another dashboard.
  - **Navigation tree**. This widget contains list of dashboards that can be navigated by the user.
  - **Pivot navigator**. This widget allows user to swipe left or right to switch between dashboard.

6.3.2 SDK

SDK is the development kit for extending the functionality of cpmPlus View. It contains tools, API and functionalities that will make it easy for a web developer to build his/her own widget or construct a specially customized dashboard. The SDK contains:

- web socket server API to access the data in the data sources.
- Driver SDK to implement the connectivity to the data sources. Deliverable includes drivers to cpmPlus History, OPC DA, HDA, and SQL databases.
- JavaScript client API to access the data in the data sources through the websocket server.
- Widget library with trend chart, chart ruler, data list, gauge, label, pie chart, time bar, value, and value bar widgets.
- Base widget to implement new widgets or wrapping 3rd party widgets.
- Examples and tutorials of display and widget implementations.

6.4 cpmPlus Calc Engine

cpmPlus Calc Engine, or CalcEngine, is a tool, which is created to implement and execute calculations on a predefined data. This predefined data can be accessed via the data abstraction interface, which is mentioned in the cpmPlus history chapter. The data could be in the form of numerical real-time and historized process data, events and alarms, notification and other type of application specific data.

The calculation itself is a logical statement and/or an equation that could have input and output data. The calculation should be written in C# code. The calculation engine would then execute the calculation code based on its other configurations.

cpmPlus Calc Engine is an independent technology component. It can be used with cpmPlus History. The current implementation of CalcEngine uses ABB's proprietary real time database engine called RTDB. This database is used to store all its calculation code and configurations. CalcEngine would constantly look through any configuration data that are inserted to its classes on RTDB and execute any entries that need to be executed.

6.4.1 Architecture

CalcEngine has two sets of tables (or in the case of cpmPlus history, it is called classes). The first set is called implementation classes and the second is operation classes. The reason behind this separation is so that it is possible to use the same logic for multiple parameter sets and task configurations.

Implementation classes contain classes, which define the calculation logic along with all its parameters. These parameters is going to be used as input/output data of the calculation. Below is the list of implementation classes:

- **CalcDefinition**
  Defines the calculation logic together with the associated CalcParameter. This is the class, which contains the calculation code. One of the properties of this class is CodeType. This is where the users define, in which language do they write the calculation code. The options are C#, Simplified C# and VB. VB option is about to be deprecated. Simplified C# means that the engine will wrap the input code with a boilerplate code like below:

```
namespace ABB.Vtrin.CalcEngine
{
    public class CalcClass_NameOfTheCalcDefinition :
       CalcInstance
    {
        public void Calculate(ref double A, double B)
        {
            A = A + B;
        }
    }
}
```

Listing 6.1: Sample boilerplate code

  Non simplified C# will execute the code, which is written by the user as is, so it gives more flexibility. With non-simplified mode, user can add their own library, namespace, class, etc.

- **CalcParameter**
  Represents a parameter, which CalcDefinition uses either as an input or an output. So, one of the property of this class is a reference to CalcDefinition. Another class, which is referred by this class, is CalcParameterType. This class defines the data type of the parameter being used. This class also has a property that define array dimensions of the parameter, it could be in any positive integer number. If the parameter is not an array, it should be set to 0. Another important property that this class has is IsOutput that define whether this is the parameter, which is being used by the user to write the calculation result.

- **CalcParameterType**
  This class contains a list of available data types for CalcParameter. All entries in this class are generated by the CalcEngine backend.

- **CalcDependency**
  This class represents an external dependency such as a DLL or code snippet that can be attached to a CalcDefinition. There are two types of dependency that the user can choose:

  – EquipmentType. It refers to an equipment model in the database.

  – External Library. It refers to an entry in CalcExternalLibrary.

- **CalcExternalLibrary**
  Contains external code/DLLs used by CalcDefinition. There are three types of library that the user can choose:
  - Code Snippet(C#). If this type is chosen, the user would have to write the code snippet in this class.
  - DLL(File path). If this type is chosen, the user would have to put the absolute path to the DLL (in which the Calc Engine process must be able to access this)
  - DLL(BLOB stored in the database). With this option, the user would have to upload the DLL to the database and it shall be store as a byte string.

On the other side, operation classes contain mappings of actual data to the parameters combined with some task configuration like scheduling, process management and batch jobs. Here is the complete list of operation classes:

- **CalcProcess**
  This class contains listing of system process executing CalcTask, which are assigned to it. This class defines properties such as process priority, number of threads and mode (either to run as 32-bit or default 64-bit).
- **CalcTask**
  A CalcTask represents an executable calculation. It joins a Calc Definition and Calc Parameters with mappings to actual values, and can be executed by various CalcTaskTrigger. This class contains a property that defines the constructor parameter for a calculation. It contains the set value of a parameter in the calculation. It is stored in JSON. There are also various properties that control how the task will be executed.
- **CalcMapping**
  This is where the user can select the data from the data abstraction layer to be used as a parameter in the code later on. After selected here, the entry would need to be connected to the CalcParameter via CalcParameterMapping. There are two most important properties on this class:
  - ***ClassName*** *: string*
    Name of the class from the data abstraction layer, which the instances should be fetched
  - ***WhereString*** *: string*
    Optional string to limit which instances are fetched.
    (e.g., "Name='MyVariable'")

  In addition, there are also other properties that define external data source. On this properties, the user would have to put the connection string to the external data source.

- **CalcParameterMapping**
  This class combines a CalcMapping and a CalcParameter for use by a CalcTask. So, this class has properties that refer to CalcParameter, CalcMapping and CalcTask. Apart from those properties, there are also properties that configure this parameter mapping such as setting only the property of the mapped class instance instead of the instance itself.

- **CalcTaskTrigger**
  A trigger that can be attached to a CalcTask to execute the calculation. There are three types of task trigger:

  - **Periodical scheduler**. With this type, the task would be triggered periodically based on the scheduler that has been defined in CalcScheduler.
  - **Event based**. With this type, the task would be triggered based on value change that happens on a CalcMapping.
  - **Triggered by other calculation**. With this type, the task would be triggered based on execution of another task.

- **CalcScheduler**
  Defines how to run a CalcTask periodically. The scheduling properties of this class includes start time, end time, execution internal, unit of execution interval and base time.

- **CalcBatchJob**
  This class is used for scheduling batch jobs for CalcTasks, for example, when wanting to perform recalculation.

- **CalcTaskDiagnostics**
  This class contains diagnostic information of a task.

- **CalcSimulation**
  This class is used for testing a calculation without actually committing any values to the database. Real values are read for the configured period, but the written values are returned as JSON instead of being committed to their actual destination.

Chapter 7

Requirements

The first research question of this thesis is "What are the requirements of the web based IDE for ABB Ability Calculation Engine?". To provide answers for that question, extensive requirement analysis needs to be done. This chapter contains in-depth requirement analysis of the web based IDE system. Chapter 7.1 will provide general overview of the requirement analysis process of this thesis work. The next chapter (7.2) contains the stakeholder analysis for the web based IDE, which resulted in the user profile. Chapter 7.3 describes several use cases of the web based IDE. Chapter 7.4 explain some issues on the approach of the user interface development. Chapter 7.5 contains complete listing of features and functionalities that the system should have. The last chapter (7.6) lists some non functional requirements for the system.

7.1 Overview

ABB already has the calculation engine back end running. There is already a detailed internal document containing requirement specification for the calculation engine. This document functions as the initial basis for further requirement analysis specific for the front end development. The requirement analysis process starts by defining the user profile. Since, of course, the user profile of calculation engine back end and front end is the same, this part is largely adapted from the calculation engine backend requirement document. The same thing applies to the next part of the process: use case analysis. In this part, the use cases, which have already been listed on the backend requirement document, are adapted to front-end requirements. The next part is UI consideration analysis. Here, I determine some issues regarding the expectation of the user interface and how it should be developed. After that, I make a list of functional and non functional requirement for the software based on all the information from the previous chapters.

7.2 User Profile

First step on the requirement analysis process is to do stakeholder analysis. In this case, we narrow down the stakeholder only to those people who will directly operate the Calc Engine. The Calc Engine is considered to have several different user groups:

1. **Subject matter experts**
   Subject matter expert is an expert in a particular field on a business unit. One of their tasks, which is related to the Calc Engine, is to calculate KPIs for the particular piece of equipment type that he knows in details. They are less likely to have in depth knowledge of programming languages, but they know math. So, they definitely know what kind of mathematical calculation is needed to measure the KPIs of the device that they are handling. They would prefer to work the properties of the equipment template instead of the individual process variables. This way, they can develop one calculation and execute it for all instances of the equipment.

2. **Site engineers**
   They are the engineer who work at the customer site. They implement calculations to detect anomalies in a process. The calculation will be based on the data from a particular device combined with data from another system. They might also want to share the results of the calculations in the form of notifications to some defined groups of users. They may have basic level of programming skills, but they would prefer to be able to set up the calculations as simple as possible. They would appreciate easy maintenance and monitoring of thousands of calculations that they have made.

3. **Process engineers**
   They are engineers who implement some process calculations for the purpose of monitoring and reporting the process operation. In this case, there could be thousands of calculated variables on the plant level. They are likely to have basic programming skills. Furthermore, they are also likely to be able to learn normal control structures and calling external functions to implement advance calculations. For this advance and complex calculation, a good way to debug that could provide extensive diagnostics and validation is essential.

4. **Project engineers**
   They are the engineers whose task is to install, configure and maintain an ABB Ability$^{\text{TM}}$ digital solution, which includes the Calc Engine. This engineer would need an easy way to configure the Calc Engine. As well as a way to monitor the execution and resource usage of the Calc Engine.

7.3 Use Cases

Use case 1 – Cyclic Process Calculations with History Values

In this first use case, the user is a site engineer in a power plant. The user already has 2000 equations, in which the parameter of the function is coming from the database of historical raw sensor/device data. The user has in depth knowledge of the equation and how the plant process work. The user don't know much about programming languages nor usage of any software development tools like Ms Visual Studio. But, it is possible for them to learn basics of programming to implement their calculations. The user understands the concept of variable, historical data stream and also the quality status information of data.

Requirements:

1. All configuration should be done inside ABB's internal user interface framework (cpmPlus UI). This UI should be accessible from the user's PC so that they don't need to access directly to the database server.
2. There should be an easy way to configure scheduling execution of a calculation.
3. The user should be able to write the calculation code using C# language right from the browser.
4. The user should be able to choose any pre-stored scheduling configuration, which can be applied to many calculations.
5. There should be a way for the user to avoid any unnecessary structure on the calculation code such as namespace and class definition. There would be many calculations, which are actually just simple equations like `A = B + C`.
6. Variable from the database along with their attributes should be simply referenced in the code editor on the browser, e.g.,
   `NameOfTheVariableFromBD.nameOfItsAttribute` or
   `NameOfTheVariableFromBD.aFunctionOfThisVariable()`
7. The code editor should have auto completion feature. The auto completion should be able to provide list of variables and methods of an object so the user does not have to remember them. It should be fast and still have good usability even if the code has millions on variables.
8. The variable in the code editor should have a way to be connected to variable value from the database, either for read or write access.
9. The UI should provide a way to set the scheduling time period for a calculation.

Use case 2 – Batch based calculations

The user is an engineer dealing with production process. He needs to implement calculations based on some event on his production process such as calculating raw material and energy consumption for a paper reel. The calculation itself would be simple, but the user would need cumulation of historical data over some batch time period.

Requirements:

1. There should be way to trigger the execution of a calculation based on some event, i.e., value change of a variable from the database.
2. Calculation execution should also be able to be triggered by execution of another calculation.
3. User should be able to alternate between mode of execution for a calculation whether it should be by periodical scheduler, event based or triggered by another calculation execution.

Use case 3 – Equipment template based calculations

The user is an engineer, who works on energy efficiency analyses for a certain type of pump system. This type of system already has an equipment template stored in the database. The user wants to implement calculations for all instances of that pump system simply by selecting the equipment template as a variable on the calculation code.

Requirements:

1. There should be a way to connect the calculation with equipment templates so that the system can execute the calculation to all the equipment instances.
2. Beside implementing a calculation on an equipment template, it should also be possible to implement calculation on a single equipment instance.
3. The attributes of the equipment should be easily be referenced from the code editor. So, whenever the user uses an equipment instance or template as one of the variable in the calculation code, they can just simply type dot after that variable and the code editor should show list of its attributes as auto complete suggestion.

Use case 4 – Alarm/event analyses

There is a case where a statistical calculations need to be done for alarms and events collected, e.g., from the control systems.

Requirements:

1. There should be a way in the user interface to use OPC events and/or other classes, which exist in the database (or more exactly data abstraction interface). The system should provide a way to include the classes along with their attribute in the calculation code.

2. It should be easy for user to switch usage of classes and/or OPC events from a calculation.

Use case 5 – Storing event frames

In the case when an interesting event happened, the user wants to store some data with time period from around the time the event happened (event frame). This data could be attributes detected by his calculation. For example, when there is a machine failure, the user might want to store detailed high resolution information before and after the incident.

Requirements:

1. The system should provide a way to create and modify class instances from within the calculation. This could be in the form of ABB standard database engine's event frames or some user configured one.

Use case 6 – Initiating notifications

After making the calculation, the user might want the system to send notification in the form of email, SMS or other type of notification. For example, an asset monitoring system detects from its calculation that an asset needs to be recalibrated, and thus sends a work order request notification to the responsible person.

Requirements:

1. It is most likely that there is already a notification/alert class in the data abstraction interface being handled by a separate notification service. So, initiating a notification could be as simple as making a new instance in that class. With that in mind, the system should provide a way for the user to easily set the calculation to make new instance to that notification class.

Use case 7 – Accessing foreign data

In this case, a calculation needs some data from other systems. For example:

- ABB database system has a hierarchical data access. In this case, the user might want to use data from another level of hierarchy.
- Another example is usage of data from another system, i.e, an asset monitoring system needs maintenance data from plant maintenance system.

Requirements:

1. The user interface should provide a way for user to use the standard data abstraction interface connection string. The usage of connection string will automatically connect the system to the other database system given that the connection string is valid and the target data abstraction interface exist.

Use case 8 – Power to simple implementation

A subject matter expert on a business unit of ABB wants to implement some performance calculations on equipments, which he/she is responsible for. He/she does not want to learn complicated programming language just for this purpose. He/she wants a system where he/she can just use predefined functions on his own calculations. For example, an engineer in motor department wants to check the temperature integral of a motor from some time periods last year. He/she wants to know if the temperature of that electrical motor has reached the limit for its damage, which has been set by the manufacturer. He/she wants to know the maximum temperature the motor has ever reached.

Requirements:

1. The system should allow the user to implement calculations on an equipment template by using the attribute names of the equipment model. It should not be a concern for the user whether those attributes are of basic data types or references to other object types.

2. It should be possible for the user to implement this kind of calculation without much hassle with the help of the language structure and the code editor itself. For example, to get the maximum temperature of an equipment from the default time period, the user should be able to just use something like `NameOfTheEquipmentAttribute.MAX`. If the user wants to get the maximum temperature from different historized database class or from non-default time period, the user should just need to use something similar like this `EquipmentAttribute.GetMAX(AnotherHistoryClass, start, time_period)`.

7.4 User Interface Consideration

Based on the list of classes used by the calculation engine described in the previous chapter, the user interface needs to provide a way to Create, Read, Update and Delete (CRUD) an instance to all of those classes in a convenient and sensible way. The CRUD should not be in the form of common CRUD UI. In this context, what is meant by common CRUD UI is to show tabular view of each class with remove and edit button on each row and add new instance button for each class table.

On a web based application, the common CRUD UI can usually be generated automatically. The generation process is done by the web application framework of choice. The framework generates the basic CRUD UI based on the data model that we have supplied to the framework.

ABB already has this common CRUD user interface for the calculation engine. The user can use the ABB's default desktop application, which is used to interact with the data abstraction interface. But, it was proven to be inefficient and confusing for end user. From the cyber security perspective, this is also not an optimal solution because it requires the user to be logged in directly to the server via remote desktop.

With all the considerations listed above, it is clear that a new approach to the user interface is needed. The user interface should allow the user to update some data into the calculation engine classes without the need to access the server directly. The ideal user interface for this calculation engine needs to reflect the flow that most users have in mind when using the calculation engine.

By analyzing the user profile and the use case described above, we can hypothesize the basic work flow for most users as follows:

1. User who is responsible for some equipments know some available raw data from those equipments and they want to work some calculation based on those data
2. They would want to choose the data from the database
3. After selecting the data and making it available to be used on the calculation code, they would proceed on writing the calculation code
4. Upon writing the calculation code, they would need to set the name and give some description to the calculation
5. When they are done writing the calculation code, they will have to proceed configuring the execution of the calculation
6. Finally, they will submit that calculation
7. If there is something wrong with the code that they write or configuration that they make, the system should give a feedback so that they can fix it
8. If everything is well configured, the user should be redirected to a page which lists all the calculation that they have along with all its essential information.

7.5 Functional requirements

Based on the user profile, use case and user interface consideration described in the previous three chapters, it can be inferred that the user interface will need two pages or dashboards: list calculation dashboard and add/edit calculation dashboard. The following subsection will detail the list of features and functionalities that are needed on each dashboard.

7.5.1 List Calculation Dashboard

The list calculation dashboard should contain the list of all the calculations created by the user. List of features and functionalities that should exist in this dashboard are listed below:

- There should be an "add calculation" button, which will redirect the user to "add calculation dashboard".

- There should be a filtering features. This dashboard will potentially have thousands of calculations, so filtering functionality is imperative. The user should be able to type something on the filter input box and the system would show the list of the calculations, which have been filtered based on the appearance of the filter keyword on the calculation name or calculation description.
- Each calculation on the list calculation dashboard should have its own box, which will contain information about this particular calculation such as:
  - The calculation name
  - The calculation description
  - The active language mode that is being used on the calculation (whether C# or simplified C#)
  - List of parameters that is being used on the calculation. Each list should show the name of the parameter along with its type.
  - List of dependencies that are being used on the calculation. Each list should show the name of the dependency along with its type (whether it is and external library or an equipment property).
  - List of tasks, which use this calculation. Each task should show its list of parameter mapping and details about the task trigger.

  In addition, each calculation box should also have these components:

  - **Remove button**. Every time the user clicks the remove button, there would be a confirmation dialog asking the user if he/she really wants to remove the calculation. Clicking yes would mean that the calculation along with all its attribute (parameters, dependencies and tasks which use it) is deleted.
  - **Open button**. This button will let users to open and edit the calculation.
  - **Clone button**. With this button, the user can easily copy a calculation and make new edits in there.

7.5.2 Add and Edit Calculation Dashboard

This dashboard will contain a big form where the user can fill all the necessary information to add and/or edit a calculation. All the information would be used to create a new or edit an existing instance of calculation engine classes which relates to a single calculation definition. Those classes are CalcDefinition, CalcParameter, CalcDependency, CalcTask, CalcTaskTrigger, CalcMapping and CalcParameterMapping. Below is the list of all features and functionalities that should exist here:

- To fill all the property of CalcDefinition, this dashboard should show a code editor, a field to fill the name and description of the definition and also a checkbox to switch between language mode (C# and simplified C#)

- A good code editor is paramount for this dashboard. Below are list of features that should exist on the code editor:
  - **Syntax highlighting**. The supported language is C#, so the code editor should be able to show good syntax highlighting for C# language
  - **Automatic indentation**. Just like all code editor, automatic indentation is a necessity.
  - **Highlight matching parentheses**. Some codes might be complex enough that finding the matching parentheses could be a hassle. So, it is important for this code editor to have this feature.
  - **Code folding**. The fact, that some method on the calculation code could be pretty complex, means that some methods could have many lines. Code folding will be a good feature to let users see the whole structure of the code.
  - **Cut, copy, and paste functionality**. ABB already has a legacy calculation engine system. That legacy system already has several calculation code. In the case of migration by the user, it will be pleasant for them if they could just copy paste their code to the code editor here.
  - **Auto completion**. The auto complete feature should be able to give completion suggestion based on what has already been put on the code and also the available attributes and properties of a parameter that has been selected by the user.
- On the standard non simplified C# mode, the code editor should initially show the default boilerplate code.
- Whenever the user switch the language, the code editor content should change to the code of the active language mode while storing the current code on the previous language mode. This way, that the user can get their old code back if they switch back the language.
- This dashboard should show UI to manage dependency for this particular calculation. The user should be able to add, edit and remove new dependency to this calculation. They should also be able to add, edit and remove any library that can be used as dependency for any calculation.
- This dashboard should have an area where the user can configure the parameter setting for the calculation code. This parameter setting area should be developed as follows:
  - There should be an "add parameter" button to add new parameter to the calculation code.
  - After the user clicks "add parameter" button, it should show a form containing field to add parameter name (which is going to be the parameter name on the code), parameter type select (the options should be taken from CalcParameterType), "is output" and "is array" checkbox.

  - when the user clicks save on that form, it should show a parameter box which shows the used parameter on the code. The code editor should be updated to show that this parameter can now be used on the code. In the non simplified C# mode, the parameter name and its type should appear as the parameter to the main Calculate method.
  - Each parameter box should have an edit button to edit any field of this parameter.
  - Each parameter box should also have a remove button to remove this parameter from this calculation.

- This dashboard should have an area to configure the task, which will use this calculation definition. Below are descriptions of how this area should be developed:
  - The user should be able to add, edit and remove new task for this calculation definition.
  - Each task should have its own configuration panel which will have all the settings to configure the task. This task configuration panel should have three components:
    - Parameter mapping setting area.
      This is the area where the user can configure, which data from the database should be mapped to the parameter, which is being used in the calculation code. Setting for parameter mapping of a task depends on the parameter that has been set by the user previously on the parameter setting area. Whenever the user clicks save on the add new parameter form, a new parameter mapping block should appear here. This parameter mapping block should contain the form for the user to configure the class name, where string, property and also the defaulthistory property. It should also show options whether to use external data source or not. If the user wants to use external data source for a parameter, the box should show a form, which contains connectionstring inputbox. After the user clicks save on the parameter mapping form, it should show the saved parameter mapping box. This mapping should then be able to be referenced from the code editor. Each box should show edit button to edit the preferred mapping.
    - Task setting.
      This component should show a form with the following field for each task: process selector, period length override input box, period length override unit selector, diag update interval second inputbox and max run seconds before killed inputbox. For process field, it should also show a button to manage process. Either to add new process or editing/removing the existing process. If the language mode being used is the non simplified c#, the task setting should show the constructor parameter buttons. When the user clicks this button, there

should be a window showing list of parameter on the code constructor along with inputbox for its value. All the parameter values set here will be stored in JSON format to ConstructorParameter property in CalcTask class.

    * Task trigger setting.
      This component should show a form to configure the task trigger. User should be presented with task trigger type selector, which contains three options: periodical scheduler, event based and triggered by other task. If the user selects periodical scheduler, the user should be presented with scheduler selector, which contains list of available scheduler configurations. Besides this selector, there should be a button to manage the scheduler. Either to add new schedule configuration, or to edit and delete existing schedule configuration. If the user selects event based, the user should be presented with mapping selector, which contains list of existing mapping. This mapping will trigger the execution of this calculation if the value of that mapping changes. If the user selects triggered by other task, the user should be presented with task selector, which contains list of existing tasks. The task trigger setting should also shows the setting for min trigger and max trigger.

- This dashboard should also have a console, which will shows any errors from the calculation engine.

- This dashboard should have a back button, which will take the user back to the calculation list dashboard without saving anything.

- Lastly, this dashboard should have a submit button, which will save all the configurations of this calculation along with its dependency, parameter and task settings. If there are still errors retrieved from calculation engine, all the errors will be shown on the console. If there is no error, then the user will be redirected to the list calculation dashboard including his new calculation.

7.6 Non Functional requirements

7.6.1 Usability

As with the case with all user interface development, usability plays a big role. The user interface has to be user-friendly, intuitive and follows user's work flow. It should be built in a way that could easily be understood right away without the need to read manual guideline.

There should be a way for users to test their calculation configuration. With this test, they could see what would be the result (values of result variables) of the execution of their calculation configuration at the moment. The execution order of the calculations shall be visualized and maintainable to the user so that he/she can easily understand the execution order relations and adjust them.

Despite the requirement that the UI should be intuitive to users, a proper engineering documentation is still needed. The documentation should have step by step instructions for configuring calculations.

7.6.2 Performance

The system should always run smoothly. The auto completion feature on the code editor need to be able to work smoothly even with code with 20000 lines of code and up to 1 million variables. The calculation listing dashboard should have no problem listing thousands of calculations at once.

7.6.3 Integration and Extendability

The UI has to be built on top of the existing ABB front end dashboard system, cpmPlus View. It needs to follow the design architecture of cpmPlus View. Each page should be a cpmPlus view dashboard of its own. Each dashboard should be constructed using widgets especially built for calculation engine web based IDE. In terms of the code, it has to follows ABB standard coding style guide. It must utilizes the internal API of cpmPlus View. All these to make sure the system has good integration with ABB existing system. This will help the system to be more extendable for future development.

7.6.4 Customizability

With cpmPlus View's dashboard-widget architecture. Each widget being built should be customizable. This way the user can customize their own calculation engine IDE according to their own preference.

7.6.5 Diagnostics

By default, the calculation Engine collects diagnostic information on the execution of the calculations such as includes invalidity of the calculation, execution time of the calculation and resource usage of the calculation. The users should be informed clearly and straightforwardly if their calculation is invalid.

Chapter 8

Development and Implementation

This chapter discusses about the development and implementation of the web based IDE. The goal is to address the requirements detailed in the previous section (chapter 7). Before going in deep with the development process of each part, chapter 8.1 will describe the general picture of the process of the development and/or implementation of the web based IDE. The next section (chapter 8.2) will contain explanation of the technology stacks chosen as the basis of the development of the whole system. Chapter 8.3 describes the process of initial prototyping phase. After those three subsections, this chapter will continue on sub chapter 8.4 with detailed development process of the parser/lexer, which constitutes the biggest part of the development of the whole system. After that, chapter 8.5 will continue with the next essential part of the system, the UI itself.

8.1 Overview

There are several steps taken for the purpose of the development of this thesis work:

1. **Initial learning of the cpmPlus view code base.**
   Since the UI needs to be built on top of cpmPlus View, it is necessary to learn the basic usage of cpmPlus view beforehand. And later on, I need to get used to the cpmPlus view SDK so that I would be able to extend its functionality. Usage of cpmPlus view itself is quite easy, the UI is intuitive enough to be learned on its own. Learning the SDK is much more harder. It is because the documentation for it is scarce. I have to dig through the code of the existing system or the existing widget in order to understand what is happening under the hood.

2. **Learning to use the calculation engine.**
   The calculation engine is the back-end, which needs the front-end yet to be built. So, it is essential for me to understand the basic usage of the calculation engine too. Basic usage of calculation engine is pretty straightforward. One day is enough to learn it. But, there are many complex use cases allowed by the calculation engine. These complex use cases need quite some time to

apprehend. One training session with the back-end developer is enough for basic usage of the engine. But for all the complex cases, I have to repeatedly ask the developer for more explanation. Again, documentation for the complex use case does not exist.

3. **Rapid prototyping.**
   Because it takes some time to be familiar with the cpmPlus View and calculation engine architecture, I decided to go with rapid prototyping immediately after I understand the basic usage of cpmPlus View and also the basic use case of calculation engine. This prototyping phase serves the purpose of showing quick proof of concepts of how the UI will be built later on. This prototyping phase also helped me getting used to the cpmPlus View SDK and the calculation engine.

4. **Parser development.**
   In order to build a better user experience for the users who will write their calculation code, it is important to provide a good code editor for them. One of the most important features that is expected from the code editor is a smart auto completion feature. To provide this auto completion feature, the system should be able to read the code and build some form of syntax tree out of it. This is the purpose of the parser.

5. **User interface design.**
   The dashboard system uses a form of widget drag and drop mechanisms. So, for the purpose of developing the UI for the calculation engine web based IDE, I developed some widgets exclusively built for calculation engine.

In addition to the main tasks, there are some subtasks, which will be explained later in respective subchapters.

8.2 Technology Stacks

The UI of the calculation engine needs to be built on top of ABB existing front end dashboard system, cpmPlus View. With that consideration, the choice of technology stack depends heavily on the technology stack of the underlying system.

The core code of cpmPlus View is built with pure Javascript without any use of third party framework. Since it was built around 2011, it still uses the ECMAscript 5.1. So far, it has not utilized any other more modern features of the latest Javascript version (ECMAscript 6). ECMAscript 6 contains several useful features that could potentially increase the efficiency of cpmPlus view. The web based IDE uses several features of ECMAscript 6 such as template literal, extended parameter handling and promises.

One crucial part of this UI system is its ability to connect to the back-end system including the data abstraction layer. The data abstraction layer interacts with the database engine. In the case of calculation engine, the database is stored in ABB's proprietary database engine system called RTDB. RTDB is short for Real

Time DataBase. As the name suggests, this database engine emphasize its real time functionality. The calculation engine will work in real-time with all its data inputs and outputs. This is why it is important for the UI system to be able to connect to the back-end system in real-time manner. For this purpose, cpmPlus view utilizes websocket technology. With websocket, it is possible for cpmPlus view to use publish/subscribe scheme instead of traditional web based request/receive scheme.

Since we are building a web based IDE, code editor will definitely be one of the most important components of the UI. For the code editor itself, cpmPlus View already uses ace code editor for some part of their system. Since ace editor has been integrated into the cpmPlus view, it is advantageous to use it for the calculation engine code editor, too. This way there will be no more additional overhead of integrating any other open source Javascript based code editor.

The whole development was done using Microsoft Visual Studio. It is the standard IDE being used internally by ABB's software engineers. For the versioning system, ABB uses TFS, which is integrated into Microsoft Visual Studio.

8.2.1 Prototyping stacks

For initial prototyping phase, the main consideration factor is to develop it as quick as possible. With that in mind, it was inadvisable to code it directly with cpmPlus View SDK. It is because with cpmPlus View SDK, the long initial learning curve will impede the process of rapid prototyping. Thus, the choice of stack for prototyping purpose depends only on my familiarity with the technology itself. Bootstrap was chosen as the front-end framework. Bootstrap is the most popular front-end framework nowadays. Bootstrap already has quite many solid and ready-to-use UI components. All interaction code was done using jQuery. jQuery also already has a mature API to manipulate DOM and utilizes Ajax based call.

8.3 Prototyping

Before getting into the actual development process, I decided to build a quick prototype first. The reason for that is because it takes time to fully master the cpmPlus View SDK and the calculation engine. The documentation is still scarce and communication with the respective developer was limited and not so smooth. So, instead of taking too much time on learning the code base myself, I decided to just go with quick prototyping as soon as I got a good enough basic understanding of how cpmPlus View and calculation engine work. This prototyping phase also helps me to get used to the cpmPlus View SDK and also the calculation engine.

8.3.1 Approach

The prototype was built outside of the cpmPlus View. It is then integrated into the cpmPlus view using ready-to-use HTML widget in the cpmPlus view. This widget allows for setting the content of the widget from external source by specifying the URL of the external content. When setting the content to be coming from external source, the widget will act as an HTML frame. I set the URL to be the location of my prototype.

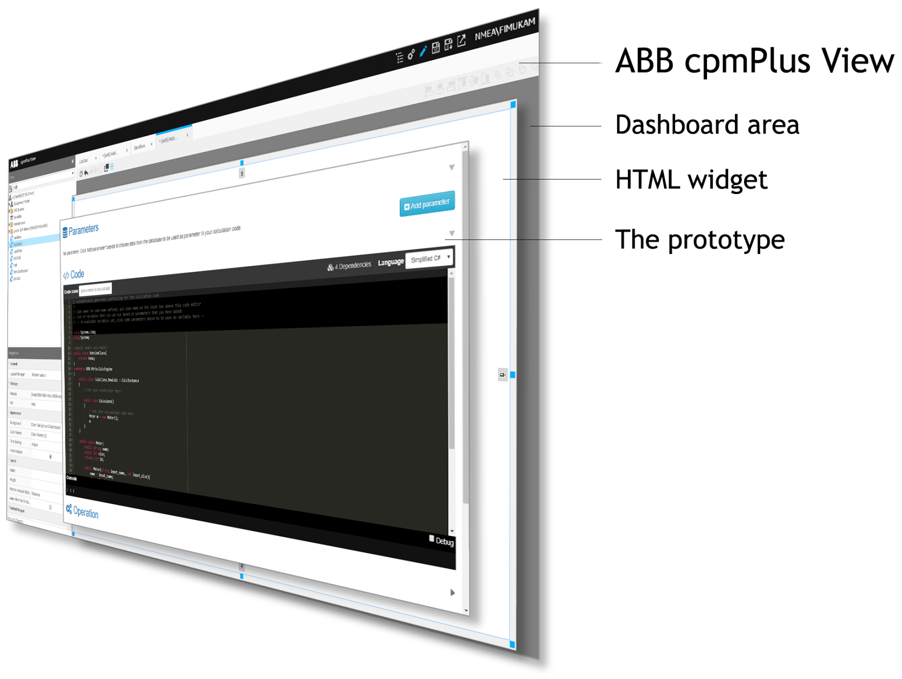


Figure 8.1: Prototype arrangement

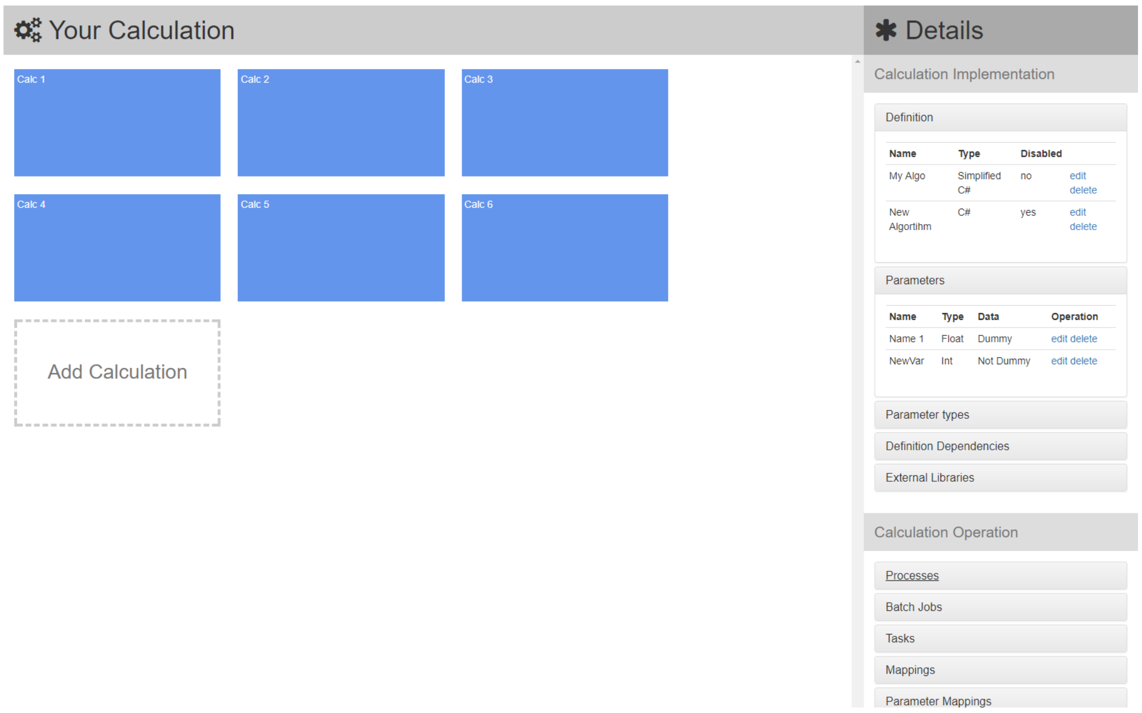
Being an externally and independently developed application from the cpmPlus view, the prototype, of course, still needs a lot of communications back and forth with the cpmPlus view, which will then communicate with the back-end system (calculation engine). For this purpose, I used Javascript postMessage API. This, of course, creates a lot of unnecessary communication overhead. In the real system, this approach will penalize the performance quite hard. But, just for the sake of prototype, it should be fine.

8.3.2 Scope

The main purpose of the prototype is to quickly demonstrate the UI for main end user work flow that have been mentioned in the previous chapter. In this work flow, there is a clear flow that the user can follow to work on a calculation. This work flow comes with the assumption that most users of the calculation engine do not really care about taking care of configurations for so many other aspects that can actually be configured in the calculation engine in a decoupled fashion. They just want to pick some data source that they know, code some calculation on those data and then immediately run the calculation. So with the proposed UI, the user will do all those three things in a single page. This work flow has a downside that it become less decoupled and less flexible than how the back-end system can actually work. The back-end system (the calculation engine itself) is actually a very loosely tangled system to ensure higher flexibility. But, this flexibility comes with more complexity for the end user.

The prototype does not fulfill the whole requirement. Two dashboards that are listed in the requirement were built. They are list calculation dashboard and add/edit calculation dashboard.

List Calculation Dashboard Prototype

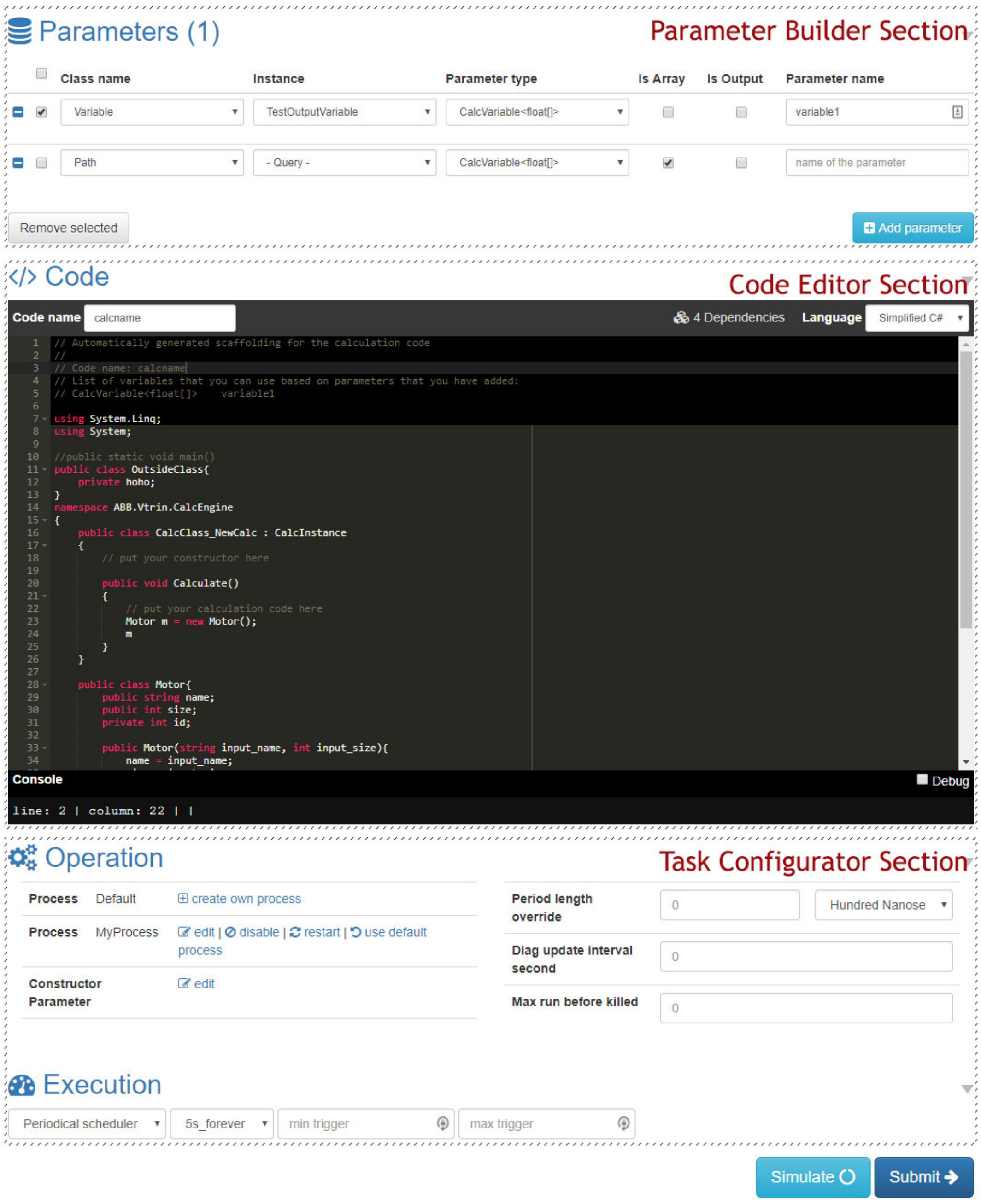


Name	Type	Disabled	
My Algo	Simplified C#	no	edit delete
New Algortihm	C#	yes	edit delete

Name	Type	Data	Operation
Name 1	Float	Dummy	edit delete
NewVar	Int	Not Dummy	edit delete

Figure 8.2: List calculation dashboard prototype

The list calculation dashboard was built without any functionality. This dashboard prototype just shows how would the layout of this dashboard be developed. The prototype consists of two columns. The first column shows list of calculations where each calculation is shown in its own box. There is also a link to add new calculation on this column. The second column shows list of calculation engine classes along with its instances. The first column is built to reflect the user perception on the cohesion of the calculation definition, whereas the second column is built on the basis of how the system is developed in a decoupled fashion, class by class.



Parameters (1) — Parameter Builder Section

	Class name	Instance	Parameter type	Is Array	Is Output	Parameter name
✓	Variable	TestOutputVariable	CalcVariable<float[]>			variable1
	Path	- Query -	CalcVariable<float[]>	✓		name of the parameter

Code — Code Editor Section

Code name: calcname | 4 Dependencies | Language: Simplified C#

```
// Automatically generated scaffolding for the calculation code
//
// Code name: calcname
// List of variables that you can use based on parameters that you have added:
// CalcVariable<float[]>    variable1

using System.Linq;
using System;

//public static void main()
public class OutsideClass{
    private hoho;
}
namespace ABB.Vtrin.CalcEngine
{
    public class CalcClass_NewCalc : CalcInstance
    {
        // put your constructor here

        public void Calculate()
        {
            // put your calculation code here
            Motor m = new Motor();
            m
        }
    }

    public class Motor{
        public string name;
        public int size;
        private int id;

        public Motor(string input_name, int input_size){
            name = input_name;
```

Console

```
line: 2 | column: 22 | |
```

Operation — Task Configurator Section

Process	Default	create own process
Process	MyProcess	edit | disable | restart | use default process
Constructor Parameter		edit

Period length override	0	Hundred Nanose
Diag update interval second	0	
Max run before killed	0	

Execution

Periodical scheduler | 5s_forever | min trigger | max trigger

Figure 8.3: Add calculation dashboard prototype

Add/Edit Calculation Dashboard Prototype

As for the add/edit calculation dashboard, only some features were developed. Those features are parameter builder and the code editor. As for the task configuration, the prototype set everything by default despite it having the UI for the task configuration. So, the UI for task configuration was there just for layout demonstration purpose.

This dashboard prototype was arranged following the presumed user work flow. Parameter builder is put on top as the first thing that the user would do upon adding new calculation. Right below the parameter builder, there is a code editor reflecting the next task on the user work flow after selecting the parameter from the database. At the bottom, there is a section to configure task for execution of the calculation.

Parameter Builder

The parameter builder was built in the form of tabular view. This section is meant to fill the data for CalcParameter, CalcMapping and CalcParameterMapping on the calculation engine class. The table has six columns:

- **Class name**. In this cell, the user would be presented with a selector, which contains list of all classes in the database. This field would be stored on the "Class" property of CalcMapping class. That property itself in the database is actually in the free text form. The choice for a free text data type is to accommodate those who need to use external data source where the list of classes is unknown by the system. In this context, the user needs to write down the name of the class manually. But for the prototype, the feature to use external data source was dismissed. So, it was decided that the class field here will use HTML selector instead of regular input box. The selector would provide better user experience for the user.

- **Instance**. In this cell, the user selects the instance, which he/she wants to use as a parameter in the code. The options of the selector depend on the class selected by the user. It will show list of instances of whatever class the user selects in the previous cell. This field will fill the "WhereString" property on the CalcMapping class. This field has the same issue as the class selector. In the database, this field is in the form of free text. For this field, it is not only to allow flexibility for those who use external data source, but also for a much more flexible way to selecting the target instance. Again, this flexibility is dismissed on the prototype. So, this field is presented as instance selector only.

- **Parameter Type**. This cell will contain parameter type selector. The type will be used as the parameter data type on the code. The options of this selector comes from CalcParameterType. This field will fill the "Type" property of CalcParameter class.

- **Is Array**. This cell contains a checkbox to determine, whether this parameter is an array or not. This field will fill the "ArrayDimensions" property on CalcParameter class. This property is meant to allow users to use parameter with any dimension of array. But for now, the system only allows 0 or 1 option. 0 means this parameter is not an array and 1 means that this parameter is a one dimensional array. Thus, the field that is being used is a checkbox.

- **Is Output**. This cell contains "Is Output" checkbox. This field will fill the "IsOutput" property on CalcParameter class.
- **Parameter Name**. This cell will show an input box for the user to put a name to this parameter. This field will fill the "Name" property on the CalcParameter.

Looking at the presented field above, it is clear that not all properties of those three target classes for this parameter builder section is presented. For example, there is no way for the user to set the DSN property on CalcMapping. There is also no way for the user to fill the "Defaulthistory" property on CalcParameterMapping. All those properties are left blank by default because it is not essential for prototyping purpose.

Code Editor

ABB's cpmPlus View already has a code editor implemented for some parts of the system. They use a third party code editor called Ace Editor. It is a Javascript based code editor. So, it is logical to use the same code editor for this prototyping purpose as well. On top of the code editor section, I put code name field, language selector field and a dependency button. On the bottom of the code editor, I added a console with a "debug" checkbox. Below that, there is a bar, which shows the position of the cursor on the code editor.

Whenever the user types something into the name field. The code editor will be updated to show that this code is using the name. Whenever the user changes the language via the language selector, the content of the code editor would also be switched to the corresponding language. Whenever the user clicks on the dependency button, a pop up will appear showing a sample of how the dependency configurator layout would look like.

The main thing that I wanted to show on this prototype is the auto complete functionality on the code editor. The auto completion feature get the list of auto complete suggestion based on the content of the code and also the selected parameter from the database. This prototype already has basic level of that functionality. The initial parser development (which will be explained deeper in the later chapter) was done in parallel during the prototyping phase. The basic parser has already been able to construct a good enough symbol table to be used on the auto completion system. The code editor is also connected to the parameter builder to provide auto completion on object property from the database.

8.3.3 Feedback

The prototype was done in roughly two weeks. After that, I showed the demo of the prototype to the team at ABB. Below are some of their feedback:

- The list calculation dashboard should not show the list of calculation in a box format. Because box format like that is only suitable for showing images.

- The add calculation dashboard has too much whitespace. There are many areas within the screen, which do not really show anything.
- This whitespace issue, in effect, forces the UI to exceed the viewport. It means that the user will be required to scroll through the page to set up all the configuration for their calculation.
- This scrolling issue is worsened by the fact that the code editor also has scroll bar of its own. This creates a stacked scroll bar, which is a bad design pattern on UX perspective.
- The instance selector might not work well in the case that there are tons of options for the instance. ABB already has UI component to overcome this issue and it was suggested that I use that UI component.

The feedback is then later analyzed whether they are justifiable. I asked for opinion from another developer regarding the feedback to get better aggregate feedback. Most of those feedbacks were eventually taken as a good suggestion for future development of the real system.

8.4 Parser Development

One of the most important components of an IDE is a code editor. The ideal code editor of an IDE should be versatile by offering features like auto completion, syntax checking, debugger integration, etc. In the case of this thesis work, we limit the scope of the expected IDE-like functionality to auto completion only. In order to provide auto completion functionality on the code editor, the system should be able to read the code and build a data structure out of it in the form of syntax tree and/or symbol table. The auto completion features will then utilize this data structure to provide the auto completion. For this purpose, we need to implement a parser in our system.

There are several design decision that needs to be made regarding the development of the parser. First, I need to decide whether to use server side parsing or client side parsing. Second, I have to decide whether to use parser generator or build the parser manually. Lastly, I have to decide what to build on behalf of the parser, whether I should build the abstract syntax tree, the symbol table or both. The following chapter will elucidate every decisions that was made.

8.4.1 Server Side Parsing vs Client Side Parsing

The first issue that needs to be decided prior to the parser development is whether to use server side or client side parsing. I and the team at ABB had a disagreement whether to use server side parsing or client side parsing.

Server side parsing means to use a fully functional third party C# parser, which is already available and mature. This mature third party parser can only work on the server. It means that the process of parsing would need to be run with some communication overhead between client and server. On every code change, the client would resend the changed code to the server to be handled by the parser. The parser would then do its job parsing the code and build a complete abstract syntax tree and symbol table. This will result in a much more versatile tree, which can also be used for syntax checking, grammatical analysis and some level of execution. But, there will be the problem of communication overhead. Resending the code over and over again from the client to the server would be an expensive operation. Especially, if it needs to be sent on every keystroke.

Another options is to do client side parsing. The advantage of doing it on the client side is that it could work faster because there is no communication over head. The downside is that it would not be as versatile as using full blown parser on the server side.

Finally, we decide to do client side parsing with consideration that we will not really need a fully functioning parser. We do not need a full complex functionality of an IDE.

8.4.2 Parser Generator vs Manual Parsing

As has been mentioned in the theoretical chapter about parsing, there are two ways to make a parser. We can generate it using parser generator or we can build it manually.

In the context of this thesis work, there is not so many options on using parser generator. This is because our case is pretty specific and rather rare. What we want is a parser generator that could generate a parser based on the C# grammar. Since we work on Javascript environment, the generator needs to be able to generate the parser in Javascript. There are several Javascript based parser generator out there such as PEG.js, nearley and Jison. They are all capable of generating a Javascript based parser with each of their own parsing algorithm. But the problem is none of them already has ready-to-use C# grammar. It means that I will have to write the grammar definition myself. This could be challenging considering the scope of C# grammar. It would be even more challenging because their documentation on writing a complex grammar is quite limited.

With that consideration, I decided to write the parser manually. The fact that we have limited the scope only to auto completion also becomes a factor in this decision. By writing the parser manually, I can have more flexibility on the scope of the parser that I am going to build. I can pick some areas of the parser development that is truly needed for the purpose of this thesis works.

8.4.3 Abstract Syntax Tree vs Symbol Table

A parser, along with the lexer, can generate both abstract syntax tree and symbol table. Abstract syntax tree will enable highly robust code editor with various functionality such as syntax checking and code validation. Symbol table contains all the objects needed for the auto completion suggestion list. Building and maintaining abstract syntax tree is harder and much more complex than building a symbol table. Since the code editor is based on Javascript, both of these objects will have to be stored as Javascript objects. Maintaining both abstract syntax tree and symbol table by a parser as Javascript objects will take considerable amount of time. Since we have already limited our scope only to auto completion feature, it was decided that the object that the parser will build and maintain is only the symbol table.

8.4.4 Syntactical Analysis Process

Before the syntactical analysis process begins, the parser needs to have tokens to process with. These tokens are generated by the lexer during the lexical analysis process. For this thesis work, the tokenization is handled by the Ace code editor. Ace code editor already has built-in lexer that they use for the purpose of code indentation and syntax highlighting.

I utilized the tokens that are already available from the Ace code editor. The tokens include: identifier (variable names, class names, method names, etc), comments, keywords, punctuation, open and closing brackets, parentheses and operators. In a full parser, all of these tokens will need to be parsed and processed into the abstract syntax tree. But, since I have decided to only build a symbol table, I am only interested in identifier, and keywords. The rest of the token is just used to determine how should the identifier be stored in the symbol table.

Every time the user makes some changes to the code, Ace will redo the tokenization and provides new streams of tokens. Once the tokens are ready, the parser will iterate through all the tokens. On every iteration, the parser will determine how and where should the current token gets stored in the symbol table based on various factors on the grammatical specification.

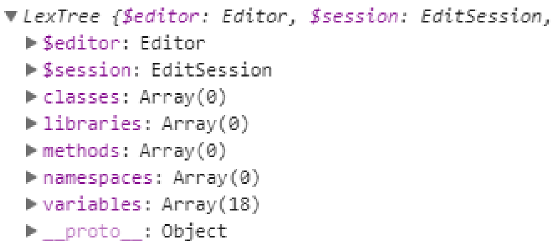
The process of rebuilding the symbol table is repeated every time there is a change in the code. If the user types a character or pastes a snippet of code in the code editor, the lexer will re-read the whole code, do the syntactical analysis process and rebuild the entire symbol table. This rebuilding process also happened every time there is a change on the code which is done programmatically by some other outer process. For example, whenever the user types down the calculation name on the calculation name input box, the system will automatically use that name and augment it to the name of the main class on the code. After that, the parser will do its job and rebuild the symbol table.

8.4.5 Symbol Table Format

The outcome of my parser is a symbol table. This symbol table is stored in a Javascript object. The way that I built the symbol table is a little bit different than the symbol table that is created by most parsers. With the different approach that I took, the symbol table here actually resembles collection of lexicons from the code which is stored in a structured tree along with its detailed information. This is why I call it LexTree.

A symbol table is commonly built as a hierarchical tree. The hierarchy of the tree follows the scoping code block of the input source. The deeper the code block scope, the deeper the symbol table tree goes. For a parser in a compiler, this approach is essential.

As for the specific need of this thesis work, a full hierarchical architecture is not necessary. It is because the usage of symbol table in our case is only to provide auto completion. It is not going to be used for further processing like building of an intermediate representation.



```
▼ LexTree {$editor: Editor, $session: EditSession,
  ▶ $editor: Editor
  ▶ $session: EditSession
  ▶ classes: Array(0)
  ▶ libraries: Array(0)
  ▶ methods: Array(0)
  ▶ namespaces: Array(0)
  ▶ variables: Array(18)
  ▶ __proto__: Object
```

Figure 8.4: LexTree symbol table

In my symbol table, the number of the root branches is fixed. As can be seen from figure 8.4, the root of my LexTree will always have 7 branches. The first two branches ($editor and $session) are there for parsing purposes only. It does not really represent anything from the code. The rest of the branches represent all type of identifiers that are being parsed by the parser.

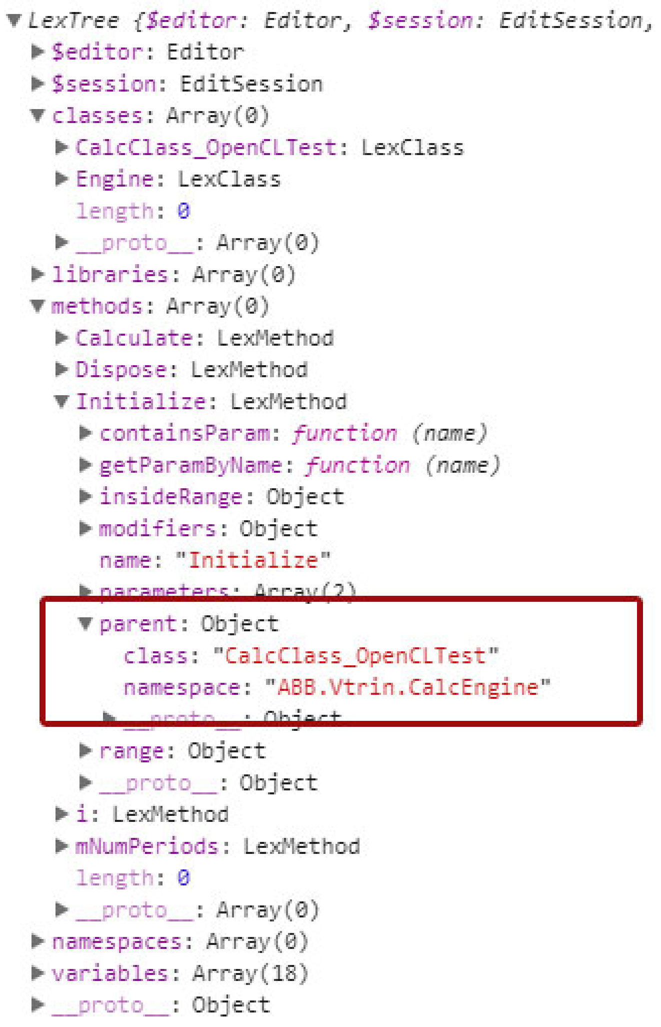
Each branch will contain all the respective identifiers successfully parsed from the code. They are all stored in a non hierarchical way. All parsed identifiers from any block scope will be stored right below its respective root branch. For example, a variable, which is declared inside a method scope, will be put right below the variables root branch along with any variable, which is declared from its class scope. The hierarchical information of each particular identifier is instead stored in "parent" attribute inside their object on the LexTree.

With fully hierarchical tree, the system would need to traverse the whole tree to gather instances for the auto completion list. With my LexTree approach, full tree traversal is not needed. The system just needs to iterate through all respective identifiers, which have the correct parent attribute.

Every identifier on this LexTree has range attribute. This attribute contains the row and column position of that identifier on the code. For identifiers, which define a new block scope such as namespace, class and method, there is also an attribute called insideRange. This is similar to the range attribute, but in insideRange the row and column position that is being stored is the first opening position of the scope and the closing of the scope.This information is used to detect position of the cursor inside the code editor relative to the structure of the code.

8.5 User Interface Design

The bigger picture of this thesis work is to design a UI for ABB's industrial IoT system. This chapter will elaborate the process of designing the UI for ABB's calculation engine.



```
▼ LexTree {$editor: Editor, $session: EditSession,
  ▶ $editor: Editor
  ▶ $session: EditSession
  ▼ classes: Array(0)
    ▶ CalcClass_OpenCLTest: LexClass
    ▶ Engine: LexClass
      length: 0
    ▶ __proto__: Array(0)
  ▶ libraries: Array(0)
  ▼ methods: Array(0)
    ▶ Calculate: LexMethod
    ▶ Dispose: LexMethod
    ▼ Initialize: LexMethod
      ▶ containsParam: function (name)
      ▶ getParamByName: function (name)
      ▶ insideRange: Object
      ▶ modifiers: Object
        name: "Initialize"
      ▶ parameters: Array(2)
      ▼ parent: Object
          class: "CalcClass_OpenCLTest"
          namespace: "ABB.Vtrin.CalcEngine"
        ▶ __proto__: Object
      ▶ range: Object
      ▶ __proto__: Object
    ▶ i: LexMethod
    ▶ mNumPeriods: LexMethod
      length: 0
    ▶ __proto__: Array(0)
  ▶ namespaces: Array(0)
  ▶ variables: Array(18)
  ▶ __proto__: Object
```

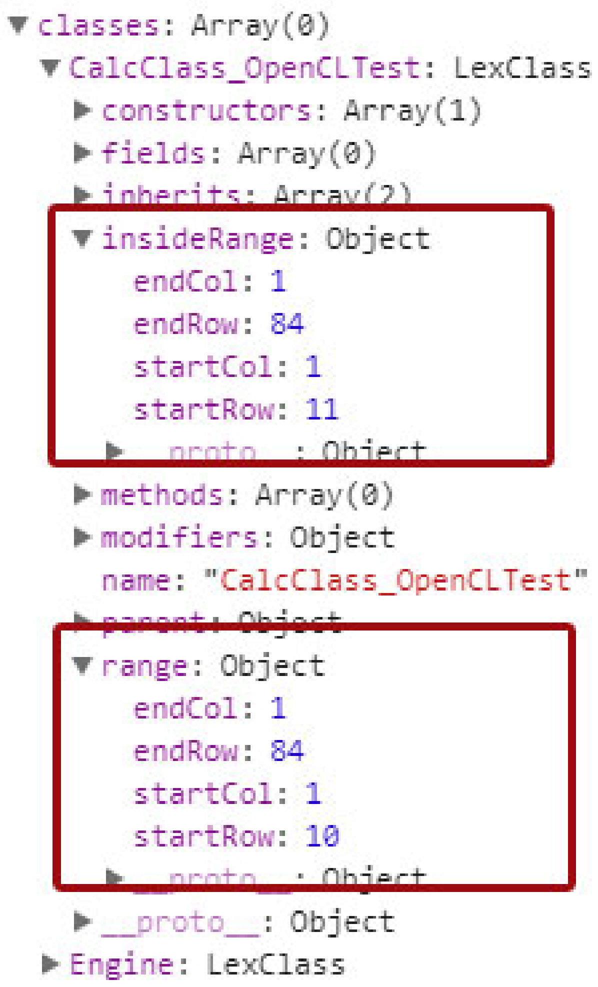
Figure 8.5: Parent attribute on LexTree object

8.5.1 Widget Development

CpmPlus View uses widgets that can be dragged and dropped anywhere within the dashboard. For the purpose of building the UI for the calculation engine, I developed several widgets exclusively. Each of these widgets work together forming a cohesive dashboard for the calculation engine UI.

A widget is coded in a Javascript file. Below is the minimum Javascript code that is needed to build a widget. This widget does nothing. All the widget code must be written inside the widget function.

```
if(!ABB.Mia.Widgets.SampleNamespace) ABB.Mia.Widgets.
    SampleNamespace={};
ABB.Mia.Widgets.SampleNamespace.cSampleEmptyWidget=function(
    parentelement, config)
{
    var pub=ABB.Mia.cWidgetBase(parentelement);
    pub.base.ReadProperties(config);
```



```
▼classes: Array(0)
  ▼CalcClass_OpenCLTest: LexClass
    ▶constructors: Array(1)
    ▶fields: Array(0)
    ▶inherits: Array(2)
    ▼insideRange: Object
        endCol: 1
        endRow: 84
        startCol: 1
        startRow: 11
      ▶__proto__: Object
    ▶methods: Array(0)
    ▶modifiers: Object
     name: "CalcClass_OpenCLTest"
    ▶parent: Object
    ▼range: Object
        endCol: 1
        endRow: 84
        startCol: 1
        startRow: 10
      ▶__proto__: Object
    ▶__proto__: Object
  ▶Engine: LexClass
```

Figure 8.6: Positional attribute on LexTree object

```

    // widget's code must be written here

    return pub;
};
ABB.Mia.Components.Add("ABB.Mia.Widgets.SampleNamespace.
    cSampleEmptyWidget", ABB.Mia.cWidgetBase);
```

Class Inheritance

This piece of code gives a brief notion about ABB's internal Javascript development framework. It shows how the internal framework implements inheritance in Javascript. In this sample code, it is shown in line 4 that the cSampleEmptyWidget has a variable called pub. Every widget must have this variable where it stores all of its public functions/variables. In order to inherit from a class, the pub object is set to an instance of an inherited class. In the case of this sample widget code, it is inheriting the cWidgetBase class. The inherited class stores all of the public functions and variables of that superclass to the child class. Every protected member of that class should be put inside pub.base. ReadProperties on the 5th line is a method that reads the widgets' saved properties that the Dashboard Editor has saved. In order to, for example, restore the widget's position when loading the widget, the widget needs to call ReadProperties with the config it has been given. Finally, the pub object need to be returned from a constructor of this widget (called SampleEmptyWidget in this example).

Widget definition

Before the widget can be used in the dashboard, there is another step that needs to be done. That step is to prepare an mdw file. Mdw file is a file, which informs the cpmPlus View about the widget's information, such as name of the widget, category, path to the icon file, and which javascript and css files to load and from where. Below is the sample of the mdw file:

```
//(widgets/tutorial/1/emptywidget.mdw)
{
   "Name": "Sample Empty Widget",
   "Category": "Sample",
   "IsBrowsable": true,
   "Class": "ABB.Mia.Widgets.Tutorial.cSampleEmptyWidget",
   "Icon": "sampleemptywidget.png",
   "Sources": ["$/sampleemptywidget.js"]
}
```

Widget Properties

A widget can have properties. These properties can be used to make a widget customizable through cpmPlus dashboard editor. To add modifiable properties to a widget, the properties need to be declared in the widget constructor. Below is the sample code on how to add a property to a widget:

```
if(!ABB.Mia.Widgets.SampleNamespace) ABB.Mia.Widgets.
   SampleNamespace={};
ABB.Mia.Widgets.SampleNamespace.cSampleEmptyWidget=function(
   parentelement, config)
{
  var pub=ABB.Mia.cWidgetBase(parentelement);

  pub.base.AddProperty(
    "NewProperty", // name of the property
    ABB.Mia.cPropertyType.Text, // type of the property
    {
      Description: "Color of the widget",
      DefaultValue: "hello",
      Browsable: true,
      Serializable: true
    }
  );

  ...
};
```

Widget Connection to the Database

cpmPlus View has many ways to connect to the database. All connections to the database are done using Websocket technology. The framework already provides several APIs to add, edit and remove any instances on the database. The main entrance to the database manipulation API is through DBConnection object. Below are some of the most important functions on the API to work with the database.

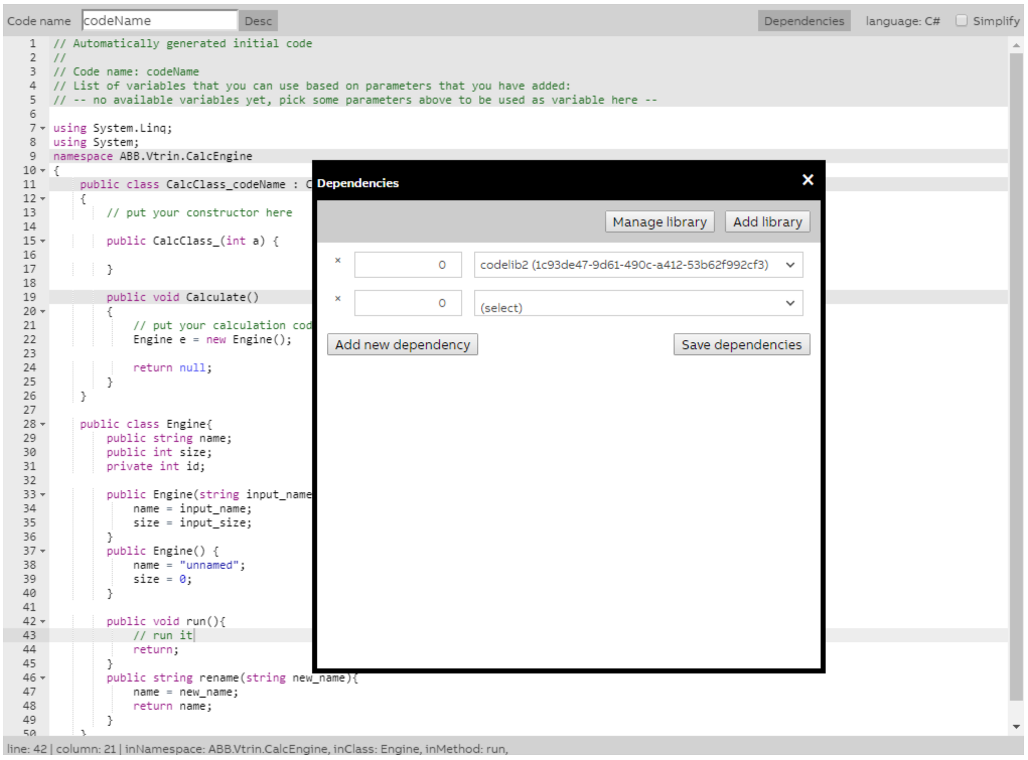
```
// create new DB Connection object
var connection = pub.base.GetDashboard().GetConnection();

// create new DB instance object
var dbInstance = connection.Classes.ClassName.Instances.Add();

dbInstance.WhenReady(function(){

    // setting property value to a DB instance
  dbInstance.base.SetPropertyValue("PropertyName", "
     propertyValue");

    // committing any changes on the dbInstance to the
       database
    dbInstance.commitChanges();

    // executed when the commit process is done
    db.WhenCommitDone(doneCallback);

    // executed when the commit process is failed
    db.WhenCommitFailed(failedCallback);
});

// used to subscribe to any change on the database
connection.SubscribeChanges(subscribedClass, null, null, null,
    callback);
```

8.5.2 Calculation Engine Widgets

For the purpose of designing a UI for calculation engine on cpmPlus view platform, I developed several widgets. These widgets were made exclusively for calculation engine. The widgets that I developed are not meant to be reusable for any other purposes. In total, I have developed six different widgets with their own unique functionality. All widgets work together through some configuration on the dashboard editor.

Here is the list of widgets that has been developed for the calculation engine:

- **Code editor widget**
  This is the widget where the users write their calculation code. Below, is the



Code name: codeName

```
// Automatically generated initial code
//
// Code name: codeName
// List of variables that you can use based on parameters that you have added:
// -- no available variables yet, pick some parameters above to be used as variable here --

using System.Linq;
using System;
namespace ABB.Vtrin.CalcEngine
{
    public class CalcClass_codeName : C
    {
        // put your constructor here

        public CalcClass_(int a) {

        }

        public void Calculate()
        {
            // put your calculation cod
            Engine e = new Engine();

            return null;
        }
    }

    public class Engine{
        public string name;
        public int size;
        private int id;

        public Engine(string input_name
            name = input_name;
            size = input_size;
        }
        public Engine() {
            name = "unnamed";
            size = 0;
        }

        public void run(){
            // run it
            return;
        }
        public string rename(string new_name){
            name = new_name;
            return name;
        }
```

Dependencies: 0 codelib2 (1c93de47-9d61-490c-a412-53b62f992cf3); 0 (select)

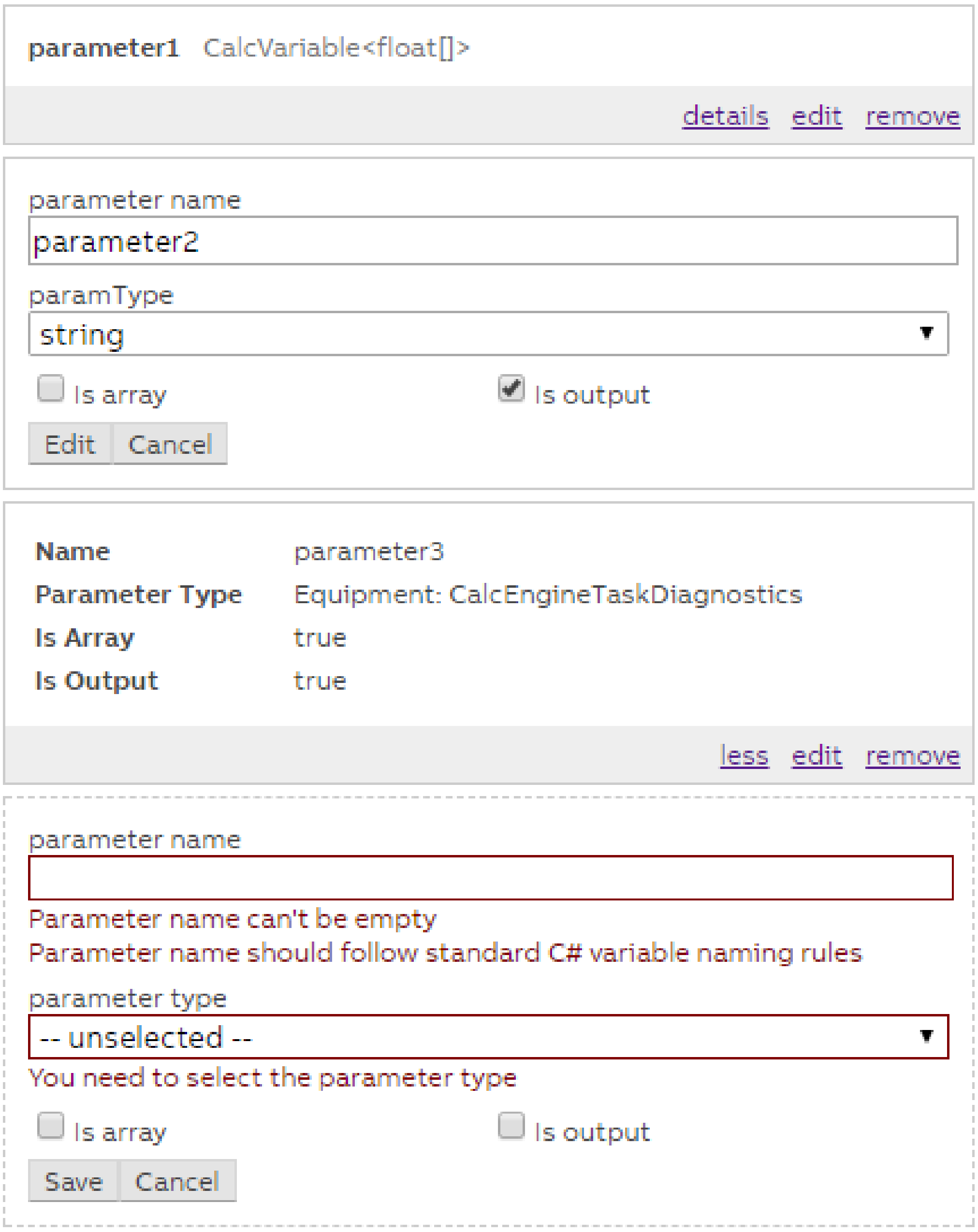
line: 42 | column: 21 | inNamespace: ABB.Vtrin.CalcEngine, inClass: Engine, inMethod: run,

Figure 8.7: Code editor widget screenshot

  list of features that are available on this widget:

  – Set the name, description and type of a calculation definition.
  – Add dependency to the calculation definition.
  – Edit dependencies of the calculation.
  – Remove dependencies from the calculation.
  – Manage (add, edit delete) external library to be used as dependency on a calculation definition.
  – Code editor with syntax highlighting, indentation, cursor detection and integrated auto completion support.

- **Parameter selector widget**
  This widget is created to easily select and configure parameter, which will be used as the parameter to the main function on the calculation code. Below is the list of features that are available on this widget:



parameter1 CalcVariable<float[]>

details edit remove

parameter name
parameter2
paramType
string
☐ Is array ✓ Is output
Edit Cancel

Name	parameter3
Parameter Type	Equipment: CalcEngineTaskDiagnostics
Is Array	true
Is Output	true

less edit remove

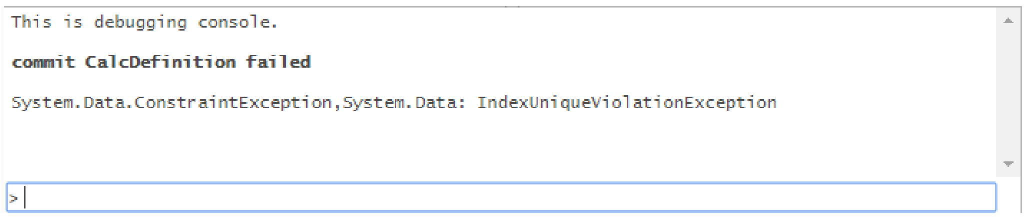
parameter name

Parameter name can't be empty
Parameter name should follow standard C# variable naming rules
parameter type
-- unselected --
You need to select the parameter type
☐ Is array ☐ Is output
Save Cancel

Figure 8.8: Parameter selector widget screenshot

- Add new parameter.
- Edit parameter.
- Remove parameter.
- See details for a parameter block.

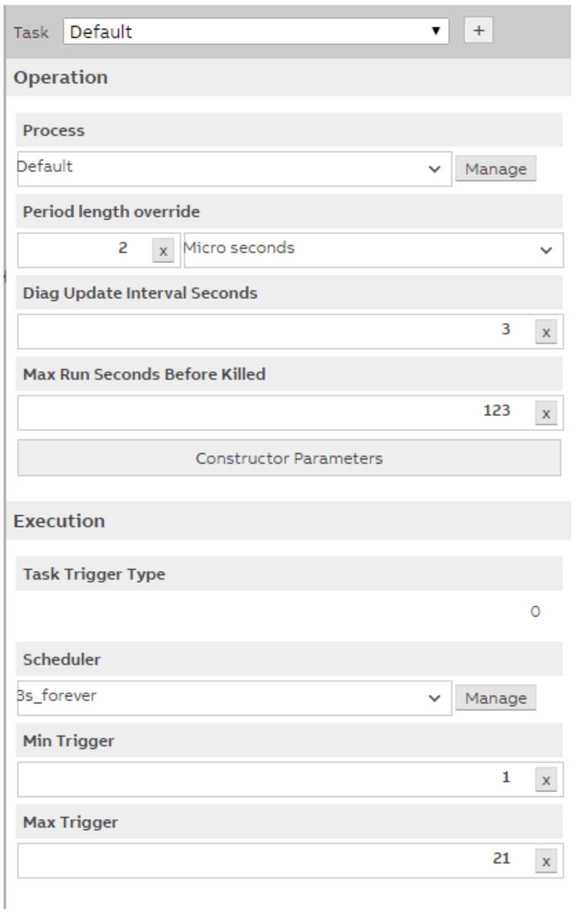
- **Console widget**
  This widget will show any error retrieved from the calculation engine regarding the calculation that is being worked on.



```
This is debugging console.
commit CalcDefinition failed
System.Data.ConstraintException,System.Data: IndexUniqueViolationException
>
```

Figure 8.9: Console widget screenshot

- **Task configurator widget**
  This is where the user can set the configuration of tasks for the calculation.



Task: Default

Operation

Process: Default

Period length override: 2 Micro seconds

Diag Update Interval Seconds: 3

Max Run Seconds Before Killed: 123

Execution

Task Trigger Type: 0

Scheduler: 3s_forever

Min Trigger: 1

Max Trigger: 21

Figure 8.10: Task configurator widget screenshot

Below are list of features that are available on this widget:

- Manage (add, edit, delete) process.

  - Manage (add, edit, delete) task.
  - Manage (add, edit, delete) scheduler.

- **Action bar widget**
  This widget is the "glue" between all the other widgets, which are used in the

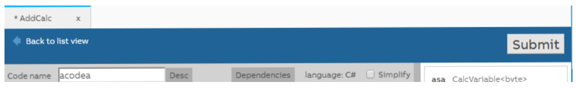


Figure 8.11: Calc action bar widget screenshot

  add calculation dashboard. This widget contains the submit/edit button that will submit new instance of calculation or edit the loaded calculation from the database. If there is no error in the user's calculation code and configuration, this widget will redirect the user to list calculation dashboard.

- **Calc list widget**
  This widget is used for the list calculation dashboard. It lists all the calculation.

8.5.3 Dashboard Arrangement

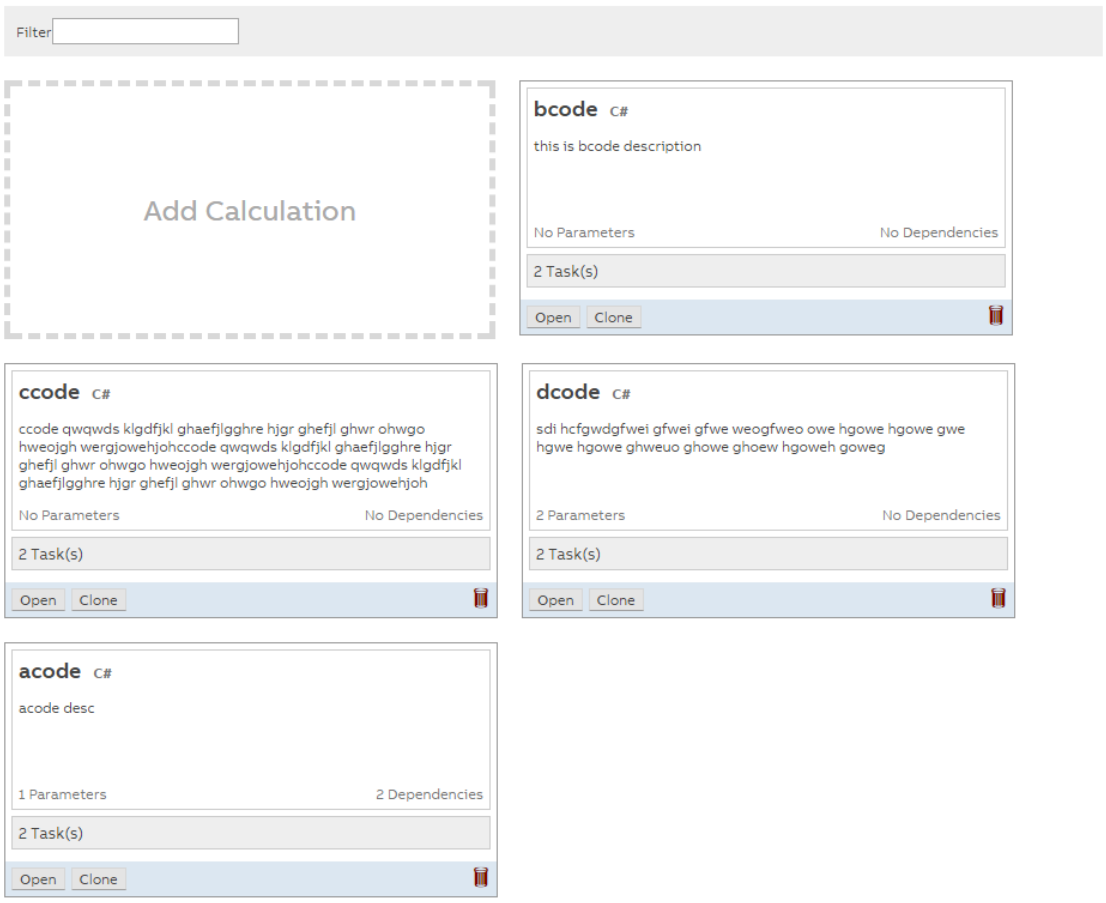
Based on what has been defined in the requirement, the UI is split into two pages: list calculations dashboard and add/edit calculation dashboard.

List Calculations Dashboard

This dashboard consists of only one big widget, which takes the whole area of the dashboard. The name of the widget is CalcList. This widget lists all existing calculations. Whenever the user clicks "open" button on this dashboard, it will open the add/edit calculation dashboard with some context data to notify the AddCalc Dashboard that it should be on edit mode instead of add new mode. And, also to populate all the fields (code editor, param fields and task configurations settings) with the value of the referring calculation.

Add/Edit Calculation Dashboard

This dashboard is used to add a new calculation or edit the existing calculations. Calculation here means the whole package of the calculation that includes the code definition, parameters setting, and task settings. This dashboard contains four widgets: code editor, console, parameter selector, configurator and action bar.


Filter

Add Calculation

bcode C#

this is bcode description

No Parameters No Dependencies

2 Task(s)

Open Clone

ccode C#

ccode qwqwds klgdfjkl ghaefjlgghre hjgr ghefjl ghwr ohwgo hweojgh wergjowehjohccode qwqwds klgdfjkl ghaefjlgghre hjgr ghefjl ghwr ohwgo hweojgh wergjowehjohccode qwqwds klgdfjkl ghaefjlgghre hjgr ghefjl ghwr ohwgo hweojgh wergjowehjoh

No Parameters No Dependencies

2 Task(s)

Open Clone

dcode C#

sdi hcfgwdgfwei gfwei gfwe weogfweo owe hgowe hgowe gwe hgwe hgowe ghweuo ghowe ghoew hgoweh goweg

2 Parameters No Dependencies

2 Task(s)

Open Clone

acode C#

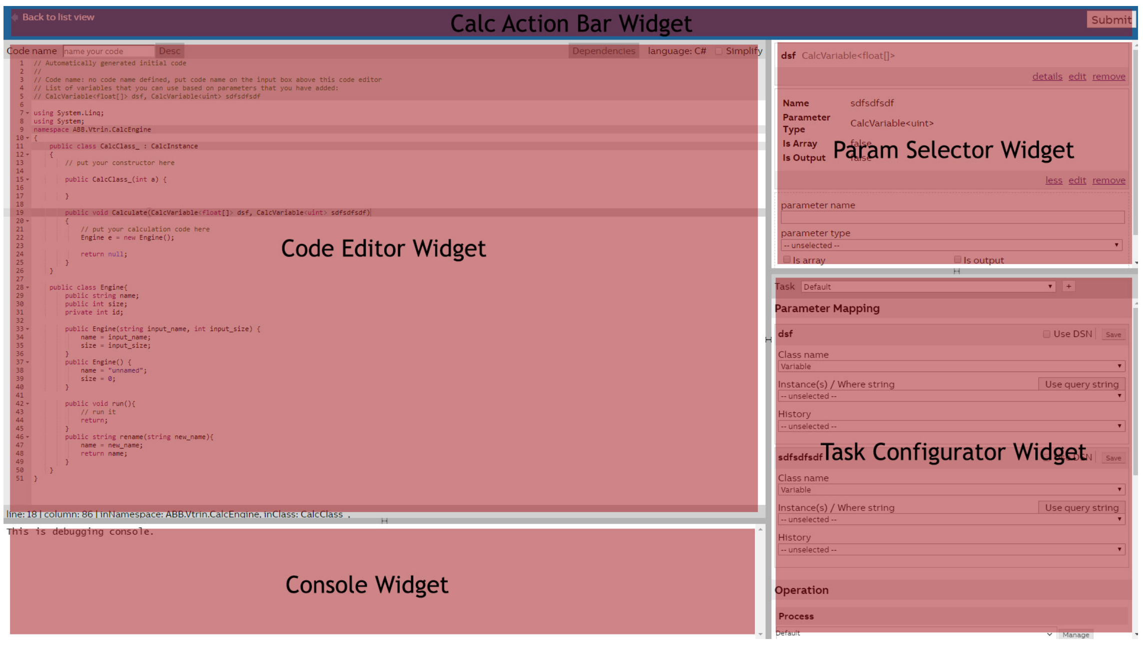
acode desc

1 Parameters 2 Dependencies

2 Task(s)

Open Clone

Figure 8.12: List calculations dashboard screenshot



```
// Automatically generated initial code
//
// Code name: no code name defined, put code name on the input box above this code editor
// List of variables that you can use based on parameters that you have added:
// CalcVariable<float[]> dsf, CalcVariable<uint> sdfsdfsdf

using System.Linq;
using System;
namespace ABB.Vtrin.CalcEngine
{
    public class CalcClass_ : CalcInstance
    {
        // put your constructor here

        public CalcClass_(int a) {

        }

        public void Calculate(CalcVariable<float[]> dsf, CalcVariable<uint> sdfsdfsdf)
        {
            // put your calculation code here
            Engine e = new Engine();

            return null;
        }
    }

    public class Engine{
        public string name;
        public int size;
        private int id;

        public Engine(string input_name, int input_size) {
            name = input_name;
            size = input_size;
        }
        public Engine() {
            name = "unnamed";
            size = 0;
        }

        public void run(){
            // run it
            return;
        }
        public string rename(string new_name){
            name = new_name;
            return name;
        }
    }
}
```

```
line: 18 | column: 86 | inNamespace: ABB.Vtrin.CalcEngine, inClass: CalcClass .
```

```
This is debugging console.
```

dsf CalcVariable<float[]>

Name	sdfsdfsdf
Parameter Type	CalcVariable<uint>
Is Array	false
Is Output	false

dsf

sdfsdfsdf

Figure 8.13: Add/edit calculation dashboard screenshot

Chapter 9

Discussion

This chapter contains evaluation of the thesis vis-à-vis the initial goal and the research questions that have been defined (chapter 9.1). Lastly, this chapter will discuss about several improvement possibilities for the future work (chapter 9.2).

9.1 Evaluation

This section evaluates the thesis work with regard to the requirement of the system and the research question that has been defined.

9.1.1 Meeting the Requirements

The requirements for this thesis work have been defined in chapter 7. The requirements are split in two categories: functional requirements and non-functional requirements. All the functional requirements elaborated in chapter 7.5 have been implemented. The process of the implementation has been explained in chapter 8. As for the non-functional requirement, more comprehensive testing still needs to be done in order to check whether the developed system has met the non-functional requirements or not. For example, non-functional requirement about usability would mean that a usability testing is applied for the UI. Performance requirement might need some thorough stress testings.

In general, the system that has been developed in this thesis work is complete. The users have been able to execute most important tasks of the calculation engine through the newly designed UI.

9.1.2 Answering Research Questions

There are two research questions that have been defined on this thesis work. The first question is "What are the requirements of the web based IDE for ABB Ability Calculation Engine?". Extensive requirement analysis has been done to answer this question. The result of this requirement analysis has been written down in chapter 7. The second question is "How should the web based IDE be designed and developed?". The whole system has now been developed and implemented. The detailed process of the development and implementation is elaborated in chapter 8.

9.2 Future work

This thesis work is not perfect. It still has some bugs and caveats. It also has countless opportunities for improvement. This section outlines potential improvements that could be done related to this thesis work. Some of these improvements are specific for the development of calculation engine UI. Some others are more general recommendations on the framework behind the UI.

9.2.1 Parser Performance Optimization

Initially the parser was built without much attention to performance. The reason for that is because it comes with the assumptions that most of the code written by the user would probably be just simple code, so there is no need to build a parser that will be able to work flawlessly with code, which has thousands of lines of code.

But later, it has come to realization that there are some edge cases where the code could reaches tens of thousands of lines of code. Basic testings were done to check how reasonable the current implementation of the parser in handling large code with thousands and tens of thousand of lines of code. The parser began to feel slow and unresponsive when the code has more than 3000 lines of code.

This problem presents opportunity for performance optimization. There are many things that could be done to increase the performance of the parser such as:

- **Diff-ing algorithm implementation**. With implementation of diff-ing algorithm, the system would be able to determine the changes that have been made to the code. This could then be used to increase the performance of the parser. The parser can parse only the change of the code instead of running through the whole code for every changes made by the user.

- **Conditional real-time parsing**. Another strategy that could potentially increase the perceived performance of the parser is to implement conditional parsing. Real time parsing could be done normally when the number of lines on the code is still below some threshold level. When it has exceed the threshold, the system could use different strategy on parsing. For example, the parser could do periodical parsing (i.e., re-parse every 60 seconds) instead of event triggered parsing (i.e., re-parse on every keystroke event).

- **Web worker utilization**. HTML5 comes with a powerful new feature called web worker. This feature lets a web page to run scripts on the background. Clever implementation of web worker could open the opportunity for multi-threading programming. Multi-threading parsing would potentially increase the parser performance dramatically.

- **Context sensitive parsing**. The way the current parser works is without looking at the context. It just re-reads the whole code and rebuilds the symbol table from the beginning. The system already has a way to determine the position of the cursor relative to the code. It could detect if the user adds new code in a particular block scope. The parser could utilize this information by re-parsing only that particular block scope since all the other block scope would not be affected.

9.2.2 More Advanced Code Editor Features

Currently, the available code editor does not have advanced features that require it to understand the code. The most advanced feature of the code editor is only the auto completion. The rest of the features are just basic features of code editor, which could be derived from the lexer such as syntax highlighting, code folding, auto-indentation and so on.

With a full parser that builds not only the symbol table, but also the abstract syntax tree, more advanced features could be developed such as:

- Error checking. With error checking feature, the code editor could tell the user, which part of the code generated the errors.
- Syntax validation. A full parser would immediately be able to tell if the syntax written by the user is not valid since it won't match the provided grammar.

9.2.3 More Advanced IDE Features

Not only can we improve the features of the code editor per se, we can also add more IDE-related features such as:

- Code versioning
- Live debugger
- Refactoring facilities
- Better navigation to and from references/declarations

9.2.4 Drag-and-drop Oriented User Interface

This idea has been surfaced several times during the development of this thesis work by some engineers at ABB. With this drag-and-drop UI, the users no longer need to write their own calculation code. Instead, they can just drag and drop some ready made calculations to the "canvas". On this canvas they can drop some data block and calculation code block. They can then wire every block according to their needs.

9.2.5 Framework Modernization

ABB cpmPlus View currently still uses ECMAscript 5. There is nothing wrong with it. But sticking with the old version of Javascript would mean that we are missing the potential for a better and more efficient way to extend the system. The latest version of Javascript offers array of new features and functionalities that could dramatically help developers code faster, more effective and more efficient. The resulting code could also be easier to read for other developers to extend.

Not all browsers have supported all the latest features of Javascript. This could become a compatibility issue. To overcome this problem, the system could implement transpiler software like Babel. This transpiler would trans-compile any code written in the latest version of Javascript into the earlier version of Javascript that is already well-supported by most browsers. With this approach, we can exploit all the new features that are offered by the language without sacrificing browsers compatibility.

9.2.6 Web Component Implementation

cpmPlus View works with widget system. Each widget has its own functionality that could be dragged and dropped anywhere on the dashboard. Each widget is expected to work independently with standardized way to communicate with other widgets. There is sometimes a problem where styling from some widgets clashed with other widgets. This problem could be solved with web components.

Web components are a set of new HTML features that encourage encapsulation and interoperability of HTML elements. This is exactly what is expected from cpmPlus view widget system. In the future, every widget can be developed as a single new web component. The encapsulation features of web components will alleviate the styling clashes problem.

There is also a possibility to import third party web component to be used as a new widget on cpmPlus View. ABB could also publish its own web components to be used by another parties. This will provide much bigger options of widgets for the user to build their own dashboard without the need to hand code every single widget for them.

Chapter 10

Conclusions

The main goal of this thesis work is to design a good web based UI for ABB's calculation engine as part of their IIoT platform. The UI is expected to have some level of IDE-like functionalities. The approach that is taken on this thesis work is by building such system with constructive research methodology.

The end result of this thesis is a cohesive UI that reflects the user's work flow. The approach helps alleviating the complexities that exist on the underlying system. The finalized design allows end users at ABB to efficiently utilize the calculation engine. The end users can operate the calculation engine in a more natural way according to their task regarding the usage of calculation engine.

This thesis also has become a proofing point of ABB's internal front-end dashboard system. It proves that such a complex UI system could be built on top of this platform.

Opportunities for future improvement of the UI are endless. The end result of this thesis work serves as the initial footstep towards an even more robust UI that could approximate the features of traditional desktop based IDE. Future versions of the UI could incorporate a code editor with full IDE functionality such as syntax checking, validation, live debugging and so on.

Once deployed, the UI along with its back-end calculation engine could help boosting the efficiency of many ABB's engineers working directly with real world problem that could be solved by utilizing the calculation engine.

Bibliography

[1] *ABB Ability.* URL: http://new.abb.com/abb-ability (visited on 08/07/2017).

[2] *About Us | Industrial Internet Consortium.* URL: http://www.iiconsortium.org/about-us.htm.

[3] *ADVANCED MANUFACTURING: A Snapshot of Priority Technology Areas Across the Federal Government.* 2016. URL: https://www.whitehouse.gov/sites/whitehouse.gov/files/images/Blog/NSTC%20SAM%20technology%20areas%20snapshot.pdf.

[4] Alfred V Aho, Ravi Sethi, and Jeffrey D Ullman. *Compilers: principles, techniques, and tools.* Vol. 2. Addison-wesley Reading, 2007.

[5] Alfred V. Aho and Jeffrey D. Ullman. *The Theory of Parsing, Translation, and Compiling.* Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1972. ISBN: 0-13-914556-7.

[6] M. Alam, R. H. Nielsen, and N. R. Prasad. “The evolution of M2M into IoT”. In: *2013 First International Black Sea Conference on Communications and Networking (BlackSeaCom).* July 2013, pp. 112–115. DOI: 10.1109/BlackSeaCom.2013.6623392.

[7] Kevin Ashton. *That ’Internet of Things’ Thing.* URL: http://www.rfidjournal.com/articles/view?4986.

[8] Debasis Bandyopadhyay and Jaydip Sen. “Internet of Things: Applications and Challenges in Technology and Standardization”. In: *Wireless Personal Communications* 58.1 (2011), pp. 49–69. ISSN: 1572-834X. DOI: 10.1007/s11277-011-0288-5. URL: http://dx.doi.org/10.1007/s11277-011-0288-5.

[9] L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice.* SEI Series in Software Engineering. Pearson Education, 2012. ISBN: 9780132942782. URL: https://books.google.com/books?id=-II73rBDXCYC.

[10] BMBF-Internetredaktion. *Zukunftsprojekt Industrie 4.0 - BMBF.* URL: https://www.bmbf.de/de/zukunftsprojekt-industrie-4-0-848.html.

[11] Ann Bosche et al. *How Providers Can Succeed in the Internet of Things.* Tech. rep. Bain and Company Inc, 2016. URL: http://www.bain.com/Images/BAIN_BRIEF_How_Providers_Can_Succeed_In_the_IoT.pdf.

[12] Patrik Cerwall et al. *Ericsson Mobility Report.* Tech. rep. Ericsson, June 2017.

[13] Ian Chiswell and Wilfrid Hodges. *Mathematical logic*. Vol. 3. OUP Oxford, 2007.

[14] Michael Chui, Markus Löffler, and Roger Roberts. "The internet of things". In: *McKinsey Quarterly* 2.2010 (2010), pp. 1–9.

[15] Cisco. *Leading Tools Manufacturer Transforms Operations with IoT*. 2014. URL: `http://www.cisco.com/c/dam/en_us/solutions/industries/docs/manufacturing/c36-732293-00-stanley-cs.pdf`.

[16] US National Intelligence Council. *Disruptive Civil Technologies: Six Technologies With Potential Impacts on US Interests Out to 2025*. Tech. rep. US National Intelligence Council, 2008.

[17] Gordana Dodig Crnkovic. "Constructive Research and Info-computational Knowledge Generation". In: *Model-Based Reasoning in Science and Technology: Abduction, Logic, and Computational Discovery*. Ed. by Lorenzo Magnani, Walter Carnielli, and Claudio Pizzi. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 359–380. ISBN: 978-3-642-15223-8. DOI: `10.1007/978-3-642-15223-8_20`. URL: `http://dx.doi.org/10.1007/978-3-642-15223-8_20`.

[18] Paul Daugherty et al. "Driving unconventional growth through the Industrial Internet of Things". In: *White Paper, Accenture* (2015).

[19] Dcoetzee. *Abstract syntax tree for Euclidean algorithm*. [Online; accessed July 25, 2017]. 2011. URL: `https://commons.wikimedia.org/wiki/File:Abstract_syntax_tree_for_Euclidean_algorithm.svg`.

[20] Grune Dick and H Ceriel. *Parsing techniques, a practical guide*. Tech. rep. Technical Report, Tech. Rep, 1990.

[21] R. Drath and A. Horch. "Industrie 4.0: Hit or Hype? [Industry Forum]". In: *IEEE Industrial Electronics Magazine* 8.2 (June 2014), pp. 56–58. ISSN: 1932-4529. DOI: `10.1109/MIE.2014.2312079`.

[22] J.S. Dumas and J. Redish. *A Practical Guide to Usability Testing*. Human/-computer interaction. Intellect, 1999. ISBN: 9781841500201. URL: `https://books.google.fi/books?id=4lge5k%5C_F9EwC`.

[23] ECMA International. *Standard ECMA-262 - ECMAScript Language Specification*. 5.1. June 2011. URL: `http://www.ecma-international.org/publications/standards/Ecma-262.htm`.

[24] Thomas Erl. *Service-oriented architecture: a field guide to integrating XML and web services*. Prentice hall, 2004.

[25] Thomas Erl. *Service-oriented architecture: concepts, technology, and design*. Pearson Education India, 2005.

[26] Elika Etemad. *CSS Namespaces Module Level 3*. W3C Recommendation. W3C, Mar. 2014. URL: `http://www.w3.org/TR/2014/REC-css-namespaces-3-20140320/`.

[27] I. Fette and A. Melnikov. *The WebSocket Protocol.* RFC 6455. `http://www.rfc-editor.org/rfc/rfc6455.txt`. RFC Editor, Dec. 2011. URL: `http://www.rfc-editor.org/rfc/rfc6455.txt`.

[28] World Economy Forum. *Industrial Internet of Things: Unleashing the Potential of Connected Products and Services.* Tech. rep. World Economy Forum, Jan. 2015.

[29] Roger Fournier. *A Methodology for Client/Server and Web Application Development.* Upper Saddle River, NJ, USA: Yourdon Press, 1999. ISBN: 0-13-598426-2.

[30] Wilbert O Galitz. *The essential guide to user interface design: an introduction to GUI design principles and techniques.* John Wiley & Sons, 2007.

[31] Jesse James Garrett. *"Ajax: A New Approach to Web Applications".* [Archived from the original on 2 July 2008. Retrieved 19 June 2008.] Feb. 2005. URL: `https://web.archive.org/web/20080702075113/http://www.adaptivepath.com/ideas/essays/archives/000385.php`.

[32] *Gartner Says 8.4 Billion Connected "Things" Will Be in Use in 2017, Up 31 Percent From 2016.* Feb. 2017. URL: `http://www.gartner.com/newsroom/id/3598917`.

[33] J Gartner. *Gartner's 2016 Hype Cycle for Emerging Technologies Identifies Three Key Trends That Organizations Must Track to Gain Competitive Advantage.* 2016.

[34] Arthur Gill et al. "Introduction to the theory of finite-state machines". In: (1962).

[35] Richard Harper. *Inside the smart home.* Springer Science & Business Media, 2006.

[36] Robert Harper. *Practical foundations for programming languages.* Cambridge University Press, 2016.

[37] J. Barlow Herget. *Abby to the rescue.* Tech. rep. ABB, 2004. URL: `https://www02.abb.com/global/seitp/seitp202.nsf/0/276d1cba54cbb71dc125723700470e55/$file/Itech.pdf`.

[38] M. Hermann, T. Pentek, and B. Otto. "Design Principles for Industrie 4.0 Scenarios". In: *2016 49th Hawaii International Conference on System Sciences (HICSS).* Jan. 2016, pp. 3928–3937. DOI: `10.1109/HICSS.2016.488`.

[39] Thomas T. Hewett et al. *ACM SIGCHI Curricula for Human-Computer Interaction.* Tech. rep. New York, NY, USA, 1992.

[40] Ian Hickson. *The WebSocket API.* Candidate Recommendation. W3C, Sept. 2012. URL: `http://www.w3.org/TR/2012/CR-websockets-20120920/`.

[41] Ian Hickson et al. *HTML5.* W3C Recommendation. W3C, Oct. 2014. URL: `http://www.w3.org/TR/2014/REC-html5-20141028/`.

[42] Jan Holler et al. *From Machine-to-machine to the Internet of Things: Introduction to a New Age of Intelligence.* Academic Press, 2014.

[43] *Industrial Internet Reference Architecture Technical Report.* Tech. rep. Industrial Internet Consortium, June 2015. URL: `http://www.iiconsortium.org/IIRA-1-7-ajs.pdf`.

[44] *"Industrie 4.0: Mit dem Internet der Dinge auf dem Weg zur 4. industriellen Revolution".* URL: `http://www.vdi-nachrichten.com/Technik-Gesellschaft/Industrie-40-Mit-Internet-Dinge-Weg-4-industriellen-Revolution`.

[45] *Ergonomic requirements for office work with visual display terminals (VDTs) – Part 1: General introduction.* Standard. International Organization for Standardization, May 1997.

[46] Robin Jeffries et al. "User Interface Evaluation in the Real World: A Comparison of Four Techniques". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems.* CHI '91. New Orleans, Louisiana, USA: ACM, 1991, pp. 119–124. ISBN: 0-89791-383-3. DOI: `10.1145/108844.108862`. URL: `http://doi.acm.org/10.1145/108844.108862`.

[47] J. Jin et al. "An Information Framework for Creating a Smart City Through Internet of Things". In: *IEEE Internet of Things Journal* 1.2 (Apr. 2014), pp. 112–121. ISSN: 2327-4662. DOI: `10.1109/JIOT.2013.2296516`.

[48] E. Kasanen, K. Lukka, and A. Siitonen. "The Constructive Approach in Management Accounting Research". In: *Journal of Management Accounting Research* 5 (1993), pp. 241–264. URL: `http://web.ebscohost.com/ehost/detail?vid=4%5C&%5C#38;hid=115%5C&%5C#38;sid=53bed057-a011-4f88-a1ed-129172160157%5C%40sessionmgr102`.

[49] J. Kaur and K. Kaur. "Availing Internet of Things in Industrial decision making: A survey". In: *2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT).* Mar. 2016, pp. 2164–2168. DOI: `10.1109/ICEEOT.2016.7755075`.

[50] B. Kitchenham and S Charters. *Guidelines for performing Systematic Literature Reviews in Software Engineering.* 2007.

[51] Barbara Kitchenham. *© Kitchenham, 2004 Procedures for Performing Systematic Reviews.* 2004.

[52] Hermann Kopetz. *Real-time systems: design principles for distributed embedded applications.* Springer Science & Business Media, 2011.

[53] G. Kortuem et al. "Smart objects as building blocks for the Internet of things". In: *IEEE Internet Computing* 14.1 (Jan. 2010), pp. 44–51. ISSN: 1089-7801. DOI: `10.1109/MIC.2009.143`.

[54] Soren Lauesen. *User interface design: a software engineering perspective.* Pearson Education, 2005.

[55] Kent D Lee. *Programming languages: An active learning approach.* Springer Science & Business Media, 2008.

[56] Shi-Wan Lin et al. *Industrial Internet Vocabulary.* Tech. rep. Industrial Internet Consortium, July 2015. URL: `http://www.iiconsortium.org/pdf/Industrial-Internet-Vocabulary.pdf`.

[57] Sam Lucero. *IoT platforms: enabling the Internet of Things.* Tech. rep. IHS Markit, Mar. 2016. URL: `https://cdn.ihs.com/www/pdf/enabling-IOT.pdf`.

[58] Kari Lukka. "The key issues of applying the constructive approach to field research". In: *Reponen, T.(ed.)* (2000), pp. 113–28.

[59] Denise Lund et al. *Worldwide and Regional Internet of Things (IoT) 2014–2020 Forecast: A Virtuous Circle of Proven Value and Demand.* Tech. rep. IDC, May 2014.

[60] Scott MacDonald and Whitney Rockley. *ABB Ability.* URL: `http://new.abb.com/abb-ability`.

[61] Scott MacDonald and Whitney Rockley. *The Industrial Internet Of Things IIoTReport.* Tech. rep. McRock Capital, 2014. URL: `http://www.mcrockcapital.com/uploads/1/0/9/6/10961847/mcrock_industrial_internet_of_things_report_2014.pdf`.

[62] *Made In Chine 2025.* URL: `http://english.gov.cn/2016special/madeinchina2025/`.

[63] James Manyika et al. *"THE INTERNET OF THINGS: MAPPING THE VALUE BEYOND THE HYPE".* Tech. rep. "McKinsey and Company", June 2015. URL: `http://www.mckinsey.com/~/media/McKinsey/Business%20Functions/McKinsey%20Digital/Our%20Insights/The%20Internet%20of%20Things%20The%20value%20of%20digitizing%20the%20physical%20world/Unlocking_the_potential_of_the_Internet_of_Things_Executive_summary.ashx`.

[64] John C Martin. *Introduction to Languages and the Theory of Computation.* Vol. 4. McGraw-Hill NY, 1991.

[65] J. Nielsen. "Iterative user-interface design". In: *Computer* 26.11 (Nov. 1993), pp. 32–41. ISSN: 0018-9162. DOI: `10.1109/2.241424`.

[66] Jakob Nielsen and Rolf Molich. "Heuristic Evaluation of User Interfaces". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems.* CHI '90. Seattle, Washington, USA: ACM, 1990, pp. 249–256. ISBN: 0-201-50932-6. DOI: `10.1145/97243.97281`. URL: `http://doi.acm.org/10.1145/97243.97281`.

[67] *Overview of the Internet of things.* Global Standards Initiative on Internet of Things, June 2012. URL: `http://handle.itu.int/11.1002/1000/11559`.

[68] Joseph D Patton. *Maintainability and maintenance management.* Vol. 350. Instrument Society of America Research Triangle Park, NC, 1980.

[69] “Recommendations for implementing the strategic initiative INDUSTRIE 4.0”. In: (Apr. 2013). URL: `http://www.acatech.de/fileadmin/user_upload/Baumstruktur_nach_Website/Acatech/root/de/Material_fuer_Sonderseiten/Industrie_4.0/Final_report__Industrie_4.0_accessible.pdf`.

[70] Jeremy Rifkin. *The zero marginal cost society: The internet of things, the collaborative commons, and the eclipse of capitalism.* Palgrave Macmillan, 2014.

[71] F. J. Riggins and S. F. Wamba. “Research Directions on the Adoption, Usage, and Impact of the Internet of Things through the Use of Big Data Analytics”. In: *2015 48th Hawaii International Conference on System Sciences*. Jan. 2015, pp. 1531–1540. DOI: `10.1109/HICSS.2015.186`.

[72] "". *Sass Documentation.* [Online; accessed August 31, 2017]. URL: `http://sass-lang.com/documentation/file.SASS_REFERENCE.html`.

[73] Sven Schrecker et al. *Industrial Internet Security Framework.* Tech. rep. Industrial Internet Consortium, Sept. 2016. URL: `http://www.iiconsortium.org/pdf/IIC_PUB_G4_V1.00_PB-3.pdf`.

[74] Bill Scott and Theresa Neil. *Designing web interfaces: Principles and patterns for rich interactions.* " O’Reilly Media, Inc.", 2009.

[75] *Singapore Smart Nation.* URL: `https://www.smartnation.sg`.

[76] GP Sullivan et al. “Operations & Maintenance Best Practices”. In: *A guide to achieving operational efficiency, Release* 2 (2004).

[77] Eelco Visser et al. *Scannerless generalized-LR parsing.* Universiteit van Amsterdam. Programming Research Group, 1997.

[78] Cathleen Wharton et al. “Usability Inspection Methods”. In: ed. by Jakob Nielsen and Robert L. Mack. New York, NY, USA: John Wiley & Sons, Inc., 1994. Chap. The Cognitive Walkthrough Method: A Practitioner’s Guide, pp. 105–140. ISBN: 0-471-01877-5. URL: `http://dl.acm.org/citation.cfm?id=189200.189214`.

[79] J Williams. “Internet of Things: Science Fiction or Business Fact?” In: *Harvard Business Review Analytic Services Report* (2014).

[80] Ming-Ji Wu. *Smart Machine and Productivity 4.0 in Taiwan: Now and Future.* May 2016.

[81] Richard Yonck. “Connecting with Our Connected World”. In: *The Futurist* 47.6 (2013), pp. 16–21.