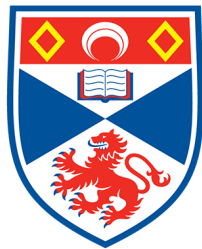


# ArchPrime

An Analysis & Visualisation Tool for Software  
Architecture Viewpoints



University of  
St Andrews

Yuxuan Wang

160000139

Supervisor: Dharini Balasubramaniam

Date of Submission: 27/04/2020

## **Abstract**

Software architecture plays a significant role in the modern software engineering. It is an effective tool for stakeholders communication and software development documentation. In the architecting process, the software architects need to build several views from different viewpoints for different stakeholders and concerns.

At the moment, there are many architecting tools that help the architects to model, visualise and analyse system architecture descriptions. However, most of these tools do not support building architecture models with multiple views. Some tools only support building a model within a single view.

This research project aims to design and implement a software tool, ArchPrime, for software architects to create, visualise and discover the potential relationship between different viewpoints. This report discusses about the background knowledge of architecture viewpoint. It also discusses the design, implementation and evaluation of ArchPrime.

## Declaration

I declare that the material submitted for assessment is my own work except where credit is explicitly given to others by citation or acknowledgement. This work was performed during the current academic year except where otherwise stated. The main text of this project report is 8482 words long, including project specification and plan. In submitting this project report to the University of St Andrews, I give permission for it to be made available for use in accordance with the regulations of the University Library. I also give permission for the report to be made available on the Web, for this work to be used in research within the University of St Andrews, and for any software to be released on an open source basis. I retain the copyright in this work, and ownership of any resulting intellectual property.

## Coronavirus note

- Due to social distancing policies caused by the coronavirus outbreak, the face-to-face user evaluation is forced to be cancelled. The result of user evaluation is included in the appendices.
- After discussing with the supervisor, two self evaluations are performed instead: the first evaluation compares the functionalities of popular architecture modelling tools and the second evaluation uses Nielsen's heuristics to inspect the usability of ArchPrime (see section 7.3).
- Performing these evaluation takes extra time. Consequently, the implementation of some system functionalities has been delayed or cancelled.

# Contents

<b>1</b>	<b>Introduction</b>	<b>5</b>
1.1	Goals & objectives . . . . .	5
1.2	Achievements . . . . .	6
1.3	Report structure . . . . .	6
<b>2</b>	<b>Context Survey</b>	<b>7</b>
2.1	Background . . . . .	7
2.1.1	Software architecture . . . . .	7
2.1.2	Architecture views & viewpoints . . . . .	8
2.2	State-of-the-art . . . . .	8
2.2.1	Viewpoint research . . . . .	8
2.2.2	Modelling tools . . . . .	12
2.2.3	Tools discussion . . . . .	14
2.3	Technology stack overview . . . . .	15
2.3.1	Data representation . . . . .	15
2.3.2	Visualisation support . . . . .	16
<b>3</b>	<b>Requirements Specification</b>	<b>18</b>
3.1	Functional requirements . . . . .	18
3.1.1	Architectural description . . . . .	18
3.1.2	Architecture model and view . . . . .	18
3.1.3	Application . . . . .	18
3.2	Non-functional requirements . . . . .	19
<b>4</b>	<b>Software Engineering Process</b>	<b>20</b>
4.1	Software development process . . . . .	20
4.2	Ethics . . . . .	20
<b>5</b>	<b>Design</b>	<b>21</b>
5.1	Application architecture . . . . .	21
5.1.1	Client-server architecture style . . . . .	21
5.1.2	MVVM architecture pattern . . . . .	21
5.1.3	RESTful Ajax communication . . . . .	22
5.1.4	Architecture discussion . . . . .	23
5.2	Data representation . . . . .	25
5.2.1	JSON . . . . .	25
5.2.2	ADL adaption . . . . .	25
5.3	GUI & diagram maker . . . . .	25
5.3.1	Application layout . . . . .	25
5.3.2	Diagram maker workbench . . . . .	26
5.4	Architecture description constraint checker . . . . .	27
5.4.1	Irrational communication . . . . .	28
5.4.2	Isolated component . . . . .	29

<b>6</b>	<b>Implementation</b>	<b>30</b>
6.1	Application architecture . . . . .	30
6.1.1	Server-side, file storage & server-client communication . .	30
6.1.2	Client-side framework . . . . .	30
6.2	Data representation . . . . .	32
6.2.1	Top level entities: views model, view and view relation . .	32
6.2.2	Hierarchical level entities: component, interface and connector . . . . .	33
6.2.3	“Intera” functions: intra- and inter-component relationship support . . . . .	33
6.3	Workbench . . . . .	34
6.4	Constraint checker . . . . .	35
<b>7</b>	<b>Evaluation &amp; Critical Appraisal</b>	<b>36</b>
7.1	DOER evaluation . . . . .	36
7.1.1	Primary objectives . . . . .	36
7.1.2	Secondary and tertiary objectives . . . . .	36
7.1.3	Summary . . . . .	36
7.2	User evaluation . . . . .	36
7.3	Self evaluation . . . . .	37
<b>8</b>	<b>Conclusions</b>	<b>39</b>

# 1 Introduction

Architecture is a general term to describe the process of establishing backbones and making design decisions. In the architecting process, architects need to work with different customers of different backgrounds and architecture concerns and create a set of architecture views. For example, the physical structure architects need different views for company stakeholders and construction workers. Analogously, the software architects need different views for all the people involved in the software development process, such as project managers, software engineers and customers.

Many architectural research believes that a single architecture model is not capable of including all design concerns from the architecture description. At the same time, designing a compound model with multiple views can be time-consuming and inconsistencies can occur between two or more views. An ideal architecture modelling tool should provide comprehensive tools for architects to design a consistent, precise architecture model at various levels of abstractions and viewpoints.

## 1.1 Goals & objectives

This research project provides an architecture modelling and visualisation tool called ArchPrime. It should help architects to build hierarchical architectural models using different viewpoint approaches. It should also be able to convert complex architecture logic as comprehensible graphical visualisation.

Primarily, ArchPrime should be able to convert architects' design into multiple hierarchical views and store them as binary data for reuse. Based on these functions, ArchPrime should also help architects to reduce their workloads by providing easy-to-use GUI, simplified operations and validation functions. Optionally, ArchPrime should provide extra functions for software engineering activities, as well as more elegant and aesthetic visualisation representations.

The objectives of ArchPrime include:

1. Implement an architecture modelling tool that helps the users to design hierarchical architecture models with different views.
2. Provide graphical visualisation for different architecture models.
3. Offer analysis tools for architects to discover the relationship between different views.
4. Provide an easy-to-use modelling tool in order to reduce the workloads of architects.

## 1.2 Achievements

Throughout the entire research and development process, I have achieved the following goals:

- Conduct a systematic survey on architecture viewpoint research, including the basic concepts, some different viewpoint frameworks and expressions. Through these surveys, I have summarised some essential properties to build an efficient architecture modelling tools.
- Implement my thoughts. Based on the survey result, I have designed and implemented ArchPrime using up-to-date web application technical stacks.
- Conduct a systematic evaluation on ArchPrime. I have evaluated the usability of ArchPrime using 3 different evaluation methods. All of them give me useful feedback and heuristics that help me to optimise my ideas.

Eventually, ArchPrime is implemented with all basic requirements and some advanced requirements: it defines a specialised architecture description standard that can be transformed between computer-readable files and graphical visualisations. As an modelling tool, ArchPrime provides a concise GUI for users to visualise and interact with different views and supports efficiency-improving functions such as model tree view and constraint checker.

## 1.3 Report structure

This project report includes the following sections:

- Context survey: in this section, the research background of software architecture, architecture view and viewpoint are introduced at the beginning. Then, state-of-the-art about architecture viewpoint research and architecture modelling tools are described, followed by a discussion of tools and technology stacks that are widely used today.
- Requirement specification: based on the context survey, a set of functional and non-functional requirements for ArchPrime are listed in this section.
- Software engineering process: in this section, the software engineering activities used for the development process and the project ethical concerns are demonstrated.
- Design & implementation: in these sections, the key design decisions and technical implementation decisions of ArchPrime will be described and justified.
- Evaluation: in this section, ArchPrime is evaluated in terms of the usability and the functionality using tools comparison and heuristic evaluation.

## 2 Context Survey

Briefly speaking, architecture viewpoint is a filter approach for architecture description, it is a way of manifesting the system from a particular perspective. In this chapter, the concept of software architecture and architecture viewpoint will be expounded at first, followed by the exploration and discussion on viewpoint research and tools. Some open problems and technological stacks on architecture modelling tools will also be discussed.

### 2.1 Background

#### 2.1.1 Software architecture

In the modern software industry, software architecture has received increasing attention because of its capability of creating requirement-focused, reusable and expressive models throughout the software development process. The term “architecture” has been used not only as an analogy to reference the similarities between software design and physical structure design, but also as an evocation of its special notions compared to the traditional software development process. These notions include architecture styles, architecture views and formal training of architects[1].

In the context of software architecture, every system has an implicit architecture as a result of design decisions, regardless of its size, usage and quality[2]. Although the designers may neither explicitly follow any architectural design disciplines nor write any architectural documentations, the system structure can still be characterised to reflect the design decisions.

Another understanding is that every application has at least one architect who is responsible for the establishment and maintenance of architecture description. This role requires solid skills of project managing, system designing, as well as stakeholders communicating and leading[2].

In contrast to the traditional design “stage”, the development of architecture description pervades the entire software development process. The traditional designing activities are bound into a specific period of time after the requirement engineering and the product of these activities are used for the subsequent implementation. As a result, undesirable requirements are referred back to the previous stage for reengineering, and infeasible designs are referred back for redesigning[2]. With architecture description, the requirements are examined by architectural design principles and constraints, which eliminate the counter-productive boundary limits and provide an architecture model with flexible requirement refinement.



### 2.1.2 Architecture views & viewpoints

The thriving of software architecture has led to various research on how to construct comprehensive architecture descriptions and how to use such descriptions in different contexts. It is generally believed that architecture description is a sophisticated documentation trying to capture all functional and non-functional aspects of a system and presenting system information for multiple stakeholders[2]. Another approach[3] has also shown that the models of an architecture description can be utilised in two ways:

1. Documenting existing design decisions: according to Perry and Wolf[1], an architecture description can be represented as a set of architectural elements with a particular form, which is the prototype of some architectural terminologies such as components, connectors and architecture styles.
2. Encouraging stakeholder communications: architecture description can facilitate the development and the evolution of system at different levels. High-level abstraction of architecture view is useful for communication with non-technical stakeholders. While low-level detail of architecture view is convenient for software engineers during implementation.

However, as Kruchten[4] states, architecture descriptions need to deal with not only composition and decomposition, but also styles and aesthetics. It is impossible for architects to capture all aspects in a single blueprint. Therefore, different viewpoints require different levels of abstraction that address the stakeholder concerns at an adequate level. This approach induces the notion of architecture view and viewpoint.

The IEEE 42010:2011 standard[5] defines the architecture viewpoint as an work product that establishing the conventions for its corresponding view to frame specific stakeholder concerns. The architecture view should adhere to the conventions of its governing viewpoint. Taylor et al.[2] also describe architecture viewpoint as the filter of information and architecture view as what stakeholders see when looking at the system through that filter. For example, the physical viewpoint captures hardware entities and their interconnections in the system. Correspondingly, the physical view is an instance of this viewpoint for a particular system, presenting the hardware mapping of that system.

## 2.2 State-of-the-art

### 2.2.1 Viewpoint research

In the industrial environment, architects usually need to confront a number of stakeholders and choose a set of viewpoints to address all stakeholder concerns. Clements et al.[6] classify the views in their research and indicate that architects must understand what views are available for their systems by considering:

- How the system is structured as a set of implementation units: this approach produces views in module view-type, documenting the principle implementation units of the system.
- How the system is structured as a set of interactive elements: this approach produces views in component & connector view-type, documenting the system's unit of execution.
- How the design structures correspond to the system structure: this approach produces views in allocation view-type, documenting the relationship between the software environment and the execution environment.

Each one of these **view-types** defines a set of patterns for different design decisions called the architecture styles[6, 7]. When making architecture description, architects can reuse the predefined styles from a particular view-type to initialise the architecture model. Besides, if a typical system demands a new view, architects can adopt the guidelines and templates of a recommended standard[5, 7]. The relationship between view-types, views and styles are depicted in figure 1.

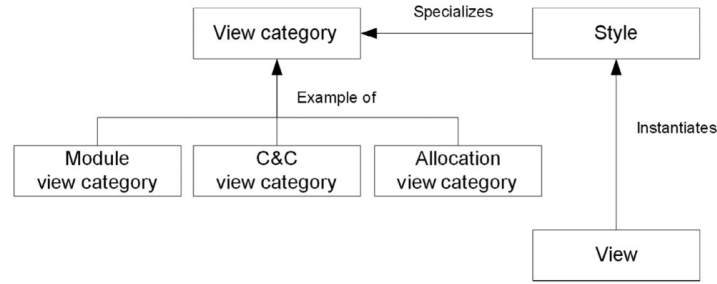


Figure 1: Relationship among views, their categories (view-types) & styles[7]

As an alternative choice, an **architectural framework** uses a set of prescribed viewpoints to establish a general practice for a particular domain of application or stakeholder group[5]. Each prescribed viewpoint in the framework addresses at least one concern from stakeholders. The process of building connections between two viewpoints facilitates the communication and interoperability among the stakeholder group. One of the famous architectural frameworks is Kruchten's "4+1 view model"[4]: in this research, the architecture description is decomposed to four concurrent viewpoints and one real scenario illustration. The architects are allowed to choose different architectural styles, notations, tools and design methods for each view. A view describes the system in a different perspective and addresses a certain concern, as shown in figure 2:

- The logical viewpoint: it describes the system in object-oriented approach and focuses on system's functionality.

- The process viewpoint: it describes the system as a set of executing processes and focuses on system's concurrency and distribution issues.
- The development viewpoint: it describes the system as a set of subsystems and focuses on the actual module organisation and software development environment.
- The physical viewpoint: it describes the system as a mapping between software and hardware, and focuses on the non-functional requirements of the system, such as availability, reliability and security.
- The scenario: it describes the use cases of the system which combines all concerns from above viewpoints and validates the design decisions practically.

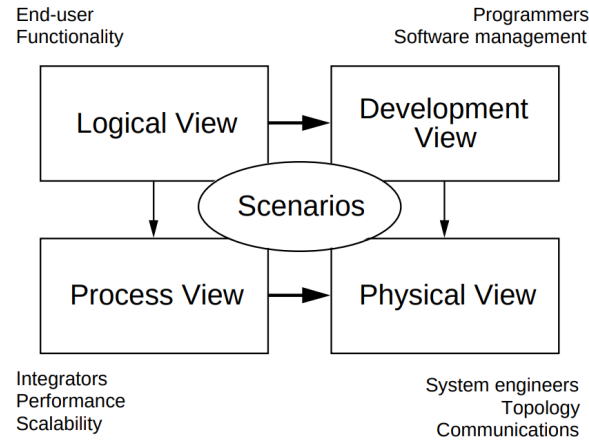


Figure 2: The “4+1” view model[4]

For each view in frameworks like “4+1”, the level of details is determined by the concerns of its corresponding stakeholders. In contrast, there are also architecture frameworks which classifying the views by its level of details. A famous example is from the Siemens, illustrated by Hofmeister et al.[8]. It uses the top-down approach to produce four views corresponding to the view-type approach, as shown in figure 3:

- Before the views are produced, architects firstly identify the factors influence the architecture and derive strategies to produce the design decisions. This process is called global analysis.
- The conceptual viewpoint: this viewpoint explains how the system is conceptually organised in terms of components and connectors. Architects can choose styles from the component & connector view-type to construct this view.

- The module viewpoint: this viewpoint explains how the system can be decomposed to subsystems, modules and interfaces. Architects can choose styles from the module view-type to construct this view.
- The execution viewpoint: this viewpoint explains how the system is deployed and behaved at the runtime in terms of processes and communications. Architects can use styles from component & connector view-type for processes communication and from allocation view-type for hardware deployment.
- The code viewpoint: this viewpoint describes the system implementation in the technical context. Architects can use styles from the allocation view-type.

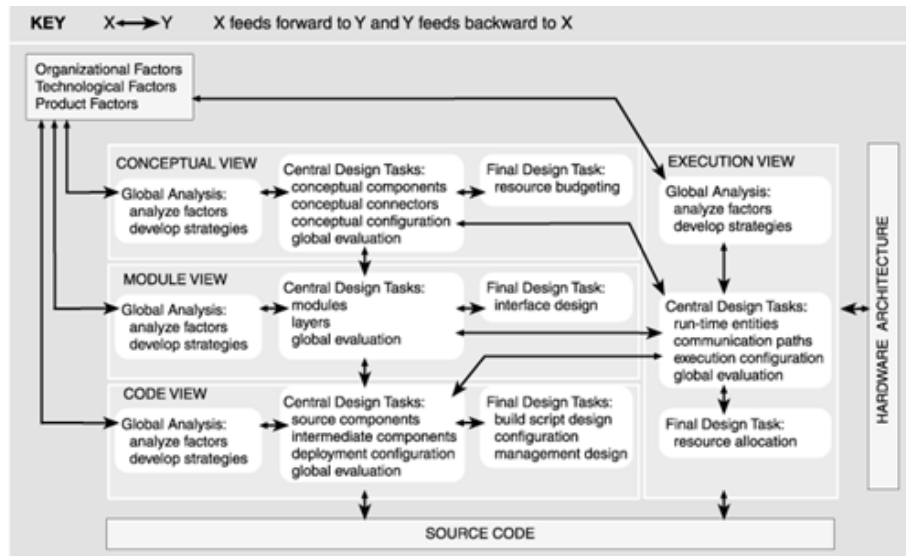


Figure 3: The Siemens four view model[8]

Another example, which is a more fundamentally hierarchical-oriented framework, is the C4 model invented by Simon Brown[9]. It abstracts the architecture of a system into four consecutive levels, as shown in figure 4:

- Context: at the highest level of abstraction, the context view displays the big picture of this system and can be used for both technical and non-technical stakeholders. It focuses on people and other systems which will use the system in advance.
- Container: the system can be decomposed into several modules called the container. The examples of container include platforms (desktop application, mobile app) and large system entities (file system, database). The

container view reveals the shape of architecture description and the responsibilities of each container, it can be used by software developers and supportive stuffs alike.

- **Component:** this view behaves like the classic component-connector view for a container. It shows the responsibilities of components and the interactions between components for architects and developers.
- **Code:** this view displays the final implementation of components and connectors using technical and coding terminologies. It is intended to provide technical supports for software engineers.

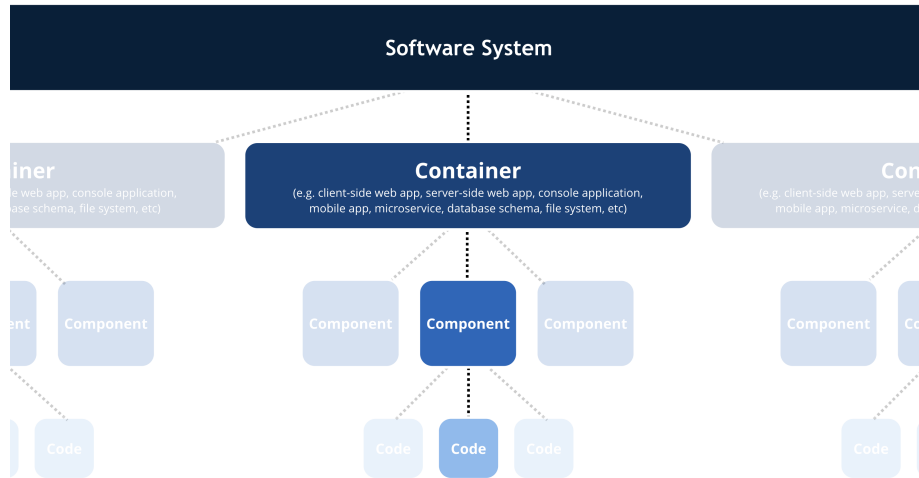


Figure 4: The C4 model[10]

### 2.2.2 Modelling tools

Despite making decisions on stakeholders concerns, it is also important for architects to choose notations and tools for modelling these concerns along different dimensions. In the early design stage, some ‘primitive’ modelling techniques can be useful, such as the natural language descriptions, paper drafts and white-board sketches, these techniques are not developed for architectural usage in particular. Therefore, they can provided flexible and unconstrained expressions for different concerns. However, they tend to become ambiguous and inaccurate as the modelling progresses.

The utilisation of graphical modelling has been greatly enhanced by the emergence of Unified Modelling Language (UML). It not only inherits the flexibility

and expressiveness from graphical representations, but also provides standardised modelling symbols and concepts. In UML 2.0 standard, there are thirteen different viewpoints (or “diagrams”) which covers three categories of viewpoints: the static application structure, the application behaviours and the interactions among application modules[11]. These diagrams provide template architecture elements, forms and stylistic constraints that cover various viewpoints. UML therefore can be used in diverse scenes from business presentations to development activities such as testing and maintenance[2].

Although UML diagrams are much more precise than “primitive” modelling diagrams, they still give priority to graphical generality and flexibility by limiting its semantic precision[2]. Architects thus seek to create formal notations and tools that typically serve for architecture-based modelling purposes, which leads to the development of Architecture Description Language (**ADL**). At the architecture level, it frames one or more concerns from different stakeholders; At the modelling level, it provides at least one model kinds that address different viewpoints[5].

Early ADLs like Darwin[12] and Rapide[13] are developed for specific architectural purposes in research projects. They are able to model a wide range of software systems but not actively used and supported by tools today. As related research deepens, some domain-specific ADLs and extensible ADLs are developed such as Architecture Analysis and Development Language (AADL, which is typically used on modelling embedded and real-time systems[14]) and xADL[15].

```

component{
  id = "datastore";
  description = "Data Store";
  interface{
    id = "datastore.getValues";
    description = "Data Store Get Values Interface";
    direction = "in";
  }
  interface{
    id = "datastore.storeValues";
    description = "Data Store Store Values Interface";
    direction = "in";
  }
}

```

Figure 5: xADLite, an example of textual visualisation way of xADL[2]

In the meantime, evolving software technologies supports architects with numerous visualisation tools. These tools can be categorised into the following types:

- General textual/graphical visualisation tools: these tools enhance the

quality of “primitive” paper and whiteboard drafts by providing easy-to-use and aesthetic interfaces. Although these tools do not support architectural semantics, it allows architects to create and modify visualisation with great simplicity and flexibility.

- Architecture development environments: these tools can be seen as “IDEs for architecture modelling”. Each tool supports one or more architectural semantics by using graphical or semi-graphical visualisations. By contrast to general visualisation tools, these tools require understanding of its underlying architectural knowledge and provide detailed canonical visualisation in exchange.

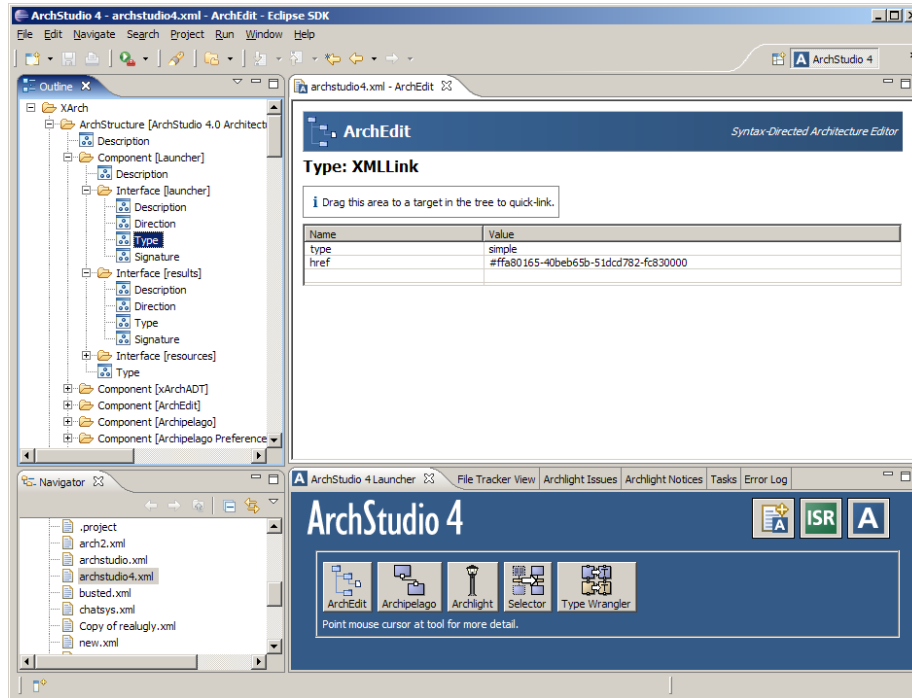


Figure 6: ArchEdit[16], an example of architecture development environment

### 2.2.3 Tools discussion

Once the architecture description is defined, architects need to select at least one viewpoint for their target stakeholders and use suitable tools to create the underlying modelling notations and their canonical visualisations for architecture views. The formats of visualisations can be graphical (like UML), textual (like xADL) or a hybrid of them. In the preceding decades, many notations, visualisations and modelling tools are invented and optimised, each of them focused

on specific sets of stakeholder concerns. Since these concerns are manifold and continuously evolving, single notation or visualisation does not have the ability to capture all these concerns. Hence, when designing and using the modelling technologies, it is important for both tool designers and architects to consider the following problems:

- **What does a stakeholder need?:** one general survey on “what is needed for architecture languages in software industry” [18] has suggested that an ideal and general-purposed architecture description expression does not exist. Personally, I would like to agree with this statement, because I believe the nature of architectural viewpoint is about information filtering and decision making. Given the information and requirements of a system, architects construct a specific view by making design decisions related to corresponding viewpoint and depicting the information based on the background of corresponding stakeholders.
- **What does an architect need?:** with the rise and fall of various architecture modelling tools, some researches also discovered the needs and favours of architects. In the same survey [18], an experiment is carried out on industrial architectural practitioners. The experiment focuses on the needs of architects in the industrial practice by asking them the useful and useless features on existing tools. The experiment result can be concluded as follows: firstly, **most experts and organisations prefer industrial-oriented, generic and light-weighted expressions with extensible collaboration supports**; Secondly, **a practical language should support both communicative and disciplined development semantics**; Thirdly, **the most important requirements for expression functionalities are design, communication and analysis**.

## 2.3 Technology stack overview

Taylor et al. describe the architectural visualisation as a composition of **depiction** and **interaction**, illustrating architects’ choices by providing visual representations and interactive manipulations on them [2]. In the actual implementation, architects firstly need to transform architecture descriptions into compilable data. Next, they need a tool that can depict data as textual, graphical or hybrid visualisation and provide interactive operations on the visualisation.

### 2.3.1 Data representation

In the technical context, architecture descriptions need to be stored and accessed in textual format, so that it can be read, written, compiled and analysed by the computer. On the one hand, many ADLs already have textual format. Most of the ADLs describe the architecture in terms of components, connectors and other entities appeared in the model. They use an object-oriented approach



which characterise these entities along with their properties and behaviours, such as xADLite. Practically, a customised compiler is required for the compilation of these ADLs.

On the other hand, some ADLs are designed to adapt the existing markup languages and data formats, such as XML and JSON. They are more readable and extensible in the actual practice, and there are already lots of tool supports. For instance, xADL[15] is designed to be an XML-based language, every xADL model is inherently a XML document.

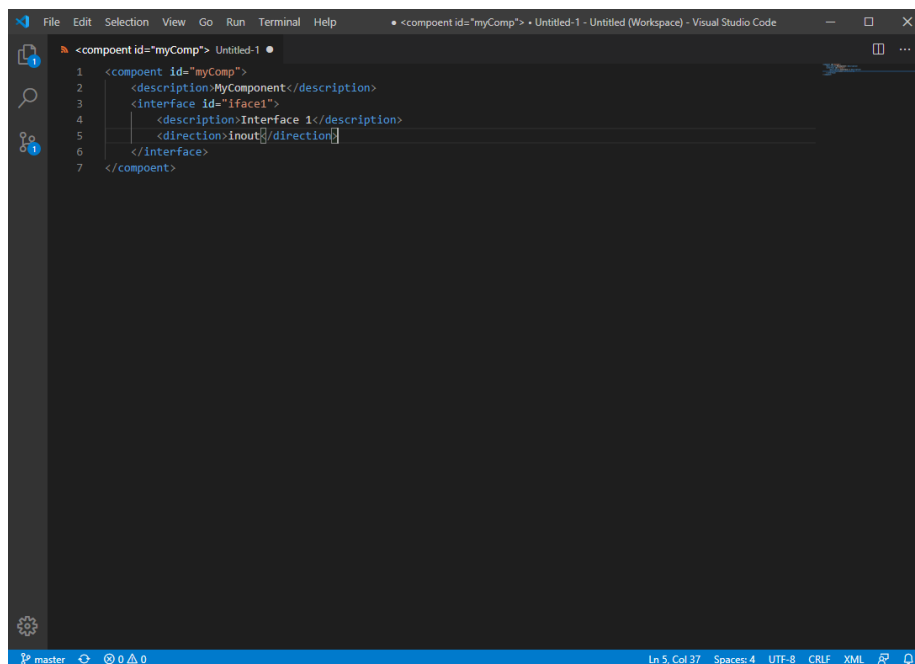
```
<component id="myComp">
  <description>
    MyComponent
  </description>
  <interface id="iface1">
    <description>
      Interface1
    </description>
    <direction>
      inout
    </direction>
  </interface>
</component>
```

Figure 7: xADL document in XML schema[2]

### 2.3.2 Visualisation support

An architecture visualisation tool takes raw data of architecture description as input and outputs human-readable texts, graphs, or a composite of both. Similar to programming languages, the textual visualisation can be displayed through plentiful **text editors**. They are capable of providing colourful font displays, templates and abundant semantic tools. For graphical visualisation, many **graphics libraries** are implemented using different languages. They are capable of rendering wide varieties of lines, shapes, colours and even three-dimensional models. They also support user interactions like dragging-and-dropping through GUI. Analogously, these tools are the “AutoCAD” for the 2D software architecture blueprints.

As web services developing, many graphics libraries are developed on the basis of web application languages including HTML and JavaScript. Comparing to traditional graphics libraries such as OpenGL and DirectX series, they are highly responsive and can be directly used within the web browser supports.

A screenshot of the Visual Studio Code editor interface. The top menu bar includes File, Edit, Selection, View, Go, Run, Terminal, and Help. The title bar shows the active file as '<component id="myComp"> - Untitled-1 - Untitled (Workspace) - Visual Studio Code'. The left sidebar contains icons for Explorer, Search, and Source Control. The main editor area displays an xADL document with the following content:

```
1 <component id="myComp">
2   <description>MyComponent</description>
3   <interface id="iface1">
4     <description>Interface 1</description>
5     <direction>inout</direction>
6   </interface>
7 </component>
```

The status bar at the bottom indicates the current file is 'master', the cursor is at 'Ln 5, Col 37', and the file encoding is 'UTF-8' with 'CRLF' line endings. The bottom right corner shows icons for file operations and a search icon.

Figure 8: xADL textual visualisation using Visual Studio Code[19], a popular code editor

## 3 Requirements Specification

The survey on the architecture description[18] suggests that a desirable modelling tool should provide features like easy-to-use interfaces, precise representations and extensible supports. The first set of requirements are developed at the beginning of the project, documented in the DOER. As the development goes on, the priorities of requirements are rearranged. Some new requirements are added and some infeasible requirements are re-specified.

### 3.1 Functional requirements

#### 3.1.1 Architectural description

1. The tool should be able to support at least one ADL or notation for architecture description.
2. The tool should be able to visualise the architecture description data using textual, graphical or hybrid representations.
3. The tool should be able to transform the architecture visualisation into computer-readable data.

#### 3.1.2 Architecture model and view

1. The tool should support creating architecture model in single-level style or hierarchical style.
2. The tool should support a standardised architectural entity representation for visualisation.
3. The tool should provide the function of creating multiple views of a system.
4. The tool should provide functions for discovering correlations and finding inconsistencies across multiple views.

#### 3.1.3 Application

1. The tool should provide a reactive GUI so that the user is able to visualise the architecture description data and interact with it.
2. The tool should provide API for creating, deleting, modifying and accessing the architecture description data.
3. The tool should be able to render the visualisation from the given file and reflect the user updates on the file.

### **3.2 Non-functional requirements**

1. Usability: the tool should provide easy-to-use GUI, the text description should be unambiguous, and the tools should be easy to operate.
2. Reliability: the tool should provide right information to its user. User updates should be correctly reflected on the data representation.
3. Performance: the tool should provide swift reactions to user inputs. The data processing should not interfere user from visualising.
4. Extensibility: the tool should be designed with high extensibility. Code modules can be easily substituted, new functionalities can be easily added to the code without interference.

## 4 Software Engineering Process

### 4.1 Software development process

The software development process lasts for 8 months. At the beginning, the project was proposed under the guidance of academic supervisors, after the initial investigation on the background, the DOER (description, objectives, ethics and resources) document is written and submitted.

The actual coding process lasts for 6 months, the Agile methodology was used throughout the whole development process:

- The priority of requirements is adjusted dynamically, infeasible requirements are obsoleted and new requirements are gradually appended. The software development process timeline is attached at the end of this report using Gantt chart.
- Each week, a simple meeting is arranged. The weekly progress is reported and the problem raised in the development are discussed.
- At the beginning of the second semester, an interim demonstration is arranged. During the demo, a minimal viable product is displayed and some new requirements are added.

In order to gathering user feedback, an user evaluation was arranged at the start of the second semester, the minimal viable product is provided to the user and some feedback were gathered. However, this evaluation was interrupted due to force majeure, and instead an heuristic evaluation was performed, the evaluation result will be discussed in the subsequent sections.

### 4.2 Ethics

This project follows the ethics guidance issued by the School of Computer Science. Self auditing was conducted following the school ethical approval application process. And it was decided that the artefact evaluation form would cover all necessary ethical considerations. After the artefact evaluation form was submitted and approved, this research project began.

In the evaluation process, all participants joined voluntarily and anonymously, no personal or human object information are gathered in the evaluation. All ethical documentations are attached at the end of this report.

## 5 Design

### 5.1 Application architecture

The following architecture approaches are used for the backbone of ArchPrime.

#### 5.1.1 Client-server architecture style

The thriving of Internet and web programming languages have greatly promoted the growth of web applications and SaaS services. These systems usually use the thin client-server architecture style, i.e. the software are stored on the remote servers hosted by companies and delivered through web browser interface on different operating systems and hardware platforms. Compared to traditional thick client-server style, these systems are easier to be deployed, updated and utilised by customers.

#### 5.1.2 MVVM architecture pattern

Although the server-side implementation depends upon the usage of software, the client-side implementation retains the use of browser-based GUI in the recent 10 to 15 years. At the architecture level, Model-View-Viewmodel (MVVM) pattern has gradually substituted Model-View-Controller (MVC) pattern, both of them attempt to separate logics of UI appearance and interaction from logics of data processing:

- Model: in both patterns, the model module implements the data model and its business logic, it receives the update notification from viewmodel and triggers the view state changes.
- View: the view module also behaves in similar ways in these patterns, it renders the GUI on the browser pages, defining the interface layout and handling the user inputs.
- Controller/Viewmodel: in MVC pattern, the controller module behaves as a bridge between model and view, it handles the user actions by defining corresponding “events” and directly maps these events to model updates; By contrast, the viewmodel module substitutes the controller by providing bidirectional binders between model and view, changes of one module can be automatically reflected on another module without further synchronisations or DOM modifications.

MVVM pattern has been largely utilised in modern UI design. Some famous implementations are Vue.js and Angular.js written in JavaScript for web page construction.

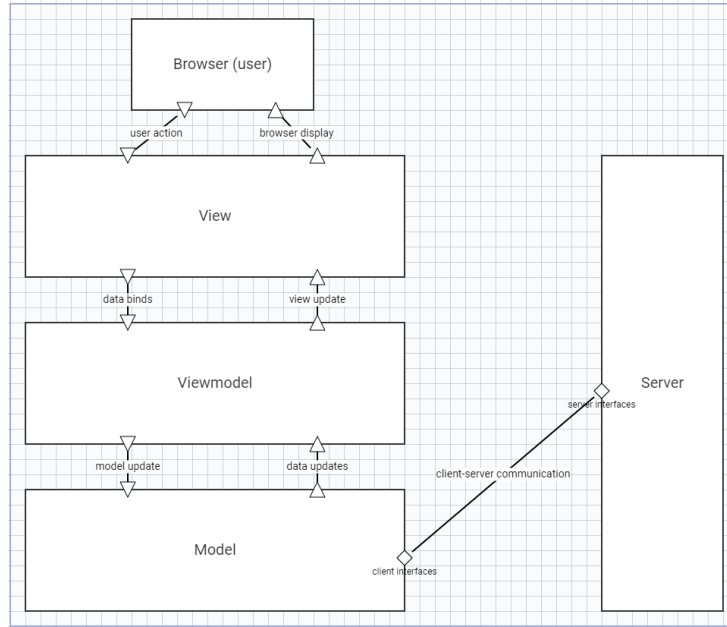


Figure 9: Client-server style with MVVM pattern

### 5.1.3 RESTful Ajax communication

REpresentational State Transfer (REST) is an architecture style designed for network-based systems[20], web services using REST style are called RESTful web services. Generally, REST style provides the following necessary constraints for client-server communication:

- Stateless communication: each client request should be independent to its previous or subsequent requests, it must provide all of the information for server. Additionally, REST style provides API constraints that maximising the utilisation of URI and HTTP.
- Cache: each response should identify itself as either cacheable or non-cacheable, the client should grant resources for reusing and caching if the response is cacheable.
- Uniform interface: each interface involved in the communication should be classified as one of the following types: resource identification, resource manipulation through representations, self-description message and hypermedia as the engine of application state.
- Layered system: cross-layer communication should be avoided, a component should only communicate with components in its previous or next layer.

For any web applications in client-server style, REST style is proved to enhance the performance in component interactions[20].

Ajax, as an abbreviation for “Asynchronous JavaScript and XML” [21], is a series of more practical technologies used for improving communication efficiency and performance between client and server. Ajax technologies appends a client-side Ajax engine that handles JavaScript calls and renders HTML/CSS updates, it allows the client to update itself without waiting for any communication results from server. As a result, the client becomes more responsive as the waiting time is reduced.

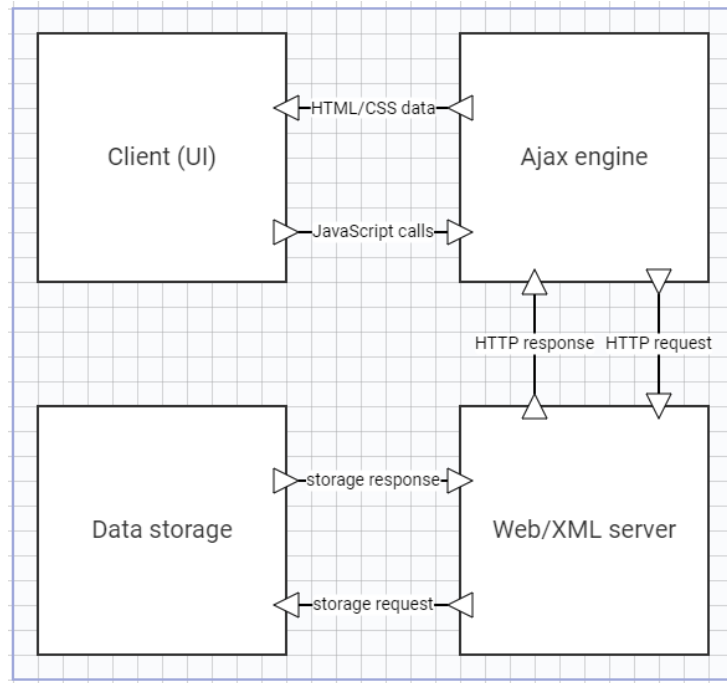


Figure 10: Ajax web application model, adapted from [21]

#### 5.1.4 Architecture discussion

This system is designed to be a web architecture diagram maker focusing on the graphical design functionalities, it uses thin client-server style with GUI in MVVM pattern and RESTful Ajax communication. It has the following main advantages and drawback:

- Fast reaction: rendering and DOM building on view and viewmodel layers are loosely coupled with data processing on model layer, while client-side logic is also loosely coupled with server-side logic. On the one hand, user



actions are directly reflected between view and model, without controller mapping, the client-side reactions are further accelerated compared to the MVC pattern; On the other hand, client has the ability to work independently before data are fetched from server, the time consuming on requests and responses can be significantly reduced due to asynchronous communications.

- High reusability and extensibility: these are the inherent properties from MVVM pattern, modules ranged from a single model function to the whole viewmodel can be reused for new functionalities, web pages or even applications. Since all modules are loosely coupled, obsolete modules can be safely replaced by up-to-date modules, extra modules can be inserted for new functionalities.
- Hard debugging and asynchrony issues: due to the complex data binding in viewmodels and the asynchronous communication between client and server, it is extremely hard to identify the component(s) that blocking the entire dataflow. For instance, designing a test for a communication functionality needs to consider the following parts: whether the server can read and write the corresponding file correctly, whether the server and client (Ajax engine) can communicate correctly, whether the Ajax engine can deliver the result within the tolerable limitations and whether the model can update the viewmodel/view correctly.

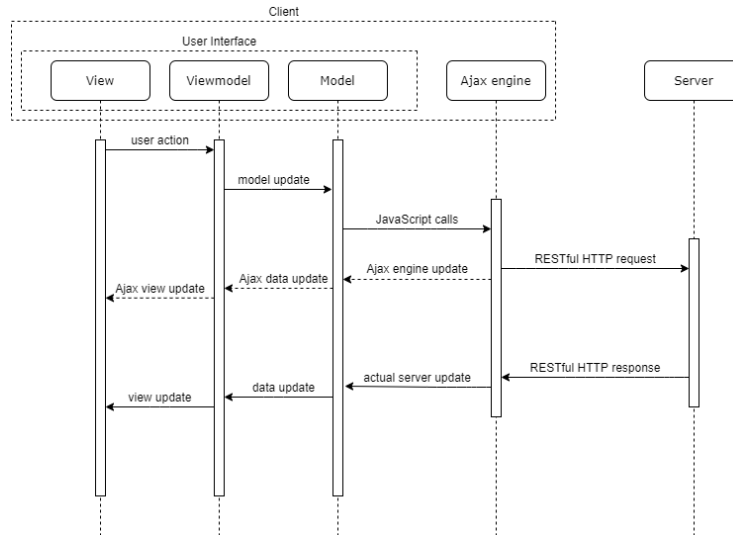


Figure 11: System architecture dataflow (sequence diagram)

## 5.2 Data representation

### 5.2.1 JSON

JavaScript Object Notation (JSON) is a widely used data interchange format. As a substitution for XML, JSON uses human-readable key-value pairs to represent an attribute and its corresponding value. In the context of JavaScript programming, JSON data can be easily read or written through inherent JavaScript methods, it can also be easily transmitted during the Ajax communication.

### 5.2.2 ADL adaption

However, little researches have been carried out for adapting JSON to ADLs, thus a new representation of architecture description based on xADLite has been used in this system. The adaption of xADLite follows the ADL classification and comparison framework[22] which suggests a list of necessary building blocks for ADLs:

- Configuration: a configuration is a connected graph of components and connectors, it displays the topology of architecture description.
- Component: a component is an element representing the unit of computation or data store, it is an encapsulated module that can only communicate through its interfaces.
- Interface: an interface on the component is a communication point between this component and the entire configuration, it is able to represent the (output) services provided or (input) dependencies required by this component.
- Connector: a connector is an element representing the interactions and governing rules among various components, it can be an independent transmission module, such as router, or a communication link, such as HTTP request.

In order to support both single-level configuration and hierarchical configuration, the component is designed in recursive structure: each component is also a configuration which contains its subsidiary components and connectors.

## 5.3 GUI & diagram maker

### 5.3.1 Application layout

The application layout is designed to contain the following components:

- The navigation drawer: this component is put at the left side of the page, it displays a router for core functionalities and an index for user files.
- The application bar: this component is put at the top of the page, it offers necessary tools through buttons and menus for the main workbench

- The workbench: this component displays the core functionality of the current page at the center. For instance, it should display the list of user files at the index page.

### 5.3.2 Diagram maker workbench

The diagram maker workbench is automatically displayed when user accesses the architecture description data, it is designed to contain the following components:

- The visualisation module: this component is what user sees through the web browser, it provides the graphical notations of data using texts, lines, boxes and other 2D graphical visualisation elements. This module is also responsible for handling user inputs and triggering events.
- The translation module: this component translates the textual data into graphical notations with the aid of third-party graphical libraries, it also controls the graphical properties of these notations, such as sizes, shapes and colours.

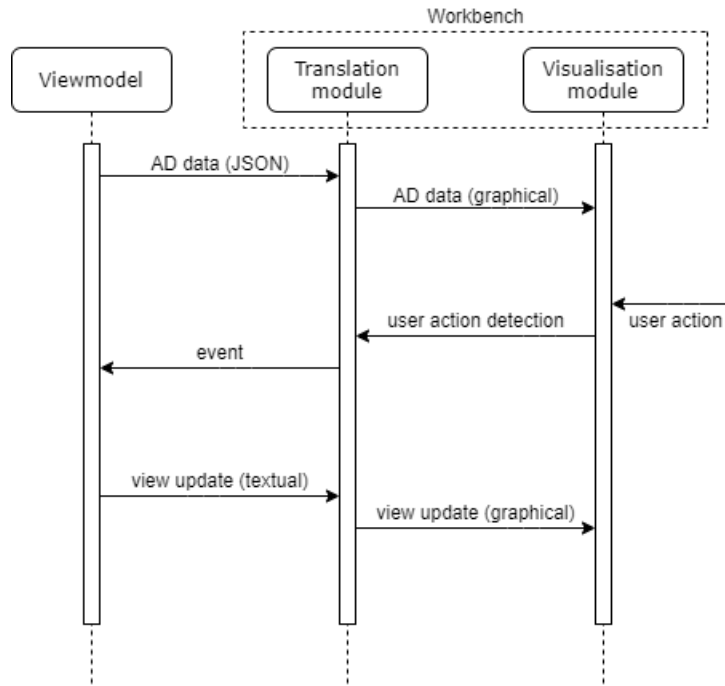


Figure 12: Workbench dataflow (sequence diagram)

## 5.4 Architecture description constraint checker

An architecture model is said to be **accurate** if it conveys correct information about the system, and to be **precise** if it conveys information as detailed as possible[2], an accurate model is able to help architects (and the development teams) to prevent mistakes from architectural drifts and design defects, while a precise model is able to present more information in different dimensions. In the modelling process, high accuracy is preferred over high precision, as inaccurate model may mislead the stakeholders and result in costly errors in later developments[2].

However, it is almost impossible for architects to produce an accurate model at one stroke, requirements may be changed over the development period, and mistakes may be made due to ambiguous descriptions, misunderstandings or merely unintentional operations. In this circumstance, some constraints need to be introduced in advance in order to keep the model accuracy: whenever a change is made on the model, this change is verified by some predefined constraints; If this change does violate the model, it should be either “debugged” by some automation or “highlighted” for further analyses and manual “debugging”.

In this system, a **constraint checker** is introduced on top of the visualisation tool. The checker acts as a unit testing checker for architecture description data, it is composed by multiple testing modules called **unit checkers**. Whenever the raw data is fetched from the server or modified by the user, it requests the access permission on data and concurrently starts the unit checkers. Each unit checker is a testing module that validates a portion of data in accordance with some predefined constraints. The constraint checker collects the validation results and eventually returns the validation analysis that can be displayed to the user.

As previously mentioned, there is a tradeoff between versatility and semantic preciseness when designing the tests, strict constraints are helpful in designing specific viewpoint semantic, while general constraints are helpful in depicting and communicating for different stakeholders. In this system, a general constraint checker is proposed because of two reasons: firstly, rigorous constraints may restrict the expressiveness of some viewpoints and architectural styles, some constraints can be valid in some viewpoints while invalid in others; Secondly, a universal checker is able to provide an modifiable template for further extensions in different semantics. The sections below illustrate some general constraint examples:

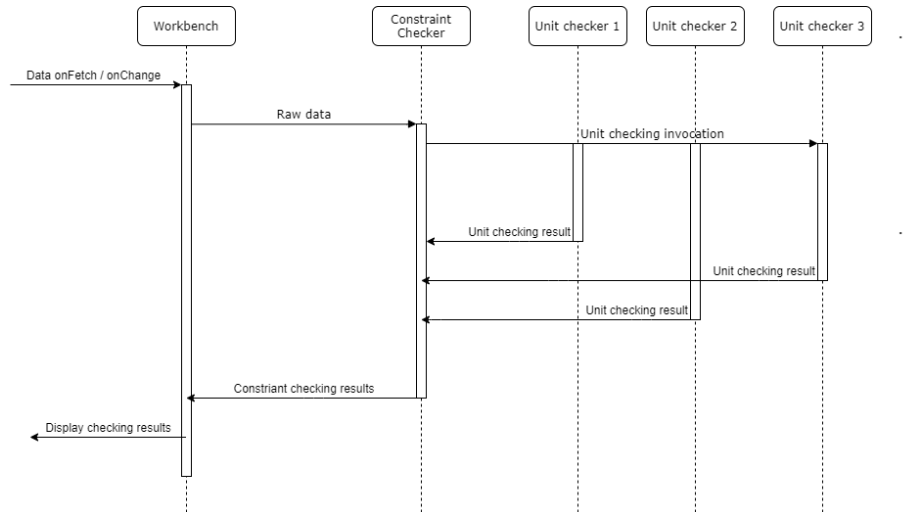


Figure 13: Constraint checker (sequence diagram)

#### 5.4.1 Irrational communication

In the context of computer science, a communication involves at least two entities and a communication channel between them. During the communication, data is transmitted from one side to another side, the format of data is predefined by some communication patterns. For instance, in the client-server model, the client communicates with the server through different kinds of networks by exchanging data, the data format is predefined by different kinds of network protocols. The communication is called “irrational” if the data cannot be transmitted from one architectural entity to another one, the figure below illustrates an example of irrational communication because of mismatched interfaces: the data sent from client or server cannot be received by the other side because two output interfaces are connected.

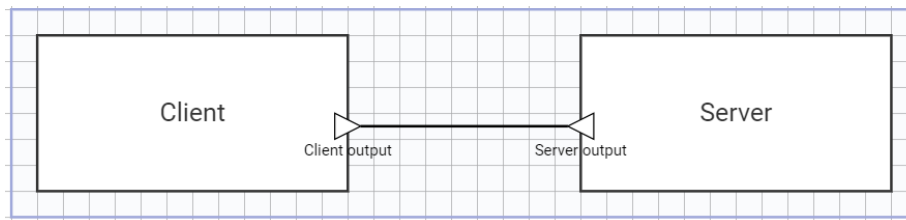


Figure 14: Irrational communication

### 5.4.2 Isolated component

In a complete and functional system, each architectural entity performs its own functions. The components behave as the fundamental units of computation and storage, and the connectors behave as the connection links between components. An architectural entity is called “isolated” if it is presented in the system but not behave as a part of the system, the figures below illustrate two examples: there are two isolated interfaces, one on client called the “client input” and one on server called the “server output”, they are not connected to any other interfaces and consequently they are not able to transmit or receive any data from other components. Also, there is an isolated component called “database”, it does not have any interface and thus is unable to communicate with any other components.

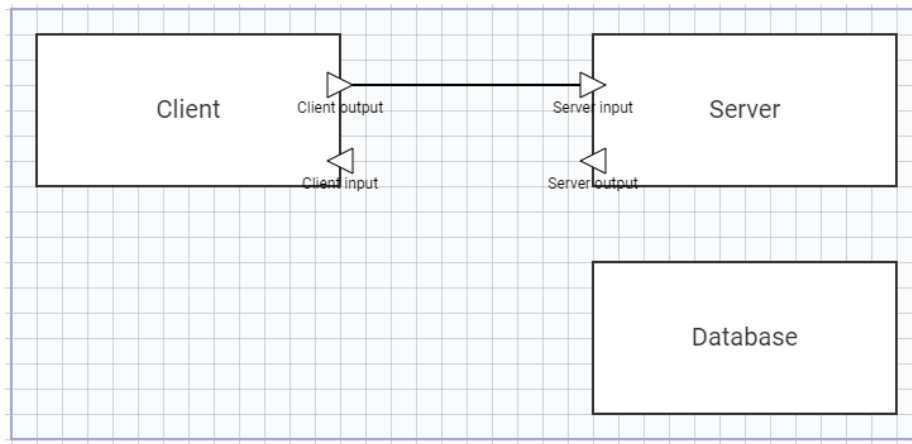


Figure 15: Isolated component and interface

## 6 Implementation

### 6.1 Application architecture

#### 6.1.1 Server-side, file storage & server-client communication

The server-side is implemented in Node.js runtime[23] and Express routing framework[24]. Once a RESTful HTTP request is received, it will be automatically routed to its associated handler, and the result will be delivered back using RESTful HTTP response. All RESTful server APIs are listed in the developer manual section in the appendix.

The server uses local repositories as file storage, inherent JavaScript APIs for file system interactions[25] have been used for synchronously creating, deleting, accessing and manipulating local JSON files. These synchronous I/O methods guarantees the consistency at the cost of comparatively slow I/O speed.

Communication between server and client uses traditional Internet protocol suite, the RESTful style have been guaranteed by using encapsulated HTTP request/response methods. On the server-side, RESTful API definition is supported in Express framework through different routing handlers; On the client-side, Axios[26] is used for generating RESTful asynchronous HTTP requests.

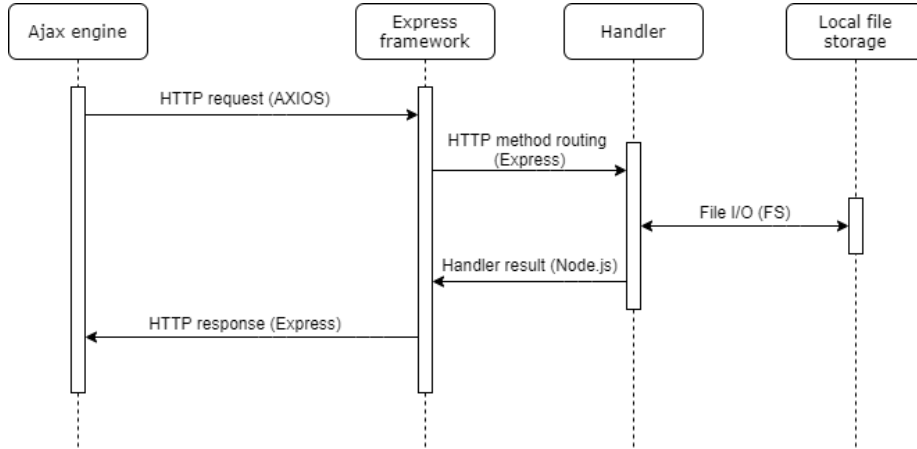


Figure 16: Server and server-client communication dataflow (sequence diagram)

#### 6.1.2 Client-side framework

As mentioned in the design section, Vue.js[27] is a versatile web application framework using MVVM pattern, it provides mature viewmodel binding methods without restricting the performance of DOM building and rendering. For this system, Vuetify UI framework[28] has been used as the backbone of the

entire client-side logic, it provides encapsulated components with additional application layout supports and aesthetic UI elements, such as interactive dialog cards and tree menus.

The navigation drawer is placed at the left side with two functionalities: in tools section, the dashboard navigates user to the index page; In architectures section, each row represents an architecture description document, they navigate user to the workbench and display the corresponding graphical visualisation.

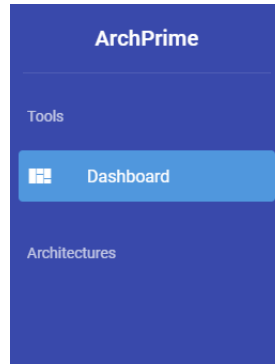


Figure 17: Navigation drawer

The top application bar provides five sets of tools for the workbench:

- View menu: this drop-down list displays all top level configurations (i.e. views).
- View tool: this tool set manages the operations on the top level configurations, it offers functions to create and delete views. Additionally, it offers a function to navigate back to the top level when user wants to enter switch views.
- Configuration tool: this tool set manages the operations at an arbitrary configuration level inside a view. It offers a function for user to navigate back to the previous configuration level, and a function for user to create a new component block at the current configuration level.
- Tree view: this tool displays the architecture description as a hierarchical tree menu. It offers an overview of the relationship between components and connectors at different configuration levels.
- Advice: this tool displays the results returned from constraint checkers, the number denotes how many inconsistencies are found at the current configuration level. It will be discussed later in the subsequent section.

The dashboard is the index of all architecture description documents stored on the server, it displays each file as a card and adds an extra card of creating new





Figure 18: Application bar

documents. When this application is opened, this page will be displayed at the beginning.

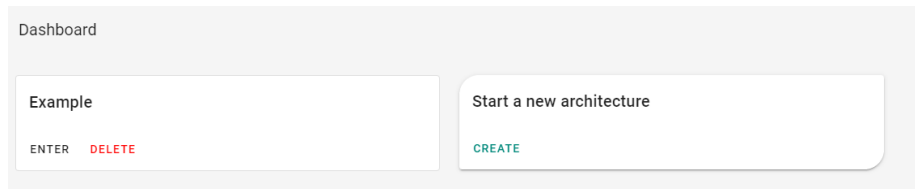


Figure 19: Dashboard

## 6.2 Data representation

### 6.2.1 Top level entities: views model, view and view relation

In this system, each JSON file represents an architecture description document for a particular system, called the **views model**. A views model is composed of at least one **view** implementation, different views can be connected through a **view relation**. In the workbench, the views model is represented as the top level configuration, with views being the components, and view relations being the connectors.

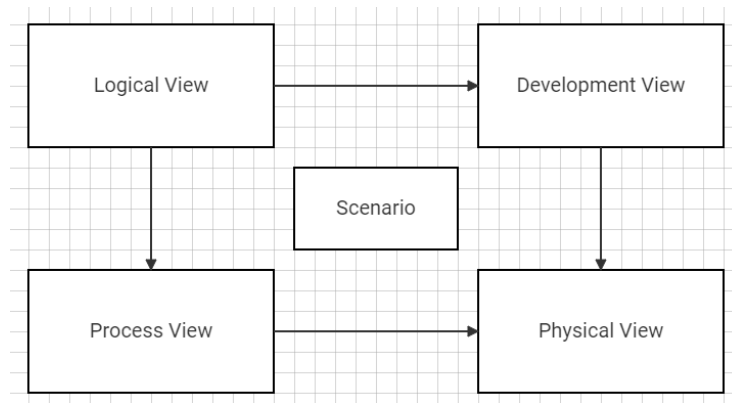


Figure 20: 4+1 view model example using view and view relation, adapted from [4]

### 6.2.2 Hierarchical level entities: component, interface and connector

Each view is composed of an arbitrary amount of rectangular blocks called the **components**. Isolated components are not able to communicate with each others until the **interfaces** are defined. An interface can be input (requiring a dependency), output (providing a service) or non-directional interface (an abstraction of the communication relationship between components). After interfaces are defined, components are able to communicate through links called the **connectors**. The figure below demonstrates the relationship between components, interfaces and connectors.

In this system, the connectors are not permitted to connect directly with components, they must be built between interfaces. This feature helps the architects to understand the data transmission direction and the relationship between two communicating components. However, creating and classifying extra interfaces also introduces more workloads to the architects.

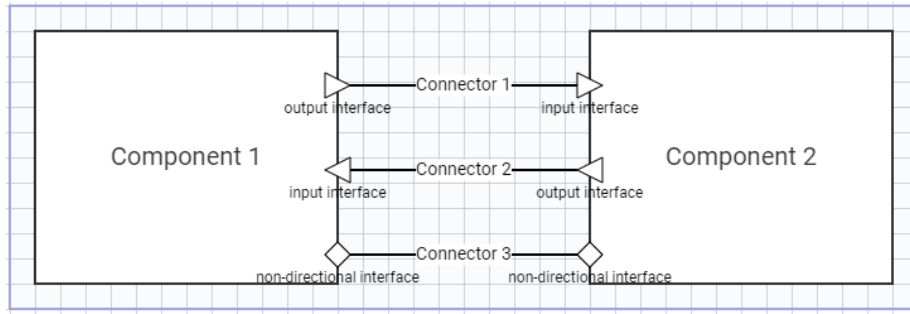


Figure 21: Components, interfaces and connectors

From the perspective of data properties:

- A component is composed of three parts: its graphical notation information (such as block coordinate and size), all of its interfaces, and its subsequent components and connectors if it is in a hierarchical model.
- An interface is composed of two parts: its graphical notation information (such as direction) and its label.
- A connector is composed of two parts: the interfaces between its source and destination and its label.

### 6.2.3 “Intera” functions: intra- and inter-component relationship support

The hierarchical model enables the architect to model the system at different levels of details. However, components need to encapsulate themselves when being used in a single level configuration, which could bring the following issues:

- The intra-component relationship: it is difficult for architects to figure out the relationship between components at two different levels inside a component;
- The inter-component relationship: it is also difficult for architects to figure out the relationship between components at the same level but in two different configurations;

In order to solve these issues, two “intera” features are introduced into the system. The term “intera” is the combination of “inter” and “intra”, they provide the following solutions for cross-component discovery:

1. Intera configuration support: if a configuration is also a component for its precedent level, this configuration will display all interfaces belonging to that component.
2. Intera connector support: based on the previous support, the configuration will also display all connectors belonging to these interfaces.

For instance, there are two connected components at the parent level, the child level components are able to see their parents’ interfaces and the connector between their parents, as the figures below illustrated:

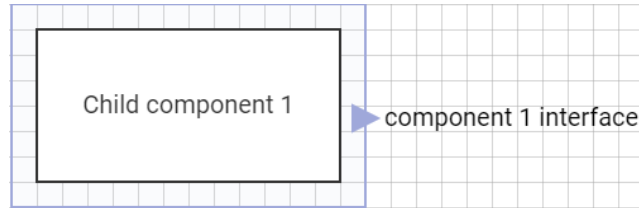


Figure 22: Intera configuration support



Figure 23: Intera connector support

### 6.3 Workbench

The implementation of workbench uses Joint static diagramming library[29]. The architectural entities are rendered as HTML SVG elements, the position and size of a single entity is controlled by HTML and CSS through joint encapsulated functions.

The workbench is made up of four modules, three of these modules are responsible for handling architecture entity data, each of them takes their corresponding parts of data and injects the data into the predefined Joint objects to produce graphical visualisations; There is also another module for handling user inputs as events, it supports the following user actions:

- left clicking: clicking on views or components will enter into their descendent level configurations.
- right clicking: right-clicking on different entities will invoke the context menu, the menu displays a list of functions supported on that entity.
- dragging and dropping: dragging and dropping allows user to reposition the components and connectors.

## 6.4 Constraint checker

The constraint checker module is built in parallel with workbench, whenever the workbench data is updated, the checker synchronises its own data copy and runs the following tests:

- Irrational communication: this test is implemented by iterating all connectors, for each connector it checks the type of its source interface and destination interface. An input interface must be matched to an output interface, and a non-directional interface must be matched to another non-directional one.
- Isolated interface: this test is implemented by recording all interfaces being connected by the connectors and finding the non-connected interfaces.
- Isolated component: this test is implemented by finding all components with no interfaces or with only isolated interface(s).

If the rules are violated, the inconsistent entities will be recorded and reported back to the main checker, they will be displayed with fixing advice as tree views on the application bar.

## **7 Evaluation & Critical Appraisal**

### **7.1 DOER evaluation**

The DOER document acts as the principal guidance during the development, for the initial DOER requirements, the requirements are divided into three categories: the primary objectives are the requisites for the minimal viable product, the secondary objectives extends the system with useful functions, the tertiary objectives further improves the software quality.

#### **7.1.1 Primary objectives**

All primary objectives are completed, the tool is designed to provide hierarchical architectural entities with graphical visualisations, including components and connectors. The intra-component and inter-component communications are supported by the “intera” functions. Also, the constraint check has been implemented to reveal the constraints. Based on the primary objectives, the tool also support features such as tree menu overview, component replication and customisation.

#### **7.1.2 Secondary and tertiary objectives**

The user evaluation has been performed with the combination of heuristic evaluation. In the evaluation, user advice and suggestions are accepted, some advice are transferred into requirements, some programming bugs and design defects are also figured out by the participants and fixed. Besides, the only feature implemented from these objectives is the reuse of files.

#### **7.1.3 Summary**

As the starting point, the DOER document displays the initial concept of ArchPrime as a modelling tool that focuses on architecture view visualisation and analysis. Throughout the entire development process, infeasible DOER requirements are reconsidered and new requirements are added to make ArchPrime more easy-to-use when modelling multiple views. At the end of development process, ArchPrime is been developed as an architecture viewpoint visualisation tool with some analysis functions supported. Although there are still a lot of unimplemented functionalities which could make ArchPrime more user-friendly and more analysis-oriented, the current version of ArchPrime already has the ability to support the architects with informative architecture model visualisations.

### **7.2 User evaluation**

After the minimal viable product is delivered at the middle of the development process, a user evaluation has been conducted. The aim of this user evaluation is to gather user feedback and suggestions and improve the user experience in

advance. At the beginning of the evaluation, the participants are asked to act as architects and design a video game software. They are given questionnaire with three questions which guide them to design two set of architecture models using different viewpoint approaches. After the modelling, two extra questions are given to the participants, asking their experiences as architects and application users.

However, the user evaluation is cancelled due to force majeure (see Coronavirus note). Only 10 users participate before the cancellation. Although these result are not enough for detailed analysis, the suggestions from these participants provides some valuable advice in the debugging process. The result of this user evaluation is documented and attached in the appendices section.

From the incomplete user evaluation, lots of advice and suggestions are received from participants. During their short architecting process, the participants are overall enjoyable when using ArchPrime. Each participant spends around 15-20 minutes to study and discover ArchPrime and finish the questionnaire. The evaluation feedback and some discussions after the evaluation have provided some inspiring ideas for me to finding the implementation bugs and adding new requirements. For example, some participants suggest that application bar could be optimised with more functionalities (and thus I have added the tree view menu function).

### 7.3 Self evaluation

Two substitutions of user evaluation are performed instead after the original evaluation is cancelled. The first evaluation compares the functionalities and non-functional performances between ArchPrime and three popular architecture modelling tools. Some general functionalities and non-functional properties are listed and the performance of these four tools are compared and ranked. The second evaluation uses the heuristic evaluation approach based on Nielsen's research to evaluate the usability of ArchPrime. The result of these evaluations are also documented and attached as tables in the appendices.

In the state-of-the-art evaluation, ArchPrime is compared with three architecture modelling tools, each of them has some similarities with ArchPrime:

- Draw.io: both of them are web applications that support online diagram making services.
- Structurizr: both of them provide online architecture development environments that support building hierarchical architecture models.
- ArchStudio: both of them provide applications for architects to analyse and visualise the architecture description.

On the one hand, ArchPrime offers more architecture-oriented supports compared to general diagramming tools; On the other hand, it is able to provide

more easy-to-use functionalities compared to professional architecture development environments. However, the result also suggests that ArchPrime can do most of the work, but cannot do them perfectly: it only provides simple visualisation elements with shallow architecture-oriented supports.

The heuristic evaluation indicates that ArchPrime has a good satisfaction on the usability of its user interface. For most of the heuristics, ArchPrime is able to deal with them by implementing at least one solution.

## 8 Conclusions

Overall, ArchPrime can be used for visualising architecture descriptions from different architecture viewpoints. Architects can use ArchPrime visualisations as the documentation for architecture models and the communication medium for different stakeholders.

The future work for this research project include but not limited to:

- Providing more analysis tools for architecture viewpoints: the “intera” functions can be further extended to display the relationship between different views. This extension could be extremely helpful when the architects is trying to combine different views and analyse their similarities.
- Providing more supports on specific frameworks and ADLs: on the one hand, ArchPrime could provide template files for architecture frameworks such as “4+1” framework or C4 model; On the other hand, a translator between THE ArchPrime JSON format and general ADL formats can be designed for a better reuse.
- Providing more expressive and aesthetic visualisation elements: the current lines-and-blocks visualisation may still be unambiguous for some stakeholders at the first glance. Some expressive visualisation elements such as Draw.io elements can be supported to make the visualisation more informative and expressive.
- Providing more unit checkers: the current constraint checker provides only 3 unit checkers. In the future, more general constraints can be introduced with careful considerations since these constraints should not limit the users’ actions.
- Providing more tools on navigation drawer and application bar: for example, a user guidance can be added on the drawer for novice users. For professional users, some customised functions such as architectural entity templates and user-defined unit checkers can be added on the application bar.

From this research project, I have learned lots of knowledge about software architecture. I have researched the theory of architecture view and viewpoint in depth and built my personal understanding on architecture studying. In the software development process, I have learnt how to develop an industrial-standard web application using up-to-date frameworks and visualisation libraries. I have also learnt how to evaluate my software from different users using different evaluation approaches.



## References

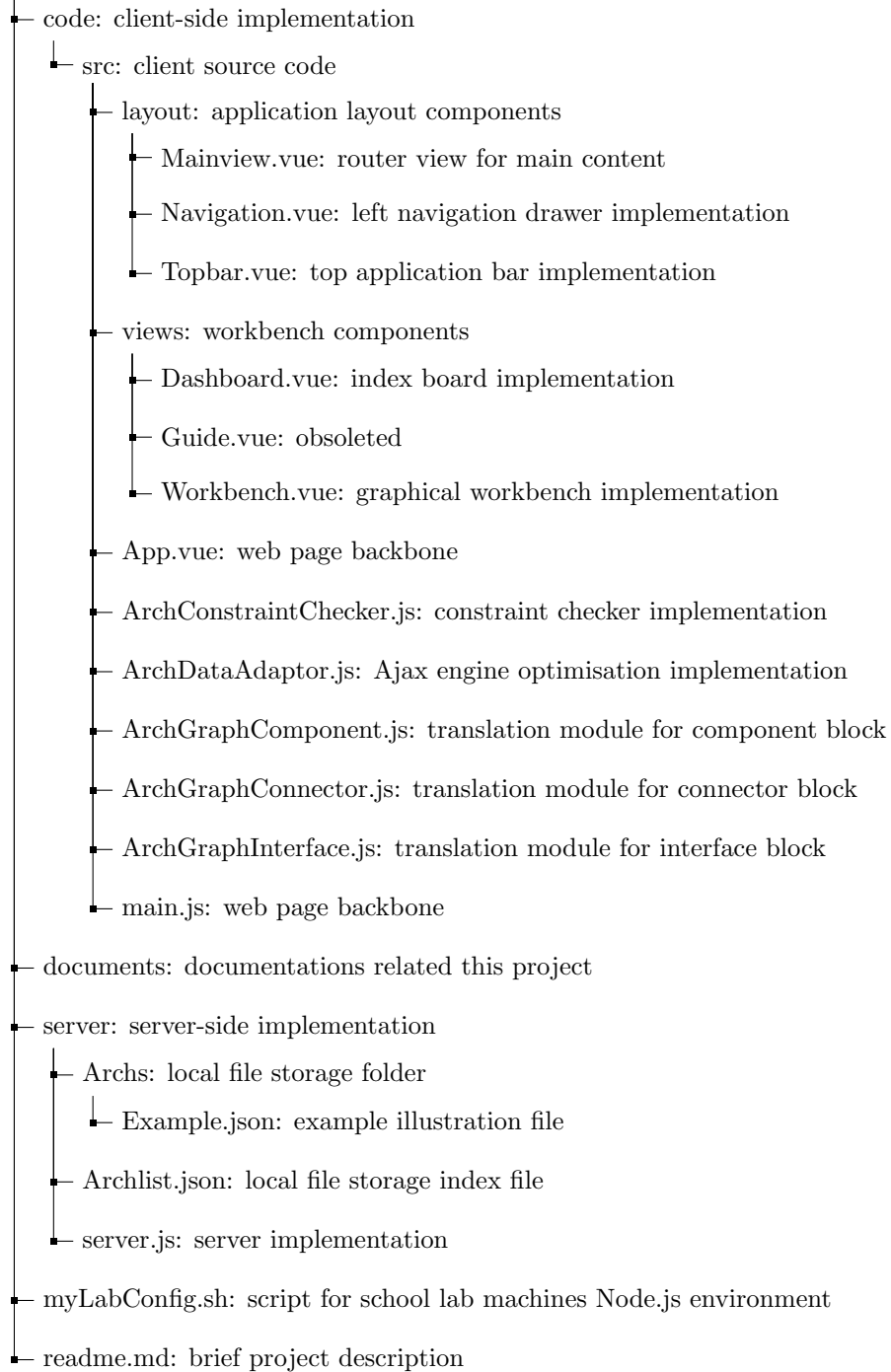
- [1] Perry, D. E., & Wolf, A. L. (1992). Foundations for the study of software architecture. *ACM SIGSOFT Software engineering notes*, 17(4), 40-52.
- [2] Taylor, R. N., Medvidovic, N., & Dashofy, E. (2009). *Software architecture: foundations, theory, and practice*. John Wiley & Sons.
- [3] Sommerville, I. (2015). *Software Engineering*. 10th. In *Book Software Engineering*. 10th, Series *Software Engineering*. Addison-Wesley.
- [4] Kruchten, P. B. (1995). The 4+ 1 view model of architecture. *IEEE software*, 12(6), 42-50.
- [5] ISO/IEC/IEEE Systems and software engineering – Architecture description,” in *ISO/IEC/IEEE 42010:2011(E)* (Revision of ISO/IEC 42010:2007 and IEEE Std 1471-2000) , vol., no., pp.1-46, 1 Dec. 2011
- [6] Clements, P., Garlan, D., Little, R., Nord, R., & Stafford, J. (2003, May). Documenting software architectures: views and beyond. In *25th International Conference on Software Engineering*, 2003. Proceedings. (pp. 740-741). IEEE.
- [7] Tekinerdogan, B., & Sozer, H. (2017). An Architecture Viewpoint for Modeling Dynamically Configurable Software Systems. In *Managing Trade-Offs in Adaptable Software Architectures* (pp. 79-97). Morgan Kaufmann.
- [8] Hofmeister, C., Nord, R., & Soni, D. (2000). *Applied software architecture*. Addison-Wesley Professional.
- [9] Brown, S. (2018). The C4 model for software architecture.
- [10] The C4 model for visualising software architecture, <https://c4model.com/>
- [11] OMG’s Unified Modelling Language, <https://www.uml.org/what-is-uml.htm>
- [12] Magee, J., Dulay, N., Eisenbach, S., & Kramer, J. (1995, September). Specifying distributed software architectures. In *European Software Engineering Conference* (pp. 137-153). Springer, Berlin, Heidelberg.
- [13] Luckham, D. C., Kenney, J. J., Augustin, L. M., Vera, J., Bryan, D., & Mann, W. (1995). Specification and analysis of system architecture using Rapide. *IEEE Transactions on Software Engineering*, 21(4), 336-354.
- [14] Feiler, P. H., Lewis, B., & Vestal, S. (2003). The SAE avionics architecture description language (AADL) standard: A basis for model-based architecture-driven embedded systems engineering. ARMY AVIATION AND MISSILE COMMAND REDSTONE ARSENAL AL.

- [15] Dashofy, E. M., Hoek, A. V. D., & Taylor, R. N. (2005). A comprehensive approach for the development of modular software architecture description languages. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 14(2), 199-245.
- [16] ArchEdit: A Syntax-Directed Editor for xADL Documents, <http://isr.uci.edu/projects/archstudio/archedit.html>
- [17] Structurizr, <https://structurizr.com/>
- [18] Malavolta, I., Lago, P., Muccini, H., Pelliccione, P., & Tang, A. (2012). What industry needs from architectural languages: A survey. *IEEE Transactions on Software Engineering*, 39(6), 869-891.
- [19] Visual Studio Code, <https://code.visualstudio.com/>
- [20] Fielding, R. T., & Taylor, R. N. (2000). Architectural styles and the design of network-based software architectures (Vol. 7). Irvine: University of California, Irvine.
- [21] Garrett, J. J. (2005). Ajax: A new approach to web applications.
- [22] Medvidovic, N., & Taylor, R. N. (2000). A classification and comparison framework for software architecture description languages. *IEEE Transactions on software engineering*, 26(1), 70-93.
- [23] Node.js, <https://nodejs.org/en/>
- [24] Express basic routing, <https://expressjs.com/en/starter/basic-routing.html>
- [25] Node.js file system, <https://nodejs.org/api/fs.html>
- [26] Axios library, <https://github.com/axios/axios>
- [27] Vue.js, <https://vuejs.org/>
- [28] Vuetify, <https://vuetifyjs.com/en/>
- [29] Joint, <https://github.com/clientIO/joint>

## Appendix

1. Repository structure
2. User Manual
  - Download & Installation
  - Application Introduction
3. Evaluation Summary
  - User Evaluation Summary
  - State-of-the-art Evaluation
  - Heuristic Evaluation
4. User Evaluation Questionnaire
5. Software Development Process Timeline
6. Developer Manual
  - Architecture Entity Data Structure
  - Server API
7. DOER
8. Artefact Evaluation Form

## Repository structure



# User Manual

## Download & Installation

### Source code

The source code of ArchPrime can be downloaded in two ways:

- MMS submission
- Github clone: [https://github.com/EI15ande/CS4099\\_SHProject.git](https://github.com/EI15ande/CS4099_SHProject.git)

### Server installation

In the `/server` directory:

1. Install Node.js dependencies if not installed: *`npm install express cors`*
2. Start server process: *`node server.js`*
3. The server will be hosted at `localhost:20804`

### Client installation

In the `/code/src` directory:

1. Install Node.js dependencies if not installed: *`npm install`*
2. An all-in-one running script has been prepared for users, the client can be started by:  
*`npm run dev`*
3. The client will be hosted at `localhost:8080`. Users can open their web browsers and access [localhost:8080](http://localhost:8080).
4. The server must be started before the client starts, otherwise files cannot be read.

# Application Introduction

## Dashboard

When both server and client are started, ArchPrime will be started defaultly at the dashboard page (figure 1).

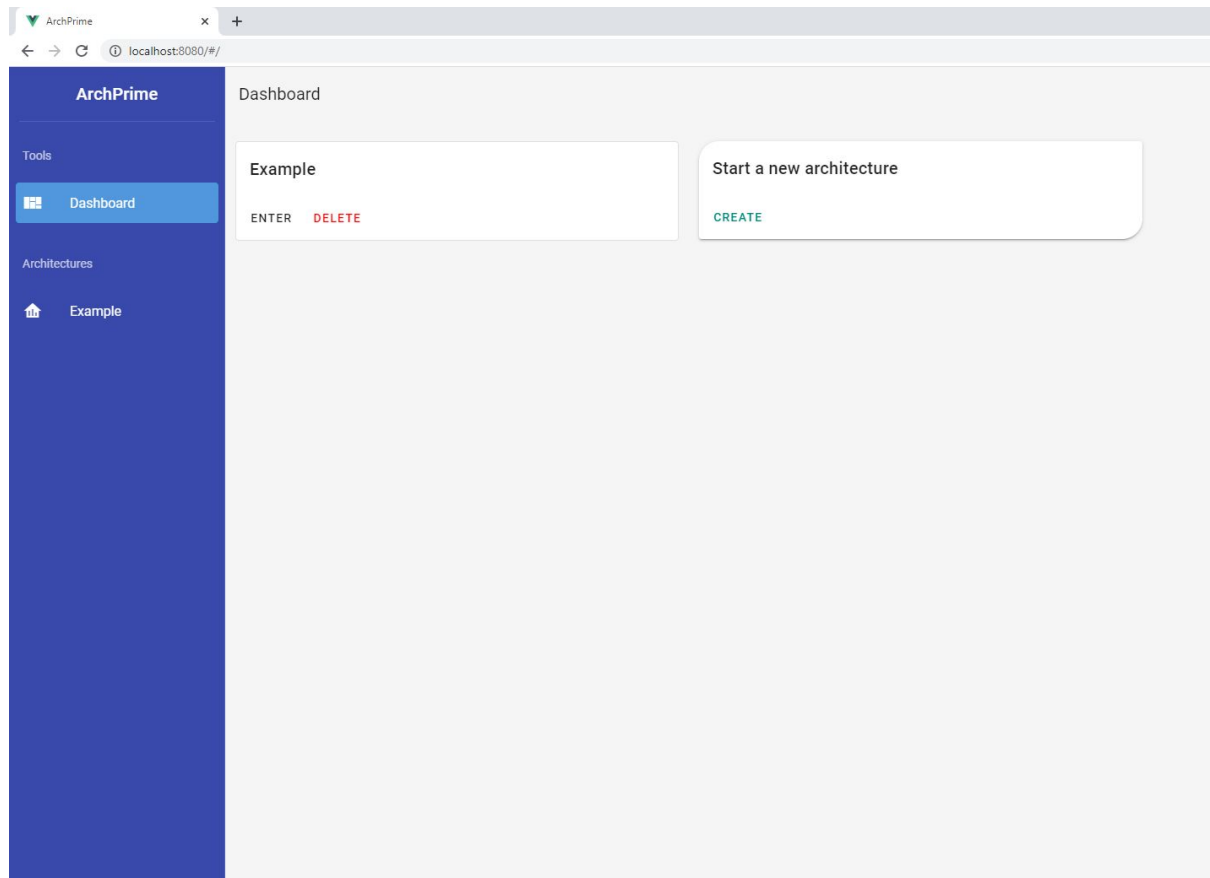


Figure 1. Dashboard

- Each dashboard card corresponds to an entry on the left navigation drawer. Clicking either 'Enter' button on the card or the drawer entry will direct the users to the graphical workbench.
- All files can be deleted by clicking the 'Delete' button on the card.
- ArchPrime provides a default 'Example' file to illustrate examples for users.
- The users can create a new architecture description file by clicking the 'Create' button and entering the file name.

## Workbench & Top application bar

The workbench layout is displayed in figure 2. Each architecture description file starts from top-level configuration (i.e. first level).

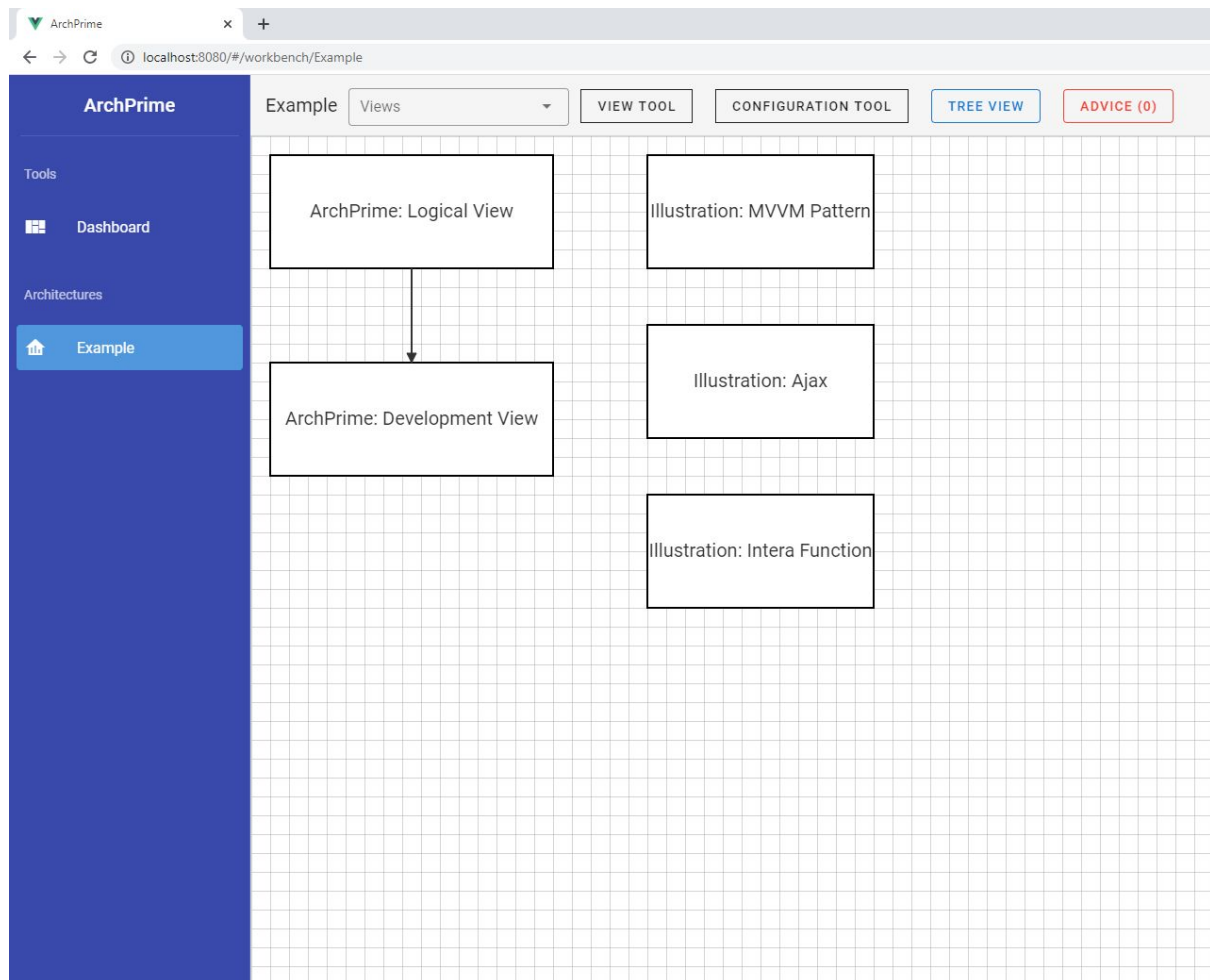


Figure 2. Workbench

- On the workbench page, the users can return back to the dashboard by clicking the 'Dashboard' entry.

The top application bar function:

- Views drop-down list: it displays all views (top-level components) of this architecture description. The users can access the view by clicking the list item (figure 3).
- View tool: it provides 3 functions for all views: returning to the first level, creating a new view and deleting the selected view (figure 4).
- Configuration tool: it provides 3 functions for all non-top-level (i.e. second level, third level, ...) configurations: returning to the previous level and creating a new component at the current level (figure 5).
- Tree view menu: it displays the hierarchical structure of a view. The user can access the entries at different levels by clicking on the icons (figure 6).
- Constraint checker: it displays the checking result from constraint checker. The user can access the test result from unit checkers (figure 7).



Figure 3. View drop-down list

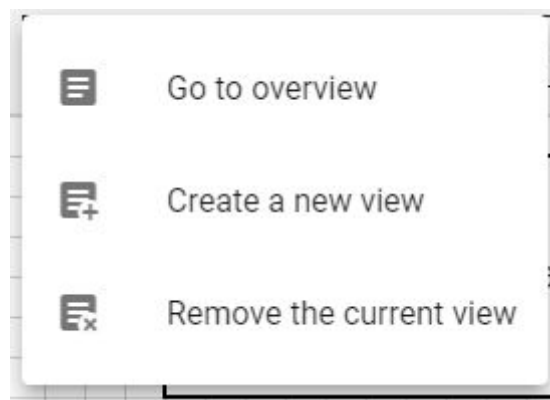


Figure 4. View tool

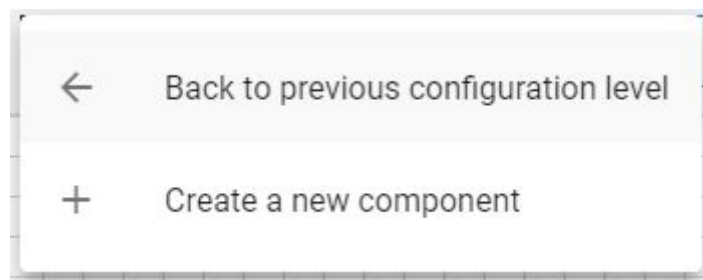


Figure 5. Configuration tool



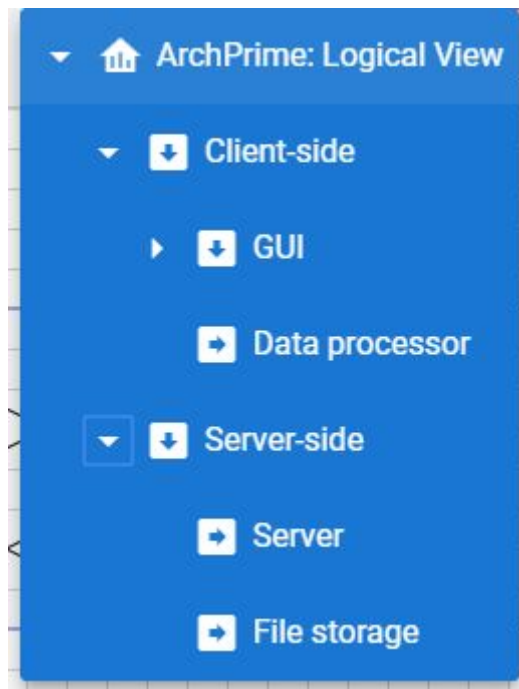


Figure 6. Tree view menu

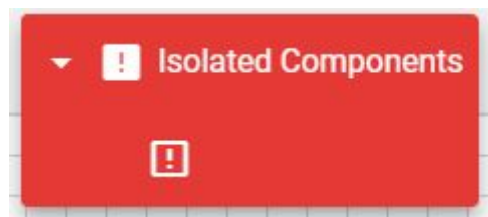


Figure 7. Constraint checker

## Workbench terminologies & operations

Views model: the top level (1st level) configuration.

View: the top level component.

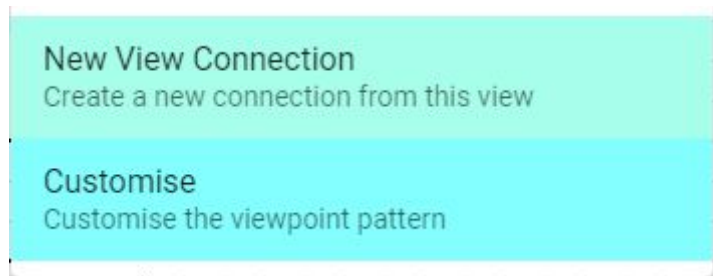


Figure 8. View right-clicking menu

View connection: the top level connector.



Figure 9. View connection right-clicking menu

Configuration: the hierarchical levels (2nd, 3rd, 4th levels...) configuration.

Component: the hierarchical levels component.

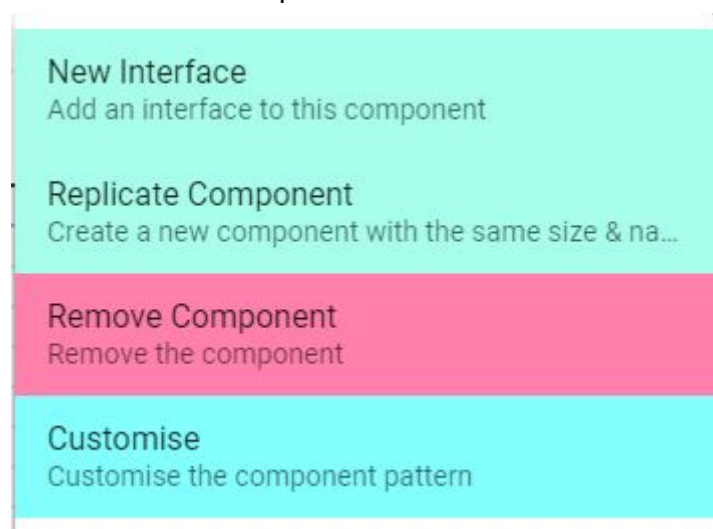


Figure 10. Component right-clicking menu

Connector: the hierarchical levels connector.

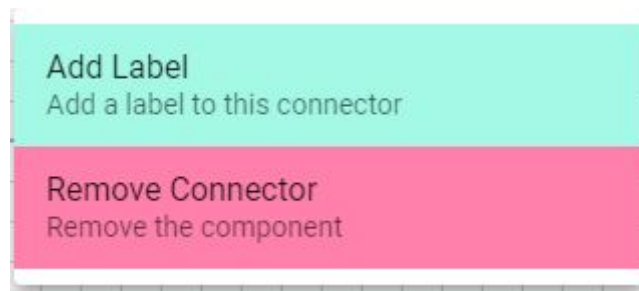


Figure 11. Connector right-clicking menu

Interface: the interface attached on components

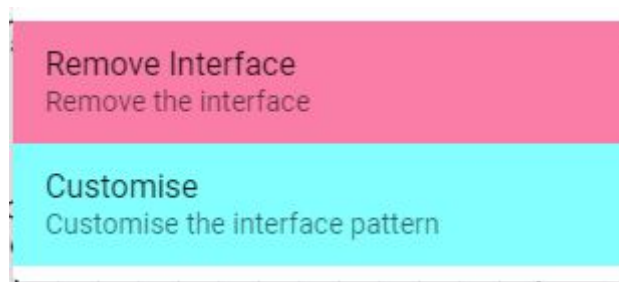


Figure 12. Interface right-clicking menu

# Evaluation Summary

## User Evaluation Summary

### Process

At the beginning, 5 participants are invited to join the first evaluation and complete the questionnaires. However, I have decided to fix the bugs between the time of evaluations so that the version used by different participants is different. This decision has added an extra variable to the software which is not a good approach.

After discussion with my supervisor, I have decided to use a fixed version of software (archived in git branch *Test/UserEvaluation1*). This time, 10 participants are invited to join the evaluation and complete the questionnaires. However, the face-to-face evaluation is forced to be cancelled due to force majeure. The result gathered from these 10 participants will be documented below.

### Questions

The questionnaire asks the participant to briefly design an architecture for a video game.

Question 1 asks about the types of game and participants want to design, all participants want to design video games using client-server or distributed system architecture styles; Question 2 asks participants to act as architects and design an architecture model for project managers. This question potentially asks participants to design the logical view for their games. 9 participants build client and server components at different levels and 1 participant uses the distributed approach.

Question 3 asks participants to act as architects and design an architecture model for a software engineer. This question potentially asks participants to design the development view or deployment view for their games. 8 participants use the object-oriented approach to design a UML diagram with classes, attributes and methods (development view), while 2 participants particularly model the hardware deployments with database storage, distributed servers and client hardware (deployment view).

### Multiple choice feedback

Question 4 and 5 asks for the user experience and feedback/suggestions from the participants:

- Question 4a asks the participants if they can find out any corresponding components, interfaces and connectors between their views. This question evaluates the consistency of participants' input and their feelings. 7 participants can find 1 corresponding entity, and 3 participants can find 2 corresponding entities.
- Question 4b asks the participants if they have used the design experience from their first views in designing the second one. This question evaluates if participants can feel the connections between their models. All participants answer 'Yes' to this question. In particular, some participants state that their second views are based on the decomposition of their first views.
- Question 4c asks the participants if they can identify any inconsistencies between their views. The evaluation is performed before the constraint checkers are

implemented so this question lets participants self-evaluate their views. All participants answer 'No' to this question.

- Question 5 evaluates the user experience from the feedback and suggestions. Each question uses a scale from positive attitude to negative attitude. The number in the table below indicates the number of participants.

	1 (Very negative)	2 (Negative)	3 (Moderate)	4 (Positive)	5 (Very positive)
The visualization is expressive and effective	0	1	1	6	2
The user control is easy-to-use and flexible	(Bad) 0	/	(Moderate) 3	(Good) 5	(Very good) 2
Not enough functionalities are provided	(Always) 3	(Sometimes) 5	/	(Rarely) 1	(Never) 1
The tool is clearly structured	0	1	2	3	4

# State-of-the-art Evaluation

## Background

This is a self-evaluation for ArchPrime. It compares ArchPrime's functionalities and non-functional performances with some state-of-the-art modelling tools. The first tool is draw.io (<https://www.diagrams.net/>), which is a general UML diagram maker. The second tool is Structurizr (<https://structurizr.com/>), which is an online architecture development environment based on the C4 model. The third tool is ArchStudio (<http://isr.uci.edu/projects/archstudio/>), which is another architecture development environment based on the Eclipse IDE.

Functionality / Non-functional property	ArchPrime performance	Draw.io performance	Structurizr performance	ArchStudio performance	Personal ranking for these tools
Visualisation					
Support textual or graphical visualisation on architecture description	ArchPrime supports both xADLite-based JSON data representation and lines-and-boxes visualisation	Draw.io supports free textual or graphical visualisation, it does not have inherent architecture description modelling functions.	Structurizr supports both code-based and diagram-based visualisation on the architecture description at 4 levels of details.	ArchStudio supports a special ArchStudio Architecture Description format that can be used for both textual (ArchEdit) and graphical (Archipelago) visualisations.	1. Structurizr 2. ArchPrime & ArchStudio 3. Draw.io
Support rich graphical visualisation elements for architectural entities	ArchPrime only supports lines, 2D shapes and simple text formats.	Draw.io supports abundant graphical visualisation elements that can be used to design different types of diagrams, these elements can also be used when designing architecture diagrams	Structurizr supports a moderate number of elements for different entities.  For example, it represents stakeholders as a person-shape component.	ArchStudio supports basic line-and-boxes visualisation with simple colours and formats.	1. Draw.io 2. Structurizr 3. ArchPrime & ArchStudio
Support easy-to-use and informative GUI with reactive	ArchPrime supports simple layout (navigation,	Draw.io is designed with a clear layout and an informative	Structurizr supports two kinds of GUI: code-based GUI	ArchStudio uses the layout based on Eclipse IDE, some functions	1. Draw.io 2. Structurizr 3. ArchPrime 4. ArchStudio

user interactions	topbar, workbench) with graphical visualisation workbench. The performance is moderate according to user evaluation.	workbench. It is sensible with user interactions.	and diagram-based GUI. Both of them have easy-to-use toolbars, informative GUI elements and are reactive to user inputs.	may be ambiguous to users. It illustrates the architecture description using an information-rich UI.	
The tool can be easily downloaded, installed and utilised by architects	ArchPrime is a web application specifically for the browser environment.	Draw.io is a free web application with multi-platform support.	Structurizr is a web service with multi-platform support.	ArchStudio is based on Eclipse IDE. It needs to be downloaded, installed and then used.	1. Draw.io 2. ArchPrime & Structurizr 3. ArchStudio
<b>Architecture Description Representation</b>					
Support architects to build an architecture model from a specific viewpoint	ArchPrime provides a modelling tool based on view and viewpoint approach.	Draw.io does not have inherent architecture modelling supports	Structurizr is able to provide an architecture model using the C4 approach.	ArchStudio provides the functionalities of modelling based on architects' decision	1. Structurizr & ArchStudio & ArchPrime 2. Draw.io
Support architects to build hierarchical architecture model at different levels of details	ArchPrime supports hierarchical structure inside a view.	Draw.io only supports single-level diagrams.	Structurizr divides the model at 4 different levels: context, container, component and code.	ArchStudio uses xADL to manipulate data, it can be used to build hierarchical models.	1. Structurizr & ArchPrime & ArchStudio 2. Draw.io
Support architects to build multiple views for an architecture description	In one architecture description, ArchPrime allows architects to create an arbitrary number of views.	Draw.io is able to store multiple diagrams in one diagram file. The diagrams cannot be inherently linked together.	The C4 model provides a single comprehensive view. It can be used repeatedly for different viewpoints.	Archipelago supports users with multiple views for an architecture description.	1. ArchPrime & ArchStudio 2. Structurizr 3. Draw.io
Support architecture analysis and checking tools	ArchPrime has a tree view menu and a constraint checker.	Draw.io does not have inherent architecture-related functions.	Structurizr supports a lot of architecture analysis tools.	ArchStudio has Archlight, which is a built-in analysis tool	1. ArchStudio & Structurizr 2. ArchPrime 3. Draw.io

# Heuristic Evaluation

## Background

This is another self-evaluation for ArchPrime. It uses Nielsen's heuristic evaluation approach (Nielsen, J. (1994)) to inspect the usability of ArchPrime. In this evaluation, 10 heuristics are given and the performance of ArchPrime regarding these heuristics are evaluated.

Nielsen's heuristics	Explanation	ArchPrime Usability
Visibility of system status	The system should always keep users informed about what is going on.	<ol style="list-style-type: none"><li>1. The navigation drawer and application bar help the users to keep track of their current architecture description file and view.</li><li>2. The tree view menu helps the users to identify the structure of the current architecture model.</li></ol>
Match between system and the real world	The system should speak the users' language, with words, phrases and concepts familiar to the user.	<ol style="list-style-type: none"><li>1. ArchPrime uses English as the default application language.</li><li>2. In the modelling process, ArchPrime uses unanimous architectural terminologies based on IEEE 42010 standard.</li><li>3. ArchPrime provides explanations for constraints so that users can easily understand which entity is inconsistent.</li></ol>
User control and freedom	Users often choose system functions by mistake and will need a clearly marked 'emergence exit' to leave the unwanted state.	<ol style="list-style-type: none"><li>1. On the workbench, ArchPrime provides 'Back' and 'Enter' functions for users to freely go up/down between the different levels of a model.</li><li>2. Components, connectors and interfaces can be easily deleted, repositioned and modified.</li></ol>
Consistency and standards	Users should not have wonder whether different words, situations or actions mean the same thing	<ol style="list-style-type: none"><li>1. ArchPrime uses consistent terminologies and semantics.</li></ol>
Error prevention	Either eliminate error-prone conditions or check for them and present users with a confirmation option.	<ol style="list-style-type: none"><li>1. Operational errors such as misclicking and mistyping can be easily reverted and fixed.</li><li>2. Some architectural errors such as inconsistencies can be figured out by the constraint checker.</li></ol>
Recognition rather than recall	Minimise the user's memory load by making objects, actions, and options visible.	<ol style="list-style-type: none"><li>1. ArchPrime allows user actions such as drag-and-drops clicking and text typing through mouse and keyboard.</li><li>2. The application bar at the top provides some helpful tools when designing the model.</li></ol>



Flexibility and efficiency of use	The system can cater both inexperienced and experienced users.	<p>1. For inexperienced users, ArchPrime provides the explanation on the context menu (right-clicking menu) and the constraint checker for novice architects to adjust their models.</p> <p>2. For experienced users, ArchPrime provides a flexible definition of components, connectors and interfaces for professional architects to adapt these entities into different views.</p>
Aesthetic and minimalist design	Dialogues should not contain information which is irrelevant or rarely needed.	1. The dialogue from constraint checker displays the necessary information and fixing advice without irrelevant words.
Help user to recognise, diagnose, and recover from errors	Error messages should be expressed in plain language.	1. The dialogue and error message explains why the architectural entities or operations are invalid.
Help and documentation	Provide help and documentation.	<p>1. ArchPrime provides an 'Example' description for new users to study the tool.</p> <p>2. There is a user manual documenting how to use the tool.</p>

# User Evaluation Questionnaire

## Background

The purpose of this evaluation is to **evaluate the basic functionalities** of my software architecture design tool. This questionnaire will firstly introduce basic tool instructions for participants, and then provide a case study with a series of questions.

Participants need to build an architecture containing two simple configurations based on different viewpoints, then answer some questions. The whole evaluation process for one participant will take about 10-15 minutes.

**Participation in the evaluation is voluntary and participants can decide not to take part at any point during the process. No personal information will be required or recorded during the evaluation process.**

## Instruction

In the beginning, each participant is given 2 rectangular blocks at the 'example' architecture page called '**view**'. Each view is a way to '**represent your software structure towards a different type of people**'. The participant can enter a view through double-clicking, edit the view information or build a connection between views through right-clicking.

Inside a view, the participant can build two types of architectural elements: **component & connector**.

A component is a **processing or data element**. It can be either atomic (single-level) or composite (multi-levels). A connector is a link between two components which needs to be built on the **interfaces** of components. The connector can only be built based on two interfaces of two different components.

In the tool, a component can be created by right-clicking on the blank space, interfaces can be created on a component by right-clicking on that component and a connector can be created by linking two interfaces. Other customisation functions can be accessed by right clicking on these elements.

## Case study: video game franchise architecture

Imagine you are a software architect who is working in a video game studio. Your company wants to start a new series of a game franchise (e.g. Call of Duty, HALO) that will be released on multiple platforms (e.g. PC, PlayStation, Xbox). You may assume:

1. This game is required to be an online game (i.e. involving Internet connection).
2. You are also the studio leader. Your team has all types of technical and marketing support, which includes (but is not limited to) artists, audio team, game designers, technical engineers, customer support, and so on. You are also not limited to any technical or financial issues (e.g. hardware support, budget).

You need to design 2 sets of architecture configurations for 2 different stakeholders: one for your project manager (one who manages the team, but is not involved in the development) and one for your technical team (one who is responsible for a particular technical issue).

Please follow the guide questions on the questionnaire page.

1. What type of game do you want to develop? Do you know any architecture style (e.g. client-server, distributed system, etc.) that would be suitable for your design?

2. Firstly, you need to demonstrate your design decisions with a project manager. He/she does not need the technical details, but an overview of the system in terms of simple technical terminologies (in terms of architecture, logical view):

- a. Component: based on your understanding of system overview, what components are needed in your design for PM? (E.g. a client-server architecture would have a client component & server component)
- b. Interface: the components you have built can only communicate through named interfaces, what interfaces are needed for each component in your design for PM? (E.g. client and server would have HTTP interfaces)
- c. Connector: the communication between interfaces can be established through a connection link, what connectors are needed in your design for PM? (E.g. the client & server HTTP interfaces can be connected to indicate an HTTP communication)
- d. Construct a configuration using the design tool and add connector labels to indicate your choices.

3. Next, you need to demonstrate your design decisions with one of the software engineers:

- a. Which perspective would you like to choose to design the architecture? A suggestion would be your most familiar role based on your past programming experience using any programming ontologies (e.g. Object-oriented, Process-oriented).

- b. Based on the chosen perspective, what necessary components, interfaces & connectors do you need? Follow the steps from 2(a) - 2(d) to construct a configuration.

4. Now you have designed 2 'viewpoints', which are 2 different perspectives for your game, please answer the following **single-choice** questions and write down any further comments:

- a. Between these 2 viewpoints, is there any components or connectors that referencing the same element? If there is, how much?

☐ None

☐ 1

☐ 2

☐ 3+

- b. Does the experience of designing the system overview configuration (1st graph) help you when you are designing the technical-specific configuration (2nd graph)?

☐ No, there is nothing related when I design these configurations

☐ Yes, I have used some experiences from the 1st one to the 2nd one.

- c. In your opinion, are there any conflicts between the views you have designed?

☐ No

☐ Yes

5. Also, as a tool user, I would like to hear some feedback from you. Please rate the tools based on your experience and write down any comments:

- a. The visualisation is expressive: I can understand the visual representations very easily as a designer or as a project manager / a software engineer.

☐ 1 (Very bad)

☐ 2

☐ 3

☐ 4

☐ 5 (Very good)

- b. The controlling is very easy and flexible: I can easily create/modify/delete the elements in the ways I want.

☐ Bad

☐ Moderate

☐ Good

☐ Very good

- c. When I am creating the configuration, some of my ideas cannot be realised by the tool functionalities.

☐ Never

☐ Rarely

☐ Sometimes

☐ Always

- d. The tool is clearly structured and there are no serious design defects.

☐ 1 (No)

☐ 2

☐ 3

☐ 4

☐ 5 (Yes)

- e. I would like a specific guide for how to use the tool.

☐ No, the current tool is OK

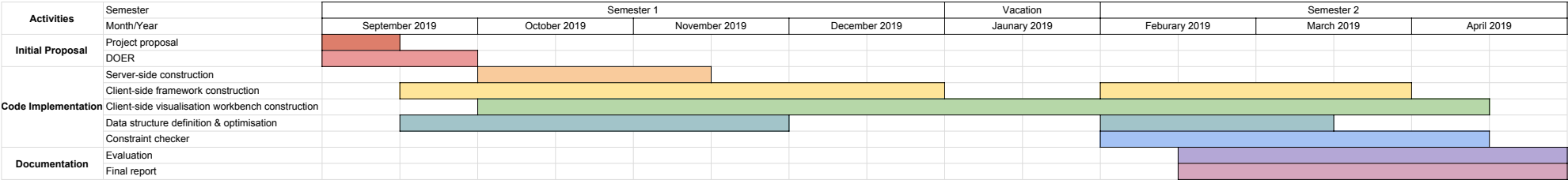
☐ It would be helpful.

☐ Yes, that is necessary.

If you have encountered any bugs during the evaluation, or want to provide some usability feedback, advice or suggestions, please leave it below:

**Thanks for your participation!**

Software Development Process Timeline



## Architecture Entity Data Structure

Features	Architecture Components	Representations	Underlying Data Structure	
Architecture	Views model	Top level overall structure	<b>name</b> (str): architecture name <b>indices</b> (str[]): viewpoint names <b>connections</b> (connection[]): viewpoint connections  [<viewpoint name>] (viewpoint): viewpoint objects	
	View	Top level 'component'	<b>type</b> viewpoint { <b>component</b> (component[]): components array <b>connector</b> (connector[]): connectors array <b>canvas</b> (canvas): canvas info }	<b>type</b> canvas { <b>x</b> (float): position x <b>y</b> (float): position y <b>width</b> (float): visual width <b>height</b> (float): visual height }
	View relation	Top level 'connector'	<b>type</b> connection { <b>source</b> (str/cor): source point/viewpoint <b>target</b> (str/cor): target point/viewpoint <b>label</b> (str): connection label }	<b>type</b> cor { <b>x</b> (float): point x, <b>y</b> (float): point y }
Component (Processing, State)	General component	Rectangle	<b>type</b> component { <b>component</b> (component[]): components array <b>connector</b> (connector[]): connectors array <b>canvas</b> (canvas): canvas info  <b>cpid</b> (int) [unique inside viewpoint]: component unique id <b>cpname</b> (str): component name <b>cpintf</b> (interface[]): component interfaces }	
	General component interface	Port attached on component	<b>type</b> interface { <b>iid</b> (int) [unique inside component]: interface unique id <b>itype</b> (str): interface type <b>ipos</b> (str): interface position <b>iname</b> (str): interface name <b>icount</b> (int): number of connections on this interface [constraint checker special] }	<b>enum</b> ipos { <b>Left</b> <b>Right</b> <b>Top</b> <b>Bottom</b> } <b>enum</b> itype { <b>Input</b> <b>Output</b> <b>Non-directional</b> }
Connector (Interaction)	General connector	Connection line between components interfaces	<b>type</b> connector { <b>source</b> (portid): source interface <b>target</b> (portid): target interface <b>cnlabel</b> (str): connector label }	<b>type</b> portid { <b>cpid</b> (int): component id <b>iid</b> (int): interface id }
Intera communication (intra- & inter-component communication)	Intera configuration	Each configuration level is wrapped by a component with its upper interfaces	component { <b>cpid</b> : INTERA <b>cpname</b> : InteraConfig }	
	Intera connection	Intera configuration port displays connected component & connector information in previous level		

## Server API

HTTP method	Parameter	Usage	Comment
GET	/	No actual usage	Network testing method
GET	/archlist	Get the index of files	Return Archlist.json
GET	/arch/<file name>	Get the specific file	Return <file name>.json
POST	/arch/<file name>	Update the specific file	Updated JSON string needs to be passed in header
PUT	/arch/<file name>	Create a new file with the specific name	Create a <file name>.json file in Arch/ folder, or clear the existing JSON file
DELETE	/arch/<file name>	Delete the specific file	Delete the <file name>.json if existed

# DOER

## Description

**Project title:** Software architecture viewpoint design and visualisation tool

### Description

In software engineering, the process of transforming textual architecture documentations into some diagrammatic or visualisation representations is inevitable for architects and project managers, this could lead to numerous components and models which emphasise distinct focal points regarding different stakeholders and contexts involved in the development (i.e. different architecture viewpoints). In the industrial environment, the static architecture diagram (drawn on the whiteboard or using generic diagram design tools) may be transient and lack visualisation or software engineering support, while professional modelling and analysing tools may be difficult to utilise and thus lead to a large amount of workload. On the other hand, existing software architecture design and visualisation tools (e.g. [Structurizr](#), [Confluence](#)) can satisfy most of the above requirements, but are rarely supported with the features of multiple viewpoints switching and connections revealing.

This project aims to provide a composite software architecture viewpoint design and visualisation tool with customised elements designers:

- For a single architecture view, it will help users design a single architecture diagram/visualisation product with customised components and connectors.
- For multiple architecture views, it will help users reveal the conceptual and/or realistic connections between different components under different viewpoints.

In addition, this tool will provide a user-friendly interface and support simple operations (e.g. drag-drop and text input) on core functions with some extension tools used for different software engineering behaviours. The final software artefact of this project could be a web application that has a strong front-end that focuses on data representation and visualisation, and a light-weight back-end that focuses on data storage.

## Objectives

**Primary objectives:** Software architecture = {Elements, Forms, Rationale/Constraints}

1. Implement an architecture component design tool that provides predefined components for users to build customised architecture components. (progress: single-level architecture component)
2. Support hierarchical features on architecture components. (progress: hierarchical architecture component)
3. Provide visualisation on architecture components.
4. Implement an architecture connector design tool that provides interactions among architectural components. (revealing the relationship by component definitions and



- supporting manually connection) (progress: hierarchical architecture component with intra-components connector)
5. Support hierarchical features on connectors. (progress: a single architecture view - hierarchical architecture component with inter-components connector)
  6. Provide visualisation on connectors.
  7. Implement a design tool that is able to edit and connect multiple views. (progress: composite views with view-view connections)
  8. Support intra-components and inter-components connection. (progress: customised model: composite views with hierarchical view connections)
  9. Based on connections, the tool is able to reveal constraints between different components and views. (progress: core function)
  10. Conduct a user evaluation on the core tool, evaluate the design functions and the visualisation design.

**Secondary objectives:** improve user experience & software quality

11. Improve user experience based on the first evaluation
12. Add IO function: the product diagram / visualisation can be imported or exported for reuse.
13. Implement extension tool(s) that provide further software engineering behaviours. (e.g. Agile developments, existing software engineering tools connection)
14. Conduct the second user evaluation on the overall tool, evaluate the quality and robustness of the system.

**Tertiary objectives:** support further user behaviours

15. Improve software quality furthermore based on the second evaluation.
16. Support collaborations on the tool, enable version control or multi-user editing on a single architecture.
17. Provide API for users to design customised extension tools.

## Ethics

Based on the objective 10 and 14, two separate user evaluations need to be conducted. These user evaluations do not need any personal information from users, but only collect their advice to further optimise the software. The artifact evaluation form is signed and submitted through MMS together.

## Resources

This project does not require any special hardware, software and licenses. All designs and implementations requirements can be fulfilled by current lab machines.

UNIVERSITY OF ST ANDREWS  
TEACHING AND RESEARCH ETHICS COMMITTEE (UTREC)  
SCHOOL OF COMPUTER SCIENCE  
ARTIFACT EVALUATION FORM

Title of project

Software Architecture Viewpoint Design & Visualisation Tool

Name of researcher(s)

Yuxuan Wang

Name of supervisor

Dharini Balasubramaniam

Self audit has been conducted YES ☒ NO ☐

This project is covered by the ethical application CS12476

Signature Student or Researcher

Yuxuan Wang

Print Name

Yuxuan Wang

Date

25/09/2019

Signature Lead Researcher or Supervisor



Print Name

D. BALASUBRAMANIAM

Date

25.09.2019