

Imperial College London
Department of Computing

Immersive Social Visual Analytics

Ahmet Bertan Kavuncu

Supervised by:
Yike Guo

Submitted in part fulfilment of the requirements for the degree of
Doctor of Philosophy in Computing of Imperial College London
the Diploma of Imperial College London, Fall 2020

Declaration

I hereby declare that the material presented in this thesis entitled "Immersive Social Visual Analytics" is the author's own work, except where it is attributed to others.

The copyright of this thesis rests with the author and is made available under a Creative Commons Attribution Non-Commercial No Derivatives licence. Researchers are free to copy, distribute or transmit the thesis on the condition that they attribute it, that they do not use it for commercial purposes and that they do not alter, transform or build upon it. For any reuse or redistribution, researchers must make clear to others the licence terms of this work.

A.Bertan Kavuncu, 2020

Abstract

Increase in data availability in the last decades started a new multi-disciplinary and data-driven approach to be taken in research projects. This new approach requires a matching set of data analytics methods that can target the broadness of data and mixture of expertise in those research projects. We believe that Large High Resolution Display (LHRD) based Immersive and Social Visual Data Analytics system is the solution.

LHRDs are large scale display walls that are built for research purposes, which are ideal for conducting the visual data analytics these multi-disciplinary data-driven research approach requires, by providing unique properties in terms of immersion, collaboration, visualisation and interactivity. This thesis surveyed existing software in the literature for LHRDs and identified many challenges that needed to be solved, in order to conduct visual data analytics for this objective. Hence, this thesis introduces and demonstrates its novel Clustered Visualisation Services Paradigm (CVSP) in its Data Observatory Software Framework (GDO) for conducting Immersive and Social Visual Data Analytics on LHRDs. This paradigm enables system and visualisation app logics to have components on distributed locations of LHRD, where these components communicate, coordinate and form functionality through structured communication using APIs.

Through this paradigm, we enable implementation of immersive and scalable display of versatile, modular, general, interactive, data-driven data visualisation applications and we also enable management of them through our underlying operating system. Furthermore, this paradigm enables collaging and syncing of applications for creating coordinated data visualisation scenes composed of multi-view visualisations that can implement complex interaction scenarios on LHRDs.

In summary, this thesis presents architecture and implementation of Data Observatory Software Framework (GDO), which is used at KPMG Data Observatory, Data Science Institute, Imperial College London; its novel software paradigm: Clustered Visualisation Services Paradigm (CVSP) and many novel or critical functionality provided by this paradigm that overcomes challenges for conducting Immersive and Social Visual Data Analytics for multi-disciplinary data-driven research projects.

Dedication

I would like to express my gratitude to the following people, without whom I could not have completed this thesis:

- **My supervisor Yike Guo** - For your energy, insights, vision and wisdom. I have learned a lot from you. I can not thank you enough for all you have given to me on this road. I will take on your motto and always try to be forward looking like you. It has been a great honour to be supervised by you.
- **My colleague and friend David Birch** - For your support, friendship and numerous contributions to my thesis. It was a great honour to work with you on this project. It was also a great pleasure to have all those brainstorm and design sessions with a like minded colleague like you.
- **My fellow colleagues and friends at Data Science Institute** - For your friendship over the years. You made my work life much more interesting and fun, without you it would be unbearable during those years.
- **My family and friends** - For your endless support and care on this long road. All of you always have been there for me.
- **My lovely wife** - For re-igniting my enthusiasm for completing this thesis and for believing in me blindly.



Where it all started, Data Observatory Construction, from left to right Dave Akroyd, Axel Oehmichen and me (Photo taken by David Birch), 2014.

‘Bertan’s work has made it possible for researchers to present visualisations that show profound patterns that would not be possible to take in at smaller scales. For example, using the entire canvas — roughly 100m2 — to visualise twitter traffic about Brexit made it possible to show a pattern of increasingly polarised discourse that is clearly influenced by bots.’

- *Dr Mark Thomas Kennedy,*
Director of DSI Business Analytics Lab, ICL

‘The Data Observatory is a world-leading visualisation facility providing a platform to gain unique collaborative insights in the context of many different data sets, including the blockchains of leading cryptocurrencies. With respect to the Bitcoin, we have been able to use force-directed layouts to observe the structure of real-time and historical transactions, exposing many complex relationships between transactions and studying the evolution of denial of service attacks. With respect to Ethereum, we have been able to bring to life the increasing adoption of smart contracts. All of this is made possible not only by the custom display, processing and networking hardware of the Data Observatory but also its supporting software stack which makes it easy to deploy and interactively explore visualisations at scale.’

- *Prof William Knottenbelt,*
Director of Centre for Cryptocurrency Research and Engineering, ICL

‘GDO have been used to present to AMR information to the UK minister for health, minister for health of Japan, Shinogi pharmaceuticals. We found the visualisations very helpful in conveying messages around our ‘big data’ sets to clinicians, policymaker and commercial partners in a clear and useful way.’

- *Dr Ceire Costelloe,*
Director of Global Digital Health Unit, ICL

‘The GDO is a great platform for showcasing transport research, including data visualization and impact analysis, to engage with governmental and industrial stakeholders. It also adds scientific values through its distributed computing capabilities so that ultra-granular and real-time traffic information can be processed and seen.’

- *Dr Ke Han,*
Associate Professor at Center for Transport Studies, ICL

Contents

Declaration	3
Abstract	5
Dedication	6
1 Introduction	14
1.1 Problem Statement	17
1.2 Thesis Statement	19
1.3 Summary of Contributions	20
1.4 Organization of Thesis	22
2 Related Work	23
2.1 Visual Data Analytics	23
2.1.1 Big Data	24
2.1.2 Data Science	25
2.1.3 Data Visualisations	25
2.2 Large High Resolution Displays (LHRDs)	26
2.2.1 Origins	27
2.2.2 Types	28
2.2.3 Properties	30
2.2.4 Notable Examples	35
2.2.5 KPMG Data Observatory	37
2.3 LHRD Software and Their Evolution	37
2.3.1 1st Generation LHRD Software	37
2.3.2 2nd Generation LHRD Software	39
2.3.3 2.5 Generation LHRD Software	43
2.3.4 3rd Generation LHRD Software (Current)	45
2.4 Critique of LHRDs Software Platforms for Visual Data Analytics . .	47

3	Design Issues and Challenges	53
3.1	Collaborative Use of LHRDs in Visual Data Analytics	54
3.1.1	Roles on LHRDs	54
3.1.2	Sessions in LHRDs	56
3.1.3	Use-cases	60
3.1.4	Conclusion	64
3.2	Immersive Display and Management of Data Visualisations on LHRDs	64
3.2.1	Display Paradigms	65
3.2.2	Management Paradigms	66
3.2.3	Interaction Paradigms	67
3.2.4	Other Paradigms	69
3.2.5	Conclusion:	71
3.3	Data Visualisation Apps and their properties on LHRDs	72
3.4	Data-Driven, API based, Collaged Data Visualization System on LHRDs	77
3.4.1	Data-Driven Apps	77
3.4.2	API based Distributed System Operation	80
3.4.3	Collaged and Synced Applications	80
3.5	Conclusion and List of Technical Challenges and Requirements . . .	82
4	GDO Architecture	86
4.1	Infrastructure for GDO	86
4.2	Distributed Display Nodes	87
4.3	Server Software	88
4.4	User Control Interfaces	89
4.5	Visualization Apps	90
4.6	Clustered Visualisation Services Paradigm	91
4.7	Conclusion	92
5	GDO Implementation	94
5.1	Creating a Distributed Display Area	94
5.1.1	Display Nodes	95
5.1.2	System Registry	96
5.2	Managing the System and Deploying Apps	97
5.2.1	Core Communication Cycles	98
5.2.2	App Deployment	100
5.2.3	Modes	101
5.2.4	Terminal	102
5.2.5	States	102

5.2.6	Scenarios	102
5.3	Visual Applications and Development	103
5.3.1	Node Component	103
5.3.2	Server Component	104
5.3.3	Control Component	107
5.3.4	Communication Schema	109
5.3.5	Instanting, Configuration and Layering Mechanisms	109
5.3.6	Types of Apps	112
5.3.7	Development of Apps	114
5.3.8	Performance of GDO	115
5.3.9	Conclusion	117
6	Clustered Visualisation Services Paradigm	119
6.1	Implementation of CVSP	122
6.2	Immersive and Scalable Display of Data Visualisations	124
6.3	Data Analytics Oriented LHRD Operating System	129
6.4	Modular and General Data Visualisations	133
6.5	Interactive and Dynamic Data Visualisations	135
6.6	Collaged and Synced Data Visualisations	137
6.7	Integration of Interaction and Sensing Devices	140
6.8	Data Distribution and Load Balancing in Data Intensive Apps	141
6.9	Reliability in Visual Data Analytics on LHRDs	145
6.10	Interoperability, Flexible Setup, Remote Deployment and Casting	148
6.11	Developer Support for Distributed App Development	152
6.12	Conclusion	154
7	Applications of GDO	156
7.1	Map based Apps and Projects	158
7.2	Graph based Apps and Projects	162
7.3	Chart based Apps and Projects	165
7.4	Multi-media based Apps and Projects	169
7.5	Integration based Apps and Projects	175
7.6	Conclusion	180
8	Conclusion	182
8.1	Summary	182
8.2	Contributions	185
8.3	Conclusion and Future Work	187
	Bibliography	189

List of Tables

2.1	Notable examples to LHRDs (hardware) in the literature	35
3.1	Summary of Challenges and Requirements	84
5.1	Developer Template for App Components in GDO	114
5.2	Memory Usage on Server vs Number of Nodes	116
6.1	Visualisation Scaling on SAGE2 vs GDO	127
6.2	Memory Usage on Clients under integration of SAGE2 to GDO . . .	149
6.3	Remote vs Local Latency of GDO Clients	151
7.1	Important Visits that GDO Software was used	156
7.2	Student Projects that have used GDO Software Framework	157
7.3	GDO Software Framework showcases or use appeared on media . . .	158

List of Figures

1.1	KPMG Data Observatory	16
2.1	3Vs of Big Data	24
2.2	Data Science Composition	25
2.3	Multi-view Coordinated Data Visualisation Dashboard	26
2.4	Origin of LHRDs: Cave Automatic Virtual Environment (CAVE)	27
2.5	Various Types of LHRDs Architectures	28
2.6	Various Examples to LHRDs	30
2.7	Parallel Rendering Mechanisms	33
2.8	First Generation LHRDs Software Examples	38
2.9	Second Generation LHRDs Software Examples	40
2.10	Second Generation LHRD Software Architecture Example	41
2.11	2.5 Generation LHRDs Software Examples	44
2.12	SAGE2 Architecture and In use	46
3.1	Examples to Sessions in LHRDs (on GDO)	57
4.1	Project Architecture	93
5.1	System Management Mechanisms	99
5.2	System Management Interfaces	101
5.3	Example to a Distributed Visualisation	103
5.4	Components of Apps	104
5.5	Distribution of App Logic	105
5.6	App Control Interfaces	107
5.7	Base Communication of System and App Operation	108
5.8	Multi-App and Multi-Instance Operation Differences	110
5.9	Operational and Distribution differences of Different App Types	113
5.10	App Development	115
6.1	Clustered Visualisation Services Paradigm	119

6.2	Clustered Visualisation Services Paradigm Communication Schema and Architecture	123
6.3	Sample Collaged Visualisations that are Synced on Temporal	137
6.4	Three-way communication in GDO	145
7.1	Map based App Visualisation and Interface Examples	159
7.2	Maps app being used for Anti-microbial Resistance Visualisations . .	161
7.3	64 Screen Graph with thousands of nodes rendered in Graph App . .	164
7.4	Crypto-currency Graph Visualisation Examples in GDO	165
7.5	Chart based App Visualisation and Interface Examples	166
7.6	Twitter App Control Interfaces and Visualisation Example	167
7.7	Sharing Economy Visualisation	168
7.8	Examples to Multi-media based Apps	170
7.9	XNAT Imaging App Visualisation and Interface	170
7.10	PresentationTool App Interface and Visualisation Example	172
7.11	WebGL App Interface and Visualisation Examples	173
7.12	Fractal Visualisation Examples in GDO	174
7.13	Eye-Tracking App Use-case Example	176
7.14	Hololens Project On-screen AR Overlay	178
7.15	TimeV Query Interface and Control interface	179
7.16	Spreadsheets App Control Interface and in Use	179
8.1	Visual Programming Concept for LHRD based Data Science	189

Chapter 1

Introduction

In the last decades, we have witnessed an increase in interconnectivity and digitalization of the world around us; which led to rise of Big Data and emergence of Data Science. This trend also changed how we approach in general to research projects in the academia and businesses, as they become more multi-disciplinary and data oriented. These research became subject to more data that have emerged in their area and to interconnected data in their surrounding areas. Eventually, increasing the importance of data analytics and cross-field collaboration in these projects.

We also started to see use of a new generation of Visual Data Analytics with emergence of Big Data. In comparison to legacy visual data analytics, this modern visual data analytics uses web-based interactive data visualisations that reads and processes data at cloud level and visualises data according to user preferences in an on-demand fashion. It allows users to analyse different angles and perspectives of data, form logic within data and extract information through allowing visualisation of multiple data sets with various visualisation methods and various data processing or filtering, and displaying these in any layout. Although, this is mostly performed individually and usually displayed on single or multiple monitors. Also, when it is required to be shared, it is shared as an output or a link thus limiting collaboration.

One of the hardware solutions to Visual Data Analytics for research hubs is Large High Resolution Displays (LHRD). These are multi-million pixel visualisation environments that can be constructed using various display technologies such as LCD tiled displays or projectors. Use of LHRDs for Visual Data Analytics is becoming a new trend among research centres as it provides a powerful collaboration

and data analytics environment for data oriented multi-disciplinary research. We see that these hardware can be combined with today's web and cloud based interactive visual data analytics to aid multi-disciplinary data oriented research projects. Furthermore, using unique properties of these visualisation hardware, such as its size, ability to host many spectators or users for collaboration, potential interactivity using various devices; web based and interactive visual analytics techniques can be extended, where users of the environment can surround themselves with data visualisations, interact with them, use them as a medium of communication and exploration. In summary, today's web and cloud based and interactive visual data analytics techniques can be deployed to these unique visualisation environments (LHRDs) and be extended to conduct immersive and social visual data analytics.

Large High Resolution Displays (LHRD) are powered by visualisation and management software that allows users to deploy and distribute visualisations. These software aid research through rendering related visualisations on LHRD screens, thus enabling research team members to collaboratively work with the help of various visualisations on that research. The research area involving advancement of these software does not have a commonly agreed name, therefore we will denote this area as LHRD software research field in this dissertation.

This area existed for decades which during various generations of software have emerged. The first generation emerged in an era where constraints in storage, processing, networking and screen resolution existed. Their common characteristic was that they were based on technologies such as MPI. Their primary purpose was to power LHRDs of that time for the specific use-cases those research were interested in. Second generation intended to generalize the use of these platforms to a certain degree. For this, they focused on render distribution working on top of more recent networking mechanisms. Unfortunately, this generalisation was limited to rendering only and did not account other aspects. Third Generation, the current one, aims to provide scalable, generic, modular, app oriented web based solutions for rapid development and cloud based management. This generation aims to solve previous generations' problems through utilizing web technologies in distribution, server, screens and interfaces areas of the software. However, later generations started to lose focus on visual data analytics and shifted their primary objective to collaboration and productivity, and since older generations cannot provide adequate

level of functionality that exist on today's web based and interactive visual data analytics; we came to the result that, existing software in the literature, past and today, is widely incapable of performing visual data analytics in desired manner in these hardware; which is: using web based and interactive visual data analytics techniques and extending them to LHRD environment through environment's unique properties to conduct immersive and social visual data analytics.

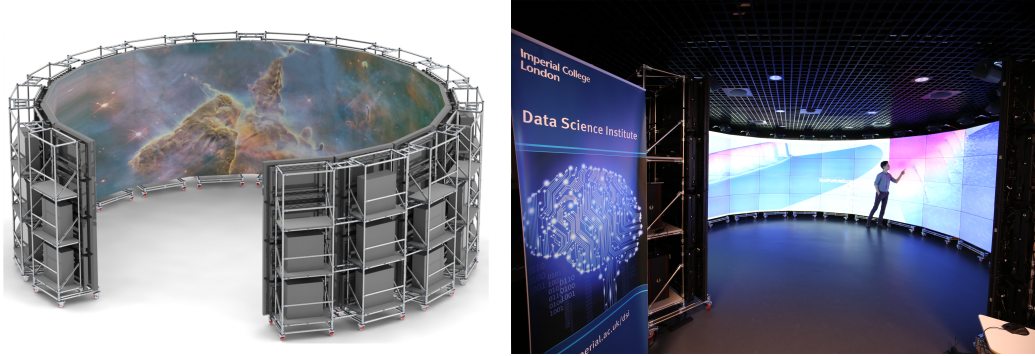


Figure 1.1: KPMG Data Observatory architecture (left) and in use (right).

In 2015, Data Science Institute in Imperial College London have constructed Imperial College Data Observatory (a cylindrical LHRD) also known as KPMG Data Observatory for conducting Visual Data Analytics. The objective of this environment was to become a hub for multi-disciplinary research and aid these research through visual data analytics that utilizes principles of data science. Hence, it required an immersive and social visual data analytics software built for LHRDs to operate. This software should be based on modern visual data analytics that is described here which is web based and interactive, where the target system should expand the use of this in aiding collaboration for multi-disciplinary research in research centres. Instead of users being able to visualise datasets on their computers, they should be able to summon various general and interactive data visualisations on this large display environment and use these visualisations to conduct immersive and social visual data analytics.

As a result, we came up with a novel approach for providing immersive and social visual data analytics in LHRDs and implemented it in our GDO Software Framework, since existing solutions in the literature were inadequate for this pur-

pose. This framework uses .NET as server back-end, JavaScript and Html for client front-end and websocket based communication for distributed operation. It is an app oriented platform, where developers implement general distributed apps for data analytic purposes and other supplementary uses. Users through web interfaces can deploy these apps to sections of LHRDs with given data and configuration. Deployed apps can be controlled and interacted through web interfaces. Moreover, data visualisation apps on the screen-space of LHRD can be synced and programmed for collective operation and thus enable us to create collaged data visualisation scenes.

The software framework uses a novel paradigm called Clustered Visualisation Services Paradigm (CVSP) for LHRDs. This architecture allows system and application code to be dynamically distributed among the server of the software, cluster of display nodes and user control clients; and allow these components to communicate with each other through APIs both locally and remotely, and provide functionality through dynamic distributed operation. This is similar to Service Oriented Architecture or MicroServices although tailored for this specific objective on this specific type of hardware. Through this architecture, we can provide many challenges and requirements we identified in the thesis.

Moreover, through this software approach, we intend to create a base platform for implementing data science oriented apps for LHRDs and be able to deploy various selections of these apps with different configurations, flow data to them, collage them, interact with them for conducting research on this visualisation environment. In this thesis, we will be discussing this novel approach for performing Immersive and Social Visual Data Analytics in LHRDs and its contributions to the literature. The next sections lays out the academic justification of this approach by discussing the problem and thesis statements and its contributions.

1.1 Problem Statement

In our literature survey, we saw that many of the software in the literature enable Visual Data Analytics to a certain degree in their era, whether their primary focus is Visual Data Analytics or not. However, we have also found that Visual Data Analytics methods and expectations have advanced faster than LHRD Software in the literature. At the same time, focus of LHRD software research have shifted to productivity and collaboration rather than Visual Data Analytics and its variation

is decreased.

In the past, the major problem was being able to visualise data in LHRD. The older generations of LHRD research focused on enabling this, where many different solutions emerged for render distribution on distributed displays. Therefore, they focused on render distribution of 2D/3D content for either specific or general visualisations. Basically, if you had a 2D/3D simulation or data visualisation for a research, you were able to render on LHRD using these software after manual implementation or integration. Today, our expectations and requirements of Visual Data Analytics are different.

In general, visual data analytics today is app oriented, modular, general, web-based, cloud-based, on-demand, data-driven and linked. Hence, the software framework to conduct modern visual data analytics of today have to take on these properties and expand them to LHRD environment by adding scalability, collaging, collaboration, casting and other related features and capabilities that benefits from unique properties of these Large Visualisation environments.

As a result, we can start utilizing LHRDs to perform Immersive and Social Visual Data Analytics for various data-driven multi-disciplinary research projects using these modern visual data analytics techniques, where users can summon many general data visualisations to screen-space with given specific data and configuration from mobile devices, perform presentations, discussions and other session types, interact with these data visualisations and control the system, even collage and sync visualisation for comparisons and analytics; and thus enable Immersive and Social Visual Data Analytics.

However, later generation LHRD software, which are limited in number fails on meeting these capabilities. Even though, they are app oriented and enable interaction with screen space and apps on the screen space, their focus shifted to productivity and collaboration. Their software and app paradigms are semi compatible with data visualisations as they are not fully scalable or data-driven, their interaction style is incompatible with desired use scenarios for visual data analytics and they lack advanced methods and techniques such as collaging, programming data visualisations on screen space.

As a result, we can conclude that: Expectations and requirements on Visual Data Analytics of today are beyond capabilities of existing LHRD Software and

therefore existing software in the literature is incompatible or inadequate for performing Visual Data Analytics on LHRDs, either because use of obsolete technology or methods or paradigms, lack of functionality, or having a different objective/focus rather than Visual Data Analytics, and thus they cannot be used to perform immersive and social visual data analytics to aid multi-disciplinary data oriented research projects that are why research facilities like Imperial College Data Observatory are built for.

1.2 Thesis Statement

This thesis argues a new approach is required for conducting modern Visual Data Analytics on LHRDs for new generation of multi-disciplinary data-driven research projects. This approach should solve challenges in the following items on LHRDs software:

- Immersive and Scalable Display of Data Visualisations
- Visual Data Analytics oriented Operating System
- Application Framework for General, Interactive and Modular Data Visualisation Apps
- Data-driven and API based Data Visualisations
- Collaged and Synced Visual Data Scenes
- Integration of Sensing and Interaction Research Devices
- Reliable Distributed Operation
- Interoperability with other Web based Visualisations and Projects
- Flexibility on Display Technologies
- Remote Operation Capabilities
- Developer friendliness for Distributed App Development

Thus, it can result in improvement in providing Visual Data Analytics for new generation multi-disciplinary data-driven research projects.

Each of the current software in the literature offer solutions on few of these items, however none of these software overcomes most of these challenges to an acceptable degree. Furthermore, some challenges like general, interactive, data-driven,

collaged and synced visualisations, that are scalable are not solved by any of these software. Therefore, we present GDO Software Framework and its Clustered Visualisation Services Paradigm (CVSP) that through it we overcome these challenges to enable conducting modern Visual Data Analytics on LHRDs for new generation of multi-disciplinary data-driven research projects.

1.3 Summary of Contributions

Our contributions to the literature is the challenges we solved using our novel software paradigm Clustered Visualisation Services Paradigm (CVSP) of GDO platform, that through it we enable conducting modern visual data analytics for multi-disciplinary research projects. To reiterate, this paradigm enables modular and dynamic distributed components of apps and system on the cluster, server and control clients of LHRD and while enabling a communication schema for all to all API based communication between these distributed components, where we can enable many functionalities we deem necessary, with most significant:

Primary Contributions :

- **Immersive and Scalable Display of Data Visualisations:** General, dynamic and interactive data visualisations and other type of apps can be immersively deployed to a LHRD in any layout and in any size which includes scaling these data visualisations to tens even hundreds of display nodes.
- **Visual Data Analytics oriented Operating System for the LHRD:** A novel underlying operating system is provided for data visualisations that enables app deployment and operation, session management and distributed debugging on the cluster, where also the system can be managed from the web interfaces enabling users to operate the system from tablets/PCs.
- **Application Framework for General, Interactive and Modular Data Visualisation Apps:** General, instanced and dynamic data visualisations that can work with different data and configurations, that are deployed modularly, and be able to scale, which have control interfaces to configure/control/steer these visualisations, which can be casted to other interfaces, can be implemented by the application framework of CVSP.

- **Data-driven and API based Data Visualisations :** Many data acquiring, processing and distribution scenarios, and also many clustering and distributed logic can be implemented on applications of GDO using CVSP.
- **Collaged and Synced Visual Data Scenes :** Collaged and synced data visualisation scenes can be created using distributed coordination between applications or using composite applications that use other applications to visualise.
- **Integration of Sensing and Interaction Research Devices:** Many different interaction and sensing devices can be integrated modularly to LHRD environment and their data and controls can be made available to other applications on the environment and control clients.

Secondary Contributions :

- **Reliable Operation:** CVSP based operation provides reliability within a diverse, dynamic, distributed system, where applications are developed by external developers that are distributed, while using many online libraries and while many of these applications work in communication to create combined functionality.
- **Interoperability with other Web based Visualisations and Projects:** Other web-based LHRD software following control-server-client paradigm or many online visualisations can be integrated into GDO.
- **Flexibility on Display Technologies and Setup :** GDO can work with any number of screens, resolutions or setup without prior installation on its clients.
- **Remote Operation Capabilities :** GDO have capability to be remotely deployed to remote locations in different scenarios and offer synchronization capabilities while doing so.
- **Developer friendly Distributed App Development:** Developers with different skill-sets can develop basic to advanced applications using many .NET and C# libraries on backend and web libraries on front-end, where they can implement also distributed functionality thus deciding how the visualisation acquires, processes, distributes and renders data and how the users can interact

with this visualisation using CVSP architecture and software framework of GDO.

Overall, GDO through its novel architecture and features enable many functionalities we deemed necessary to enable conducting immersive and social visual analytics on LHRDs for multi-disciplinary research projects. Most significantly, in order to enable this level of visual data analytics on LHRDs for data-driven multi-disciplinary research, many of these contributions should be provided together, hence GDO using its CVSP paradigm is the only platform that provides all these novel contributions and other existing but critical functionalities in the literature in a single platform.

Moreover, in alignment with these academic contributions, we also contribute open source (MIT licensed) GDO Software Framework alongside many of its apps on <https://github.com/bkavuncu/GDO>[39] as practical and real world contributions that have been used by many research at Imperial College London (Contribution level of the author on the apps in the repository varies).

1.4 Organization of Thesis

Chapter 1: This chapter introduces the thesis, its contributions and the reasoning behind it.

Chapter 2: This chapter will study various areas that are interest to this thesis like Big Data, Data Visualisation, LHRD Research.

Chapter 3: This chapter will explore design issues for performing Visual Data Analytics and will compose a list of technical requirements for a Visual Data Analytics platform on LHRDs .

Chapter 4: This chapter will introduce the architecture of GDO

Chapter 5: This chapter will discuss implementation of GDO

Chapter 6: This chapter will discuss Clustered Visualisation Services Paradigm and how challenges of this thesis are solved using CVSP.

Chapter 7: This chapter will introduce applications of GDO

Chapter 8: This chapter will summarize the thesis, its contributions and its achievements alongside the future work for this thesis.

Chapter 2

Related Work

This chapter will survey related work in the literature for this thesis. It will start from a brief summary on the general topics of Big Data, Data Science and Visualising Data for giving an overview for the area of Visual Data Analytics. It will continue with Large High Resolution Display (LHRD) Systems which are visualisation environments for research and projects, which we intend to integrate modern Visual Data Analytics with LHRDs for collaborative use in multi-disciplinary research. Afterwards, it will iterate through generations of existing software for these specific type of visualisation environments and finally review these software in order to discover their capabilities in terms of performing visual data analytics on LHRDs.

2.1 Visual Data Analytics

Visual Data Analytics is a key aspect for understanding and working with research projects that are data oriented. Today many of these projects involve large amounts of data. Due to increase in data availability and interconnectivity, data involved for many research project spans to wide areas in their surrounding fields. This results in emergence of multi-disciplinary approaches for conducting data-driven research projects. Thus, visual data analytics becomes an important aspect of this process. It aids research through representing complex data and information as simple and easily understandable visuals and help researchers to comprehend common data and information in multi-disciplinary teams using these visualisations, and finally enable them to communicate and collaborate with each other through using data visualisations. There are few key topics in Visual Data

Analytics: Big Data, Data Science, Visualising Data, and we will go through them in order to understand the transformation this area passed in recent years.

2.1.1 Big Data

In the last decades, the interconnectivity and digitalization of world around us increased incredibly. Today we sense or produce data at an immense way. This has given to rise of Big Data. Big Data conventionally described through its Vs as properties. There are various definitions to these, however we will summarize the most basic 3Vs of Big Data. These are: Volume, Variety and Velocity [157]. Volume defines the change in size of data. Today we generate more data in a day than whole data in the world around 1980s. Variety defines the diversity of data. Today data is highly diverse, we store almost all kinds of data from all possible sources. Velocity defines the speed of data production, processing and storage. In the past, data was stored in batches, today we process and store and even turn it into a product as soon as data gets generated.

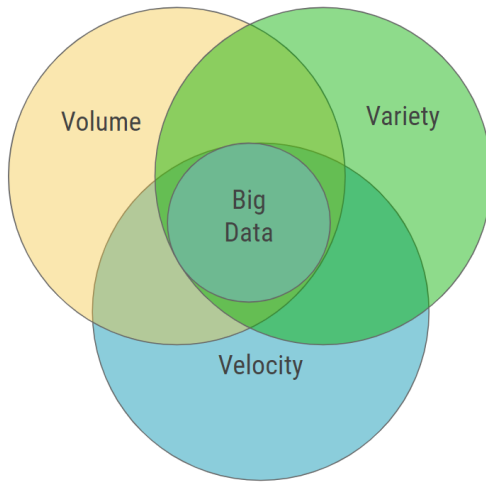


Figure 2.1: 3Vs of Big Data: Volume, Variety and Velocity [157].

The rise of big data also changed solutions. On the lowest level, it changed how we store and process data. Instead of using relational databases, today we use map-reduce based low latency, scalable cloud solutions such as Hadoop, Cassandra or Spark [20]. These systems allow retrieval or processing of data distributed to clusters of machines in milliseconds. It also changed high level solutions as data storage, processing and analytics became core part of many research projects. Data availability on many areas and sectors enabled research projects to utilize data in

their surrounding fields to enhance the research, or even find new solutions based on these newly introduced and linked data. As a result, research projects became more data-driven and started using new paradigms in their roots such as Data Science.

2.1.2 Data Science

Data Science is an inter-disciplinary field that aims to create methods for extracting knowledge and insights from data to be transformed into products and systems. It is a combination of Data Visualisations, Machine Learning, Mathematics, Statistics, Computer Science, Communication and Domain Expertise. Through data science, solutions and analytics are created on a single or multiple domains using this process [56]. As a result, data science can be seen as a paradigm and guide for engaging with data for data-driven research projects in given field or fields.

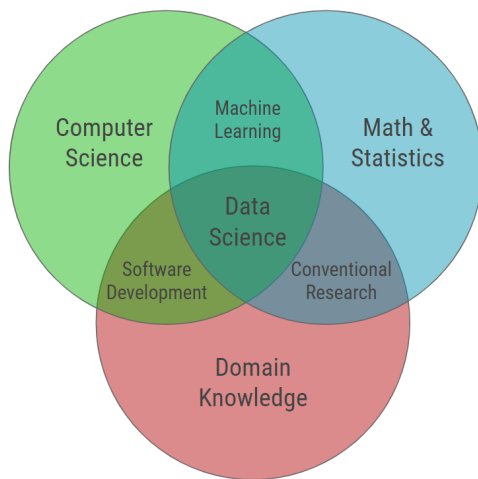


Figure 2.2: Combination of fields that make Data Science

A significant aspect of Data Science and Big Data is how they transformed research projects into being inter-disciplinary both internally and externally. Data-driven research projects that apply data science principles require to have team members from different areas such as computing, mathematics, statistics ..etc. Furthermore, due to data availability, these projects are now subjected to new data surrounding them. Understanding, processing and use of this data requires experts in those domains to directly or externally join these teams. Thus, re-

search project teams of this age become highly diverse in terms of expertise and fields; and require to engage in large volumes of data in diverse areas. Due this trend, data visualisation becomes a key aspect of this process, as communication, collaboration and comprehension is even more needed within the research teams of this age.

2.1.3 Data Visualisations

Data Visualisation is a critical aspect to understand and share information and insights extracted from data. Data contains unlimited angles and perspectives. Thus visualisations work through questions to light up an angle or perspective of

data. For example, what are the top producing cities in US in terms of GDP. Afterwards comes the right selection of visual representation for that query. It can be represented by charts, graphs, plots, networks, matrix, trees, maps, word clouds and many other. This gives creation of a data visualisation. This visualisation than be used to understand the data or one of its angle or perspectives. [99]

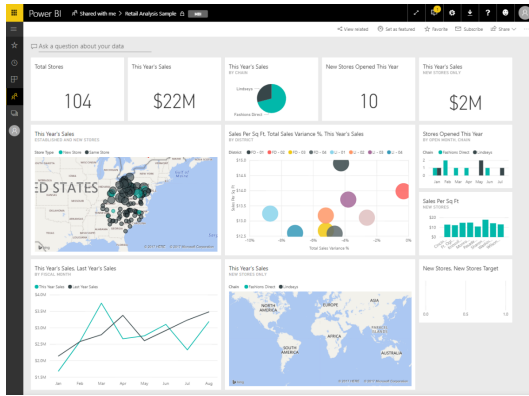


Figure 2.3: Microsoft PowerBI as an example to Multi-view Coordinated Data Visualisation Dashboard [99][102].

Single or multiple visualisations are combined to represent knowledge extracted in a structured way through forming a logic or model with visualisations. If these visualisations are interactive and dynamic, it enables provision of variety of perspectives and angles over data and enable engaging with data and exploring data. If multiple coordinated views of visualisations are provided such as overlays and dashboards, which are created through multiform linked visualisations, these can show connections

and logic between multiple attributes and perspectives of data [99], thus enable data visualisations to be used for communication, collaboration and comprehension within inter-disciplinary research projects. Adding interaction and dynamicity to this enable different stages and processes of data driven research to be executed using data visualisations [87]. The other variable missing in the equation is the environment to display these visualisations, which can support visualisation of Big Data with it's Variety, Volume and Velocity.

2.2 Large High Resolution Displays (LHRDs)

Large High Resolution Displays [12] are multi-million pixel visualisation environments used for various scientific and commercial purposes. In the literature they have been named as Scalable Resolution Shared Displays (SRSD) [129], Scalable Resolution Display Environment (SRDE) [128], Scalable Display Wall Systems (SDWS) [24], Scalable Resolution Display Walls (SRDW) [133], Large High Resolution Displays (LHiRDs) [131], PowerWalls [76], Scalable Tiled Displays [166], Cave Automatic Virtual Environment or Audio Visual Automatic Virtual Environment

(CAVE) [142][29], Hybrid Reality Display Environments [109], OptiPortals [146] and many other general or specific names.

In general, these environments provide large screen areas through multi-monitor setups, to tiled LCD displays, to seamless projection based displays. These environments are used for scientific research, single or multi-site collaboration, virtual reality applications and many other. They can be powered by various hardware and software solutions.[12]

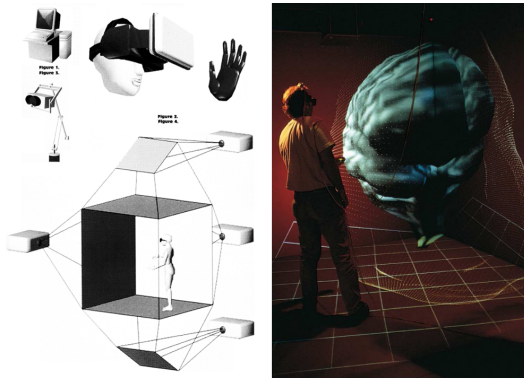


Figure 2.4: Origin of LHRDs: Cave Automatic Virtual Environment (CAVE), Architecture (left) [142], In use (right) [144].

Our interest in these visualisation environments are due to their unique properties. Briefly, these environments have unusually large display areas, where they can display many data visualisations simultaneously or display extremely large data visualisations. They can also host many spectators in the visualisation environment encouraging collaboration and potentially can enable interaction with the visualisations. As a result, these environments offer unique opportunity in the area of Visual Data Analytics, where users of

the environment can surround themselves with various data visualisations, which they can interact with, and collaborate on, to perform Visual Data Analytics. This section will discuss the hardware and properties of these visualisation environments, where the next section will discuss the software systems that power these hardware.

2.2.1 Origins

LHRDs have two origins in the literature. First origin is the large displays that are used for operational purposes in operation centres. One classical example would be NASA historic mission control center for space missions [77]. This center have many Large Displays to show various mission data and visualisations for scientists to conduct operations of space missions.

The other and more recent historic origin for Immersive LHRDs would be Audio Visual Automatic Virtual Environment (CAVE) [142]. CAVE is a cube shaped

room where each side in the cube is a projector screen. User wears a stereoscopic glasses and immerse into a 3D environment surrounding him or her. It was first conceived and designed in Electronic Visualization Laboratory (EVL) at the University of Illinois at Chicago (UIC) [144] in 1991. From these origins various high resolution and immersive displays have emerged in last two decades that we can categorize as LHRD.

2.2.2 Types

There are various types of LHRDs. We came up with a categorization according to their layout. This categorization divides these environments into 6 categories:

A. Enclosed Projection Rooms: First of these are projection based rooms. These rooms come with various shapes but more commonly known for cube shaped ones within them. They are primarily used for virtual reality purposes. They are comprised of a room with most sides and walls are projection screens. These projection screens are usually stereoscopic to give users the sense of being immersed into the environment. CAVE is the most classical example of these types of LHRDs. There are various CAVEs built around the world either following original CAVE design and name or other custom designs and names [103]. There are different versions and shapes of this type of LHRDs that are built, such as StarCaVE which is 5 sided projection room or Cornea which is powered by 24 projectors with a seamless blending of images [144].

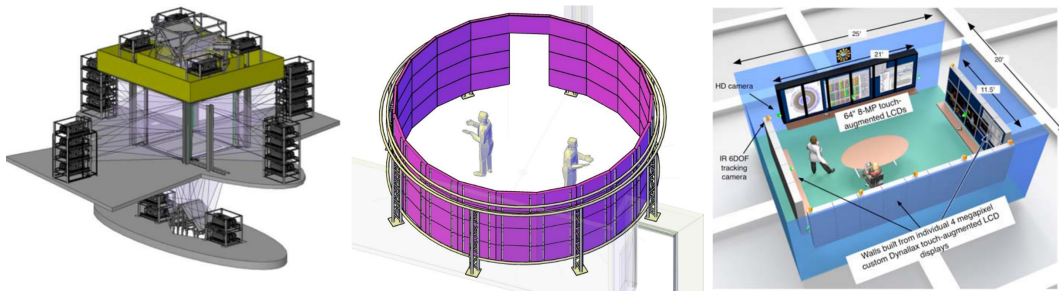


Figure 2.5: Various Types of LHRDs Architectures: Cornea Architecture (left) [144] as an example to projection based LHRDs, CAVE2 architecture (center) [29] as an example to cylindrical LHRDs and Next Generation CAVE Design (right) as an example to Enclosed Tiled Display Rooms [58].

B. Projection Based Walls: These are another LHRD type that uses projection based technology to create an immersive and interactive screen area. Instead of creating an immersive environment through surrounding users, they use many projectors in an array to create a large screen area on a screen wall. These LHRDs use the similar seamless tiling method used in the previous type in order to accurately blend images of overlapping projections [24]. There are less examples of these since the emergence of low bezel LCD tiled display walls.

C. Tiled Display Walls: Tiled display walls are large display systems created by tiling low bezel LCD screens in a grid format. There are various uses for these walls, such as science, business, art, education and many others. These can be built by smaller LCD panels such as 20-30 inch ones used in OptiPortals [133] EVL developed in the past or large low bezel 46-55 inch screens like in AESOP that uses 46 inch ultra-narrow bezel panels [92]. Most notable ones for scientific tiled display walls are OptiPortals [146]. Original OptiPortals were display walls that are composed of 4 to 70 tile, 20 to 30 inch displays. These were connected to a cluster of computers through high speed networking. They used LHRD software such as CGLX or SAGE. Tens of OptiPortals were built in the previous decade and low bezel LCD based ones are still being built in this decade.

D. Curved Tiled Displays: These tiled displays are display walls that are curved in both X and Y dimensions. They form a shape of partial sphere for immersion where the user would stand in the center of that sphere. Most notable examples of these are NexCave [92].

E. Cylindrical Tiled Display Walls: These are cylindrical tiled display environments that surround users with LCD panels in X dimension. Most notable examples are early built Varrier ones [67], and recent built CAVE2 [29] and KPMG Data Observatory [93]. These can surround users with various angles. Varrier surrounds around roughly 130 degrees while CAVE2 and KPMG DO surround users in 320 and 313 degrees respectively. While Varrier does not provide high square footage for multiple users to exist in the environment surrounded, larger ones such as CAVE2 and KPMG DO enable tens of users to be surrounded by high resolution LCD panels. Therefore, this kind of environment can enable collaborative visual analytics and research due to their unique immersive properties.

F. Enclosed Tiled Display Rooms: Last type of LHRDs is enclosed rooms with tiled display walls as sides. This is another attempt to create surrounding immersive tiled display based environment similar to last type. These rooms can also host large number of spectators and users, and surround them with immersive visualisations. Most notable example to these is Reality Deck which hosts a unique 1.5 gigapixel room with 416 screens in total of 4 walls [122].



Figure 2.6: Various Examples to LHRDs: CAVE2 (top-left)[29], Stallion (top-right) [139], Cornea (bottom-left) [144] and NexCave (bottom-right) [144].

2.2.3 Properties

LHRDs can come with various technical properties. These can vary from display technology used, to screen area, to interaction capabilities, and many others. These properties define the usability and use-cases available for that LHRD. For example, a stereoscopic LHRD can be used as an immersive VR environment; or a high resolution large user and screen area LHRD that is connected to a computing cluster can be used for collaborative data science.

A. Display Technology: First property is the display technology used. There are two main technologies that can be used for LHRDs: projectors and LCD panels.

Projector based LHRDs are usually used for immersive enclosed environments for Hybrid or Virtual Reality Applications. LCD panel based LHRDs, due to their high resolution, mostly used for data-intensive research and collaboration. Both technologies can offer stereoscopic rendering. Users can wear 3D glasses and observe the immersive and surrounding environment in 3D. In the future, LED Panel Walls can be expected to replace these technologies, but as of today their pixel per inch (PPI) ratio is too low to be used effectively.

B. Screen and User Area: Another property is screen area. This property determines the size of visualisations and number of visualisations that can be displayed in a LHRD. CAVE2, Reality Deck or KPMG DO like large screen area LHRDs can display tens of visualisations simultaneously due to having large screen areas. These LHRDs can also scale a single visualisation to unusually large resolutions. Or they can do a combination of these.

LHRDs that are designed with collaboration in mind require to have a certain area for users to roam. This is a related property to screen area. Small LHRDs and Cube rooms cannot host many collaborators, while large LHRDs such as Cylindrical, Wall and Enclosed Room ones can host many collaborators. The main challenge is providing immersion. Surrounding users creates immersion, but also reduces the usable user area. Therefore, while display walls provide large and open user areas, they lack in immersion. Rectangular shaped rooms provide more immersion in comparison, however the ideal immersion can be gained through cylindrical tiled displays as they surround the users uniformly due to having a cylindrical shape where displays are relatively at a more equal distance to users.

C. Resolution and PPI: Resolution and Pixel per Inch (PPI) affect the use-cases of LHRDs. While low resolution LHRDs can be used for purposes that are image-based such as VR, LHRDs with high resolution and PPI can be used for data visualisations which are both shape and text based in comparison to VR oriented LHRD usage. For example, in a graph visualisation with thousands of nodes, where nodes have labels; users in LHRDs with low resolutions will not be able to read those labels in contrast to the users in LHRDs with high resolutions. Same goes for spreadsheets, charts and other data based visualisations. Thus, for scientific purposes LHRDs with high resolutions and PPI are more beneficial as they provide details in large visualisations and images to be present for users.

E. Networking: Another important factor for LHRDs is their networking capabilities. Usually, these displays are powered by a cluster of computers connected on a high speed network. Furthermore, this cluster usually connects to other cloud resources in order to run its data and computing resources. There might be many configurations for how data, computing, rendering resources and displays are connected to each other. There are three areas networking can be divided. Internal network which is between rendering nodes and the server. Internal Connection to other local cloud and data resources. External connection to global internet, cloud and data sources.

Internal network capabilities affect the clustering and rendering system's functionality. The other two categories affect the access to data and processing. Many data-driven applications might require cloud data resources for processing or storage; or they may require to connect to global internet resources through APIs for streaming or retrieving data. The speed of these connections affect the overall system operation speed. For example, slow external connections on systems that require on-demand data processing or retrieval can break immersion by causing users in LHRD to wait minutes for visualisations to load.

F. Clustering: Another aspect is how these rendering nodes are clustered. There is two conventional clustering methods for LHRDs: Master-Slave and Client-Server [12]:

- Client-Server: A server runs management logic and applications' core logic and in charge of coordination of clients, and client nodes form the cluster of the system which perform rendering and if required certain amount of processing.
- Master-Slave: On the other hand, in master and slave configurations every node is a client. Each node runs the application but master node runs users interaction in addition to the application and they have a serverless model.

As a result, the difference is in master-slave configuration the master is also a client running management logic in comparison to the client-server model where server does not provide any client functionality. There can also be other modes in between these two. Clustering also affects and related to which kind of parallel rendering is performed.

G. Rendering: Rendering happens on rendering nodes. These are computers that are connected to displays in an LHRD. These computers are either directly connected to displays through their graphics card or cards, or use other specialized hardware for connecting many screens at once. These specialized hardware enable a powerful rendering node to connect to many displays in a LHRD at once. On the contrary, conventional rendering nodes are hosted close to displays, they power and use single or many graphics cards to connect to displays. Depending on software solution used, these nodes may be required just to render visualisation. Although, they can be used for their CPU or GPU processing in addition to rendering in some cases. There are 3 ways parallel rendering is performed [10] in general: Sort-First, Sort-Middle and Sort-Last; and use of these in LHRDs result differently:

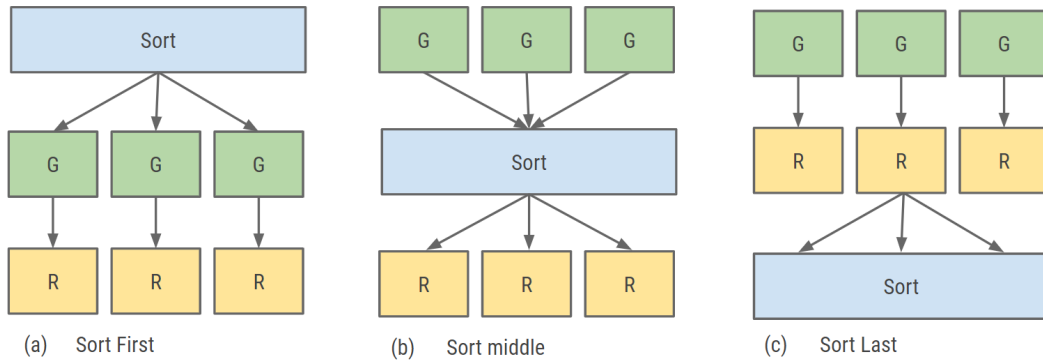


Figure 2.7: Parallel Rendering Mechanisms (G:Geometry R:Rasterization) [10][135]

- **Sort-First:** in this mode, graphic primitives are sorted and distributed to tiles before geometry processing or rasterization. This is natural for render clusters which are tied with displays. Server or cloud sends graphic primitives to render nodes and each render node renders their part of the screen space.
- **Sort-Middle:** in this mode, geometry processing happens on cluster and afterwards sorted to rendering nodes where rasterization happens. Thus, sorting happens in the middle. As a result, a server or cluster performs certain geometry and other processing, sorts the data, distributes and it gets rasterized on rendering nodes.
- **Sort-Last:** in this mode, geometry and rasterization happens in parallel but the sorting happens at last. thus, the cluster or server performs all rendering

and finally pixels or frames are sorted and transferred to distributed displays.

For clusters that are tied with displays directly, sort-first is the native mode to use. A server knows which node will render which section of the tiled display, distributes primitives as part of sort process and each node does local rendering. In other two modes, exchange of directives or resulting renders are required as part sorting. Server requires to distribute primitives to cluster unsorted, cluster processes either geometry or both geometry and rasterization, and performs a distributed sort. This requires one communication exchange between server and clients and one communication exchange between clients in comparison to first mode where there is only one communication exchange.

There are other variants used in LHRDs. For example, there is a Sort-First variant where instead of sorting directives, data is being sort and nodes perform local rendering with given data and view (2D/3D). Another example is Sort-Last variant without a true parallel processing. In this variant, server processes rendering either on CPU or GPU resources, parallel or single or multi-thread, and then distributes pixels/frames to nodes through sorting, which is commonly used for distribution of video based content.

Finally, remote rendering or in other words cloud based rendering [11] is another solution that could be used in LHRDs. In this type of rendering, cloud hosts GPU resources and executes different stages of rendering on the cloud, then distributes the rendering to cluster of nodes. There are several levels this can be performed. These levels are based on which 3D data is processed on cloud and which 3D is processed on clients as post-processing and in which order. In order to utilize this for LHRDs, a powerful GPU cloud should be available for this purpose.

As a result, different local and remote rendering and clustering mechanisms are available for LHRDs, which should be selected depending on the use-case.

H. Resources: LHRDs are usually connected to cloud resources on local or external clouds. These can be storage, processing, parallel graphics, CUDA based AI applications, Data streams and many other. These resources enable different uses and applications for these environments. Of course their use is also based on the software the LHRD is using and its applications.

Various applications built on LHRD can utilize these resources for functioning.

<i>continued from previous page</i>						
HIPerSpace	286	21.7	70	Wall (14x5)	LCD	[158] [150]
HIPerWall	204	18.6	50	Wall (10x5)	LCD	[123]
NASA HyperWall	167	21.3	128	Wall (16x8)	LCD	[104]
KPMG DO	130	37.2	64	Cylindrical (313°)	LCD	[93]
Varrier (CV-65)	124	13	65	Cylindrical (132°)	LCD	[67]
Lambda Vision	100	17	55	Wall (11x5)	LCD	[35] [150]
Cornea (C6)	100	55.7	24	Room (Cube)	Projector	[144] [141]
CAVE2	74	41.8	72	Cylindrical (320°)	LCD	[29]
StarCave	68	37.9	15	Room (5 sided)	Projector	[144]
Varrier (CV-35)	67	7	35	Cylindrical (132°)	LCD	[67]
NexCave (KAUST)	43	12.2	21	Wall (Curved)	LCD	[92]
AESOP	42	23.2	40	Wall (10x5)	LCD	[92] [150]
VuePod	25	10	12	Wall (Curved)	LCD	[103]
REVE	18	4.4	9	Wall (3x3)	LCD	[6]
CAVE	5.2	55.7	6	Room (Cube)	Projector	[29]

2.2.5 KPMG Data Observatory

One of the LHRDs in the previous list has a significance to us, since the GDO Software Framework is implemented for it. This is KPMG Data Observatory in Data Science Institute, Imperial College London [93]. It was built in 2015 and was the largest of its kind in Europe as of today. It features 132.7 million pixel resolution on 64 screen circular wall with 313 degrees of surround vision. It uses Samsung UD46D-P ultra narrow bezel professional video wall screens. It is powered by 32 rendering nodes. It has 16x4 layout with 2.53m height and 6 meter diameter. It is equipped with surround sound, distributed motion sensing and many other interaction devices. It has a large touch screen console for collaborative user input. It runs in 2 modes: one is clustered and the other is sectioned. In clustered mode 32 computer nodes power 64 screens. In section mode, 5 super nodes power 5 sections of screens varying between 12-16 screens per node. It was designed, built and housed by Data Science Institute at Imperial College London. It is home to many research projects in the Imperial College London.

2.3 LHRD Software and Their Evolution

LHRDs are powered by various software systems around the world. The first software for LHRDs is proprietary CaveLib that run original CAVE system [30]. We have divided these software to 3 generations. First generation is the legacy generation that used MPI and OpenMP like clustering technologies for powering clustered displays. Second generation focused on render distribution and streaming to LHRD screens. There is a sub generation between 2nd and 3rd. This 2.5 generation focuses on render distribution, built on legacy technologies but has 3rd generation properties such as heterogeneous visualisations and applications. Finally, the third generation of LHRD software focus on web technologies, interaction, app orientation and collaboration with only one example SAGE2.

2.3.1 1st Generation LHRD Software

First generation was designed and implemented in an era where there were constraints in every area of resources such as storage, processing power, networking, screen resolution and many more. They used legacy technologies as their building blocks while using legacy methods to overcome that era's problems. The main

paradigm of that era was to use MPI based high performance clusters to power systems like these. These clusters were using OpenGL or other rendering mechanisms for distributing renders to multiple screens. Another objective of this era was to find solution for collaboration between different sites that these environments were built. In order to achieve this, there were both software and hardware attempts to share large volume of data and information between sites around the world.

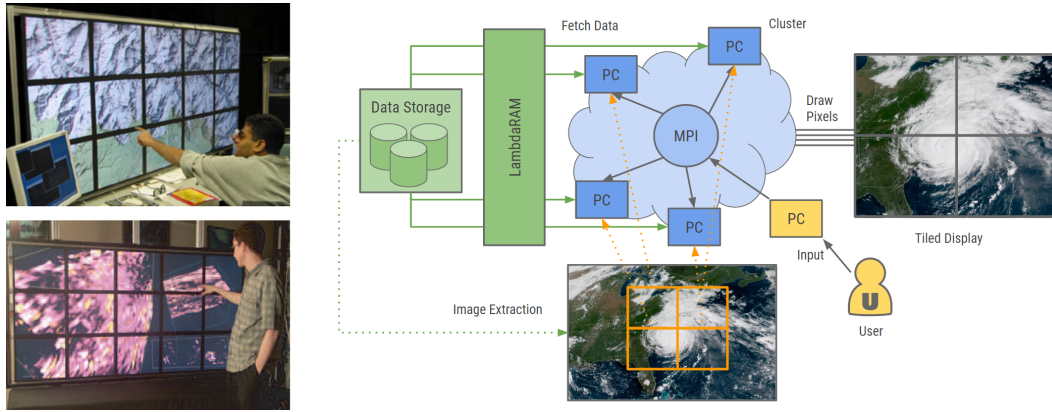


Figure 2.8: First Generation Examples: JuxtaView (top-left) [90] Vol-a-Tile (bottom-left) [166] and JuxtaView Infrastructure (right) [91].

A. CaveLib: CaveLib [30] [4] [27] is a proprietary closed source library that was developed for CAVE system. It abstracts applications of CAVE from the system and manages display process such as handling distributed perspective or issuing synced swapbuffer commands. These applications use various CAVE provided callback functions to both render and access to CAVE specific hardware such as sound system or wands to operate. It is an advanced system compared to its time and can be seen as origin to other EVL libraries such as SAGE.

B. JuxtaView: JuxtaView [90] is a first generation LHRD software created to power distributed displays like GeoWall2. It retrieves data from data storages and pushes it to a distributed shared memory infrastructure called LambdaRAM (which is built on QUANTA networking API). LambdaRAM then feeds this to MPI which renders as OpenGL on screens. It has been used by US Geological Survey (USGS) in the past for geographical data visualisations and The National Center for Medical Imaging and Research (NCMIR) for medical visualisations[91].

C. Vol-a-Tile: Vol-a-Tile [166] is a parallel application that uses MPI, QUANTA and OpenGL to render large volumetric data on distributed tiled displays. It receives data through QUANTA from OptiStore and Vol-a-Tile slaves and master using MPI coordinates the visualisation using OpenGL with synced frame swap.

D. AURA: AURA [124] is a parallel rendering software for OpenSceneGraph [145]. It can be deployed in multiple instances and thus enable parallel rendering of OpenGL on LHRDs. It is similar to early versions of WireGL, however uses MPI thus it can be categorized in 1st generation. It has two modes: Multiple Copies and Broadcast. In multiple copies mode each node has a copy of the applications and perform local render. In broadcast mode, only master runs applications and graphic commands are distributed to nodes and transformed into OpenGL renders.

E. OptIPuter: OptIPuter [147] [13] is a conceptual hardware infrastructure for implementing many to many Mux/deMux photonic switch hardware. These devices designed to be used to create an optical network that can use various sources like data and visualisation clusters, and distribute data and graphic streams coming from these sources to compute and render clients and thus displays across the network. It is a specialized hardware and middleware solution to replace existing networking between clients of LHRDs and servers, data sources and compute cluster of LHRDs by easy and fast switching and multicasting.

F. LambdaBridge: LambdaBridge [94] is an experimental architecture that uses optical switches to create data and graphic IO network for large areas. It was envisioned that LHRD research hubs on different cities can send data and graphic IO between each other using optical network composed of optical switches and specialized hardware on nodes (research hubs) to decode these. These systems can also be used for optical multicasting which can scale images to tiled displays. Today, this type of hardware is widely obsolete as general networking solutions are sufficient to perform the objectives of this architecture.

2.3.2 2nd Generation LHRD Software

Second Generation LHRD software were designed to distribute rendering of 2D/3D content mostly based on OpenGL. These libraries were not encumbered by constraints on resources as much as the first generation libraries and thus dismissed

use of legacy technologies such as MPI and instead used conventional networking. They used various OpenGL and other graphic libraries and expanded them to distribute rendering in an LHRD using different methods.



Figure 2.9: Second Generation Examples: Garuda (left) [71] Equalizer (center) [59] and CGLX (right) [32].

A. Distributed OpenGL and Virtual Display Driver: Princeton University for their Scalable Display Wall project made a display wall out of 8 projectors behind a screen. This system aligns projectors and performs parallel rendering where projector images are optically blended. They developed 2 libraries for this wall: a virtual display driver on Windows2000 for distributed desktop and Distributed OpenGL driver for running OpenGL based applications such as classic games of that era. In order to power these 8 projectors, a display cluster, computer cluster, input cluster, a storage server and a sound server was used. [24]

B. TeraVision: TeraVision [140] is many to many video stream software designed for LHRDs. Graphics rendered in cluster using TeraVision servers and clients streamed to displays and LHRDs over TCP and UDP. It uses hardware aid to scale to Tiled-Displays called VBoxes.

C. DMX: Distributed Multihead X Project [53] is a project to distribute X windowing system to multiple displays that mainly used unix based operating systems. It works through setting a single X server as a proxy to multiple X servers that render on multiple displays. Through this a single continuous desktop can be drawn on multiple displays. It is a broadcast based system where all X11 request and OpenGL request are sent to X11 server nodes to be rendered. This system can be run together with Chromium too. Although, this system has scalability issues with large amounts of displays [2].

D. CGLX: Cross Platform Cluster Graphics Library [32] is an OpenGL distribution based second generation LHRD software. It was developed for X-based operating systems and utilized GLX for direct rendering (AGL or Carbon in Mac OS X). It features programming interface for hardware-accelerated rendering of OpenGL visualisations on distributed display clusters. It works through providing a unified and scalable distributed OpenGL context for users which modifies OpenGL directives for distributed rendering. It uses Master-Slave architecture.

In CGLX, users interact with an application on the master node and generated graphics events are distributed to slave nodes and rendered in sync. For allowing multiple OpenGL graphics, CGLX has "World Concept" which allows multiple simulation worlds to run on different parts of the tiled display. It also comes with a program to manage nodes and servers of tiled-displays for using CGLX.

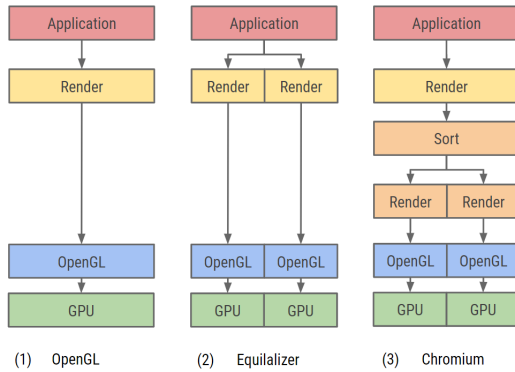


Figure 2.10: OpenGL (left), Equalizer (center), Chromium (right) architecture [60] as an example to second generation architectures.

E. Equalizer: Equalizer [60] [59] is toolkit for OpenGL scalable parallel rendering for LHRDs. It is written in C++. It uses client-server based clustering. It performs sort-first or sort-last parallel rendering. It has a hierarchical clustering where the system operates through nodes (machines), pipes (graphic cards), windows (display segment on screens), channels (rendering area on a window). It uses a compound paradigm where compounds are data structures to describe parallel rendering and compound trees are logical

description of rendering pipeline. For example, a tile compound describe in unit how tiles should be rendered in the screen-space, a 2D compound describes how unit 2D content should be distributed and rendered parallel, Pixel compound does this for pixel to pixel. There are various compounds for enabling a catalogue for different kind of visuals to be rendered in parallel. Compound trees utilize this to create complex logical rendering pipelines with decomposition and re-composition. This framework does not send OpenGL commands over the network. Instead all

rendering is executed on OpenGL context where rendering tasks are sent over the network. Moreover, Equalizer allows application development. Developers can create C++ based applications that render on OpenGL. These applications have to base on the provided C++ classes from Equalizer such as node, pipe, window, channel to utilize Equalizer.

F. WireGL: WireGL [172] is a solution for scalable interactive rendering on a cluster of machines. Its primary purpose is to enable parallel rendering to a single display. However, in networking mode, it can render from multiple applications, on multiple renderers to multiple displays. It conducts sort-first parallel rendering using OpenGL API. It is comprised of application and rendering nodes. Clients submit OpenGL commands in parallel to distributed renderers called pipeservers which performs the parallel rendering. Through an image composition network, these parallel renderers can serve images to multiple displays in a LHRD. This is done through partitioning images that are designated to be displayed in LHRD into tiles on pipeserver. Each pipeserver can manage multiple tiles. Final image will be composed in parallel through image composition network. This pipeline will stream OpenGL commands into renders from multiple applications to multiple renders to multiple displays. The development of this project is currently suspended and it has been rolled into Chromium.

G. Chromium: Chromium [34] [2] is another OpenGL distributed virtual graphics pipeline (not to confused by web browser by the same name). It works in three modes: One to many, many to many and many to one, applications to renderers. In chromium OpenGL applications can run unmodified. It uses a client-server model and enable sort-first and sort-last parallel rendering options. It is successor of WireGL as mentioned.

H. HiReD: HiReD [76] is a high-resolution multi-window visualisation environment for cluster-driven Displays. It is developed in C++ and uses OpenGL for visualisations and VRJuggler[168] for middleware. It utilizes VNC for remote control of the windows. It is not open-source. It is another conventional example of window based second generation LHRD software.

I. Garuda: Garuda [71] is client-server based LHRD software that distributes to client nodes geometry of visible objects through multi-casting. It can run OpenScen-

Graph (OSG) [145] based applications without any modification as Garuda provides underlying system for distribution. In Garuda, server upon receiving a new view-point determines visible objects in the scene and send visibility information to nodes. Nodes determine if the objects are still in the cache or not and report back to server. Server then sends missing objects through serialization, form scene-graphs and multicast them to clients. Clients upon receiving this wait for swap order from the server and when server broadcasts a swap order, they swap buffers to change frame.

J. CalVR: CalVR [26] [25] is an OSG based VR software designed for utilizing clustered GPUs for 3D VR use. Although, it also supports displaying OpenGL renders on tiled display walls. It is using a similar rendering distribution to WireGL since WireGL is also designed initially for utilizing clustered GPUs to display on a single screen. In design it has equivalents to pipeservers. Its programming language is C++ and it supports plugins.

K. Infiniviz: Infiniviz [84] is a virtual machine based high resolution display software for tiled displays. It renders desktops of operating systems run in virtual machines. It distributes these desktops to virtual GPUs and these monitors send frames to a frame buffer manager which encodes them with H.264. The manager sends encoded frames over the network to display nodes of the tiled display which renders these frames after decoding. It enables virtual desktops to be distributed and rendered on tiled displays.

L. IceT: Image Composition Engine for Tiles [78] [138] is a system that performs sort last parallel rendering to compose images for LHRDs. It is written in C. It is designed to overcome the problems of software that uses sort-first approach which renders frames at higher rates but has bottleneck on input for large data sets such as WireGL. IceT has ParaView support, which enables scientific visualisations to be visualised using IceT or system that enable IceT support.

2.3.3 2.5 Generation LHRD Software

Previous (2nd) generation LHRD software primarily focused on render distribution to LHRDs. However, they lacked application development or multi-user interaction or collaboration support which 3rd generation focuses on. There are few platforms that provides such functionality without the new technologies and

paradigms used in the 3rd generation such as web technologies. These platforms were ahead of their time and provided advanced functionality using conventional technologies. Therefore, we will denote these software as 2.5nd generation. There are three main examples to these platforms: DisplayCluster, OmegaLib and SAGE.

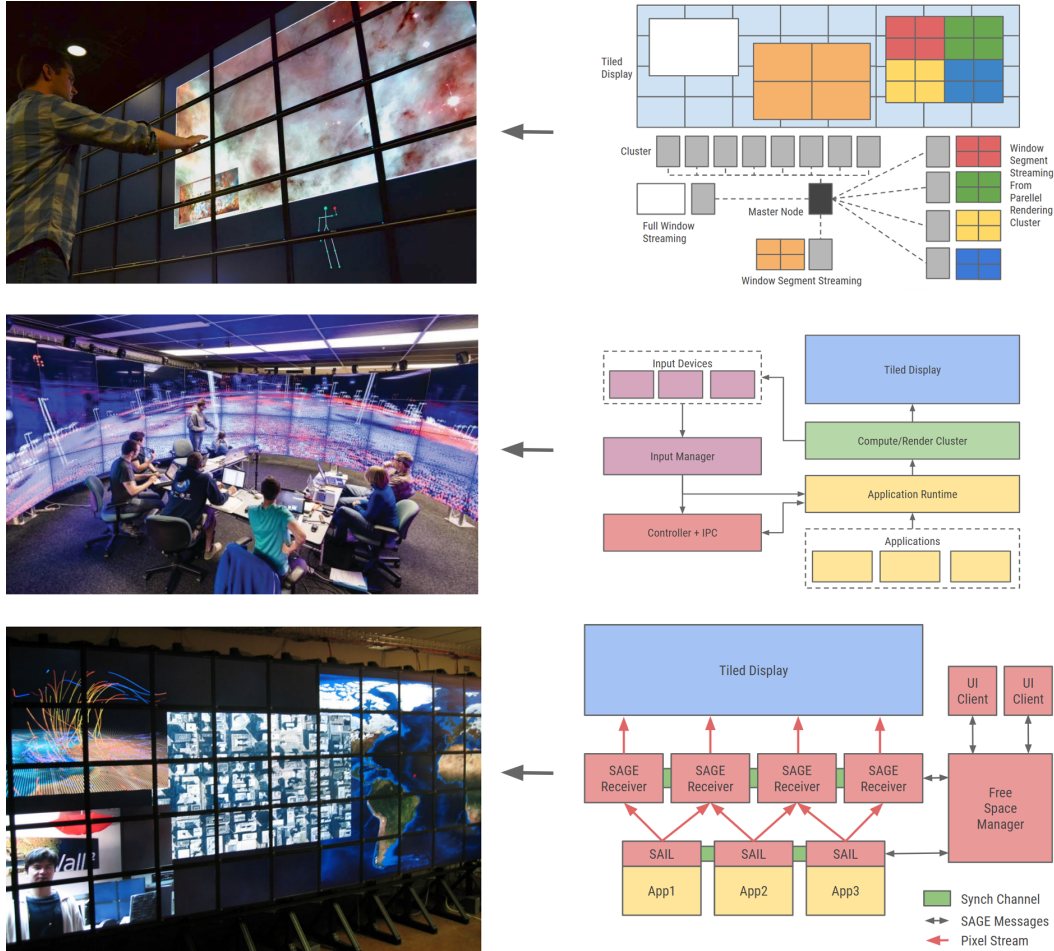


Figure 2.11: 2.5 Generation Examples: DisplayCluster (top) [51] OmegaLib(middle) [109] and SAGE (bottom) [73]

A. DisplayCluster: DisplayCluster [51] [50] is a cross-platform multi-user interaction oriented environment for LHRDs. It provides a virtual desktop for High-res imagery, desktop streams, videos and parallel 3D rendering. It can connect to gamepads, smart phones, touch screens, kinect and other interaction for multi-user

interaction. It is window based and users can summon various applications anywhere on the LHRD. It uses master-slave clustering. It is built on C++ and uses python for scripting. It supports IceT applications, which can link to DisplayCluster and stream directly, thus indirectly supporting ParaView visualisations.

B. OmegaLib: OmegaLib [109] [108] is a application framework for LHRDs. It uses Equalizer for rendering pipeline and focuses on application support. It support both 2D and 3D application development (OpenGL). Applications have replicated execution on nodes that power the tiles they are on. If the application is not on that part of the screen-space it does not run, or if it is on background, it runs just on CPU. OmegaLib is built on C++. It uses a master slave clustering. It has a custom template for applications which are developed on C++. It uses message passing communication system to manage execution of applications such as starting or stopping instances of them. Applications modules also use the same system to communicate with each other. As a result, OmegaLib applications can run on a distributed manner.

C. SAGE: Scalable Adaptive Graphics Middleware [125][73] is a software system for LHRD based collaboration and visualisation. It is a window based LHRD software that is written in C++. It uses screen-space as a canvas and users interact with the screen space directly through a pointer application. It has a file server where users can place various file types such as audio, images, office documents, videos and 3D apps. Users than use SAGE apps to display these files. It uses three main libraries: one for streaming pixels (libsail), one for streaming audio (libsac) and one for networking (QUANTA), which is used and evolved from OptiPuter. It has an application interface called SAIL: SAGE Application Interface Library. Apps using SAIL stream pixels to SAGE receivers (distributed) that renders the apps on LHRD. Sage Receivers and SAILs are synced for this process. They also communicate with the window manager for organizing the distributed continuous desktop. This manager is also connected with UI clients which users can utilize to open apps, move or resize windows.

2.3.4 3rd Generation LHRD Software (Current)

Third generation LHRD Software development, current one, is focused on web based and app oriented platforms. This generation aims to provide scalable, generic

web based software. It intends to enable rapid development of applications through using web technologies and thus provide versatility. There are only one example to this generation: SAGE2 which is developed in EVL.

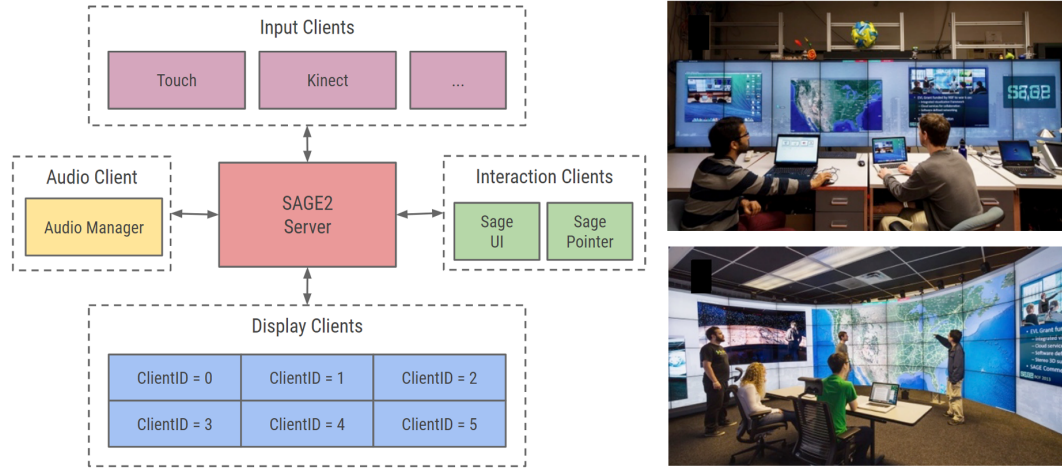


Figure 2.12: SAGE2 Architecture (left) and Examples to SAGE2 uses in different display configurations (right) [129]

A. SAGE2: Scalable Amplified Group Environment [129] [128] [126] is a local and remote collaboration and visualisation software for LHRDs. It originates from SAGE and other EVL software developed over years. It provides similar functionality to SAGE, although it uses web technologies and javascript instead of C++. In SAGE2, LHRD screen nodes are run by web based display clients that can run on browsers. These clients are connected to a SAGE2 server which is a node.js based javascript server. These clients render visualisations through various mechanisms. Common content are rendered through using HTML, videos are pixel streamed in YUV blocks and for 3D WebGL is utilized. There are also interaction and audio clients and other input servers that connect to this server. It uses web sockets to communicate between clients and server. SAGE2 is window based and similar to SAGE in use. Users can open application instances using UI page and on-screen virtual desktop. They organize and control these applications through an on-screen pointer called SAGE Pointer.

SAGE2 also provides an application development framework alongside an app-store which is hosted in their webpage [126]. These applications are developed in

javascript and utilize SAGE2 rules and API to render on LHRDs. For this to happen, SAGE2 provides API and context for rendering various HTML based and other typed content. Developers using javascript program how canvas provided by SAGE2 is utilized. In SAGE2, developers are not given direct access to HTML content on nodes and have to use SAGE2 context to manipulate given canvas. Although, a workaround is provided by SAGE2: which is webview based applications, that enables running web pages distributedly in SAGE2. Thus, developers can create a visualisation web page and try to run it in SAGE2, although there might be scaling and interaction issues with complex web pages. Furthermore, instead of rapid development, developers can choose to implement complex applications within SAGE2. They can utilize various external libraries or non-native language and create complex, multi-user and interactive applications.

There are various applications developed for SAGE2. These include unity demos, to image browsers, to ParaView[116] integration for scientific visualisations[75], maps applications to charts applications and many others. These applications allow, users to deploy various visualisation content on the virtual on-screen canvas/desktop SAGE2 provides. Furthermore, they can use the SAGE2 UI to have an on-screen pointer, share their screen as a window, load multi-media content as a window, arrange the windows on the canvas or configure settings. There can be multiple users doing this simultaneously.

As a result, SAGE2 is a good example of 3rd generation LHRDs software. It provides various content to be visualised, has app development, is web-based and allows multi-user interaction. Thus, it provides a collaborative environment for LHRDs.

2.4 Critique of LHRDs Software Platforms for Visual Data Analytics

In summary, we have explored three research areas in this chapter. First of these research areas is Data-Driven Multi-disciplinary Research Projects that are trending with emergence of Big Data and Data Science. These projects utilize the increase in data-availability in last decade and try to find new solutions through collaboration of cross-field teams. Imperial College Data Science Institute and other similar newly built research centres aims to target these multi-disciplinary research

as an objective.

Second area we explored is Visual Data Analytics. This area intends to help research by providing data analytics through visualisations. This area, which we see as one part of the solution to data-driven multi-disciplinary research projects, also evolved in last decades. It now uses web-based, interactive, app oriented, on-demand, cloud hosted visualisations and analytic methods to conduct visual data analytics.

Finally, the third area we explored is Large High Resolution Display Environments (LHRDs). This is the second part of the solution to aid data-driven multi-disciplinary research projects. These are multi-million pixel LCD tiled displays or projectors that are used in research centres as visualisation environments. The software powering these environments have few generations. Older generations are mostly obsolete in technology and methods. Over the years the research in the area have slowed down and only few examples stand still on later generations. Unfortunately, these examples also shifted their focus to productivity and collaboration.

Combining these areas, we set our objective to enable performing Immersive and Social Visual Data Analytics and Data Science on Data-Driven Multi-Disciplinary Research utilizing collaboration and unique display properties of LHRDs. This requires existing techniques of modern visual data analytics to be integrated into LHRD environment and then extension of these techniques to further utilize LHRDs unique properties for given purpose. We also want to utilize existing research on LHRDs software as well.

In order to do this, we have explored design landscape for a Visual Data Analytics software on LHRDs and identified the challenges it has to solve. This is discussed in length in 3rd chapter, since it is a new and novel approach. Here, we will briefly introduce these, so we can evaluate the existing software on these requirements/challenges and justify our decision to take a new and novel approach to enable Immersive and Social Visual Data Analytics on LHRDs for given objective.

We can divide these challenges into 4 aspects, these are:

1. **Collaborative Use of LHRDs in Visual Data Analytics:** LHRDs should be utilized in various types of sessions with users with different roles for Visual Data Analytics. Furthermore, they should be able to enable various use-cases

such as: Data Exploration, Collaborative Analytics, Design, Simulation, Operation and Control, Research and Development, Presentation, etc. The requirements of this aspect is very diverse. In summary, LHRD software should have app deployment; layout, interaction and control capabilities; management functionality focusing on visual data analytics; presentation utilities or capabilities; distributed debugging utilities; flexible setup for various deployment options; external library freedom for versatile app development for satisfying designated roles; session types and use-cases in general.

2. Immersive Display and Management of Data Visualisations on LHRDs:

LHRD platform should provide various display, management and interaction capabilities for Visual Data Analytics purpose. They should be able to create unified display area out of distributed display nodes, enable deployment of visualisations on this area. They should employ management mechanisms for visualisation deployment, setting layout of visualisations, saving and restoring sessions and providing user interfaces for management of the system. They should be flexible on display technology and setup, and enable remote deployment.

3. Data Visualisation Apps and their Development on LHRDs:

LHRD software should allow development of data visualisation apps. These apps should support a certain list of properties. Most significantly, these apps should work as plugins and can be instanced. They should be deployable with various data and configurations. Users should be able to interact/steer/control or configure these applications. They should be able scale to any section of nodes on LHRD. They should be able to utilize various internal or external libraries, data or APIs. Their front-end should be web-based and rapidly developed. They should follow certain standards so global utilities can work over them. They should be distributed and API-based and thus they can coordinate and have distributed operation.

4. Data-Driven, API based and Collaged Data Visualisations on LHRDs:

LHRD system should connect and utilize many data sources, enable many data processing and distribution mechanism. It should also enable programming functionality over dynamic distributed components over API on the system thus users or developers can script complex session types. Finally, system

should allow syncing deployed data visualisations on certain parameters and enable creation of dynamic and interactive data scenes that are composed by collaging and syncing heterogeneous data visualisations.

When we evaluate existing LHRD software on these key aspects, we discover that they show significant lack of capability in this type of use.

A. 1st and 2nd Generations: The first and second generations of LHRD software, only able to satisfy certain items on first and second aspects and lack on other two aspects completely. These generations usually can create unified screen-space and enable distributed rendering where a large visualisation can be scaled. However, they do not provide app-orientation and hence enable implementation or integration of general data visualisation apps, do not provide mechanisms for interacting with visualisations, do not provide mechanisms for management of data visualisations or creating layouts. The visualisations that can be used with these generations usually have to be implemented in OpenGL or other libraries which are not developer friendly and in some cases they should be implemented part of system code and not as plug-ins. There are no mechanisms for data-driven visualisations, linking visualisations or logic, or collaging synced visualisations. Finally, they are based on or use obsolete technology or methods, thus cause compatibility problems with more recent resources and libraries that developers may want utilize. Therefore, we can conclude that they are highly incompatible with this purpose.

B. 2.5 Generation: In the 2.5 Generation, we have DisplayCluster, OmegaLib and SAGE. These platforms again satisfy certain requirements in first and second aspects, similar to 1st and 2nd generations, as they can create a single display area out of distributed display nodes and offer deployment of visualisations. What is improved from first and second generation is layout options as these can create visualisation scenes in many layouts like a desktop and offer system level interaction such as on-screen control. However, management of the system do not take data visualisations or visual data analytics into consideration. So, they do not have configuration, control or interaction over applications, saving or restoring sessions. On the third aspect, even though they provide app development, these platforms fail on many properties. Most significantly, they fail on providing apps that are general, interactive, distributed, web-based, collagable, data-driven, API-based; where these

properties are essential to data visualisation apps that are to be used in visual data analytics. In all 3 of these platforms, applications are developed in C++ and therefore they are not very developer friendly and also not enable rapid development as much as web-based technologies. Finally, for the 4th aspect, they completely lack on enabling data-driven, API-based, collaged or synced applications.

C. 3rd Generation: In the 3rd Generation, the only platform is SAGE2. This platform satisfies certain requirements of the first and second aspects. It creates a unified display area out of web pages and allow deployment of visualisations as windows to the display area. It contains many management functionality mostly focused on on-screen control of visualisations. It is more flexible on display technology and setup than 2.5 generation since it uses web technologies. However, management functionality is again oriented around desktop like use and not visual data analytics. Applications are interacted on screen-space and in a very limited fashion. Another problem is session management, where it is limited with sage2 as users can not save, load and program sessions containing multiple apps. However, we can also see that this generation has major improvement in collaboration in terms of roles and session in general, but still lack in satisfying use-cases that we target.

SAGE2 allows more properties than the 2.5 generation on the 3rd aspect. It allows application development and has library of applications. Although most of these applications are collaboration and productivity oriented instead of data visualisations. These applications are scalable to screen-space, although SAGE2 limits scalability of applications, such as resizing them, so they come in given sizes. Still, developers can implement applications that can span to most of the displays. This pre-defined scaling approach creates problems on creating collaged data visualisations in layouts, where users may want to create different layouts out of data visualisations. Another problem regarding app scaling on SAGE2 is it uses pseudo scaling and distribution. SAGE2 application runs in copy on all nodes with very limited coordination. As a result, they do not run in a distributed manner where they coordinate properly. They only differentiate on parameters. For example, an image is deployed to SAGE2 screens, every node displaying the part of the image loads the whole image and zooms to corresponding part. If there are 64 nodes displaying the image, that means each node loads 64 times more data or resources than they need to and thus creating a potentially disruptive overhead, which can slow the

system to a point where browsers on the display nodes no longer respond.

In terms of interactivity and generality, SAGE2 applications show limited interaction options as they can only be interacted through on-screen. SAGE2 interaction focuses on desktop style use. This creates problems in data analytics presentations as users cannot control and interact with applications through tablets, but instead deploy them through tablets and interact with them through virtual pointers. Therefore, SAGE2 applications cannot be configured/controlled or steered the way our use-scenario intends. In terms of generality, these applications are not designed to be general data visualisations. Their system paradigm do not support configurations, templates or data selection for applications, furthermore their distributed app paradigm have limitations on dynamic distributed operation due to their serverless app design, thus it is not possible to provide different data and configuration to app logics on the displays nodes without major overhaul on the system, thus seriously limiting implementation of general data visualisations.

As a result, we can conclude that none of the existing solutions are compatible to be used in immersive and social visual data analytics on LHRDs for data-driven multi-disciplinary research projects including only third generation software SAGE2. Therefore, in the next chapter, we will explore in depth the design issues and challenges for a novel approach for this objective.

Chapter 3

Design Issues and Challenges

In the previous chapter, we have surveyed existing software platforms on LHRDs. We have concluded that performing modern visual data analytics bares significant problems on existing platforms for data-driven multi-disciplinary research projects. As a result of this, this thesis took the task of searching for a new software approach for enabling immersive and social visual data analytics on LHRDs for this objective. But first of all, this approach should identify which capabilities, features and properties should be suitable for a platform with that orientation. Therefore, the purpose of this chapter is to layout design landscape and issues for a software framework for performing immersive and social visual data analytics on LHRDs for multi-disciplinary data-driven research. Furthermore, it should identify technical challenges that needs to be solved which these findings impose on this platform.

In order to do this, we divided this landscape into several aspects. This chapter will go through these aspects to identify design issues and challenges for implementing a platform for the given task. Afterwards, in the final section of this chapter, we will transform findings of these aspects into a list of technical challenges that needs to be addressed. These aspects are:

- **Collaborative Use of LHRDs in Visual Data Analytics:** Explores different types of users, session and use-cases.
- **Immersive Display and Management of Data Visualisations on LHRDs:** Explores various display, management and interaction functionality for this purpose.

- **Data Visualisation Apps and their Development on LHRDs:** explores app-orientation of the system, its development and properties of applications.
- **Data-Driven, API-based and Collaged Data Visualisations on LHRDs:** explores data-driven property, API based distributed operation and scripting, syncing and collaging visualisations.

3.1 Collaborative Use of LHRDs in Visual Data Analytics

LHRDs potentially can provide an environment where research teams are surrounded by data visualisations, which they can interact to conduct visual data analytics. Therefore, they can use this environment as a medium to perform discussions, decision making, analytics and other collaborative activities. It enables the members of the research team to communicate and work through data visualisations. This considerably new way of communication and work style increases collective understanding of data, ideas, concepts, logic and information on the research subject. Considering most of the research are performed by cross-field teams in recent years, this becomes significantly beneficial, since the collective understanding of each other's field to a degree is required during working in a cross-field project. Thus, these environments can be used as a hub, where cross-field projects' analytics and discussions are performed through on-demand interactive data visualisations. In this aspect, we would be identifying user and activity types, session types and potential use-cases of these visualisation environments under visual data analytics use. The purpose of this section is to align the software framework's features and properties to this targeted type of use and extract requirements by this more abstract aspect, in comparison to other aspects in this chapter which are more concrete.

3.1.1 Roles on LHRDs

First topic is the user and activity types. This topic will help us understand what kind of interactivity would be needed in a LHRD platform for visual data analytics. During our experiences in KPMG DO, we have encountered with mainly 4 types of user roles. These are Administrator, User, Spectator and Developer.

A. Administrator: In our experiences, administrator should be a super-user that has access to management functionality of the environment. These include system

settings, configuration, visual layout of apps, casting functionality and other utilities. Admins are expected to be tasked to perform management activities on the environment such as preparation and execution of sessions through setting overall configuration of the visualisation environment. Furthermore, they are expected to be tasked controlling the environment during sessions and supervising the session both digitally and physically by administrating spectators and users. As a result, they would need administration interface to control and manage the system. This interface should have all the functionality mentioned above and moreover it should be available on a mobile device where they can carry such as a tablet, so admin can also administer spectators and users on foot while managing the system.

B. User: User is an entity that interacts with visualisations on the LHRD. Users should be able to deploy and control visualisations through interfaces on laptops or tablets or other interaction devices. They should have a control interface made available to them which can be used to change visual layout of apps, launch apps, control parameters of apps, deploy data to apps or steer visualisations. Users through these activities can perform data exploration, presentations and collaboration based sessions. In these sessions, user can summon up various visualisations and data to express themselves or back their ideas up through visualisations. As a result, users should made available with a similar interface to admins. In addition, this interface should enable control, configuration and steering of visualisations.

C. Spectator: In contrast to other visual environments, spectators benefit from interactive and coordinated approach of the environment in addition to its size which can be configured to deploy various layouts with tens of visualisations on the display area. This gives a new perspective to spectators as they can both see a larger picture on data provided by visualisations and go near the visualisations and see finer details on data on large visualisations, or compare many different aspects of data on multiple visualisations. Spectators in the environment can host a discussion related to visuals. Being surrounded with related visualisations can enhance the experience of that discussion and act as a base to the discussion that is being hosted. Existence of multiple visualisations or interactive visualisations can be used to enrich even further the discussion's effectiveness.

However, in data analytics orientation, use of screen-space should be different than other examples on the literature. It should not be in a desktop style where

visualisations represented as windows and moved around. First of all this creates nausea on users in a large environment from our experiences, and secondly it reduces efficiency on usable screen-space. Since the purpose is visual data analytics, instead of windowing the focus should be on a dynamic collage of visualisations like a dashboard, which enables users to compare and contrast. Thus, this mechanism will enable more efficient use of screen-space.

Moreover, spectators or users in the environment can be subject to research. This is performed by reading data of spectators through various devices. The usual research type for this is behavioural research to learn how spectators or users engage in visualisations or respond to them. The resulting requirement of this is being able to integrate sensing devices to the software, where visual applications can make use of the data generated by these devices, aligned with what is being displayed.

D. Developer: Developers should be able to use these environments for testing or debugging applications. They should have elevated access to the environment in order to perform their duties. Usually, core developers are given full access, while app developers are given limited or supervised access during their test or debugging session. This involves deploying and testing components and applications in distributed environment. Furthermore, developers should have debugging utilities to ease performing testing on this distributed system. Developers should be able to know what is happening on each display node or on the server, be able to execute actions on any of these distributed locations to effectively perform testing and debugging. This involves enabling a more permissive and accessible distributed system where distributed locations and components can be accessed and debugged remotely and they should be able to display their status and events on screens during the process.

3.1.2 Sessions in LHRDs

There are certain types of sessions we have encountered during our experiences in KPMG DO, which are different in terms of layout of spectators, communication style and how the environment being utilized. Identifying these will help us understand what kind of usage scenarios that we should enable in LHRDs in contrast to other platforms. These session in general are Unsupervised, Supervised, Presentation, Debugging, Partitioned and Remote.



Figure 3.1: Examples to Sessions in LHRDs (on GDO).

A. Unsupervised Session: Unsupervised session is a session where multiple users are free to launch and interact with various visualisations in order to perform tasks at hand. The common use case for this session type is exploration and free flow discussions and meetings. In comparison to the conventional discussions, users utilize the environment to launch various visualisations to support their ideas, show interesting aspects, compare to other users' visualisations and perform many other collaborative activities. Thus in general, visualisation space is used as a medium and aid for communication and collaboration. The boundaries where each user can deploy visualisations or control visualisations can be determined both digitally and verbally, or they can just deploy wherever is available and control all on-screen visualisations (simultaneously). Generally, this type of session better suited for exploration, fact-finding, discussion and other brain-storming and creativity sessions. Specific requirement arising from this session type, other than visualization deployment and management capabilities, is availability of concurrent user interfaces that can interact with display area simultaneously, thus multiple users can interact with display area using their designated UIs.

B. Supervised Session: Supervised session is where the environment control is not free for all like unsupervised session. In this session, a person with administrator role has the control of the environment. Admin has the decision over casting controls to spectators and making them users over decided apps on screen-space. Admin casts these controls to user tablet or laptop. In case admin decides not to share control, spectators can request to see different aspects of visualisations or data, and admin through his or her control interface decide to perform those changes. This is preferred session type when spectators are unskilled or untrusted for operation of the environment or when the discussion should be performed on a given structure or flow. One way of having this supervised control is to have a touch console on the grounds of the environment, where users can interact with the environment under supervision. Also, user devices can be given with a limited functionality UI, where only admin can cast controls to users. In comparison, the large touch console enables many spectators to engage collectively, and admin can teach how to use the system.

C. Presentation Session: This session type is used for presentations. There is almost no direct control given to spectators. Presenter either directly through an interface or by the help of admin guide the presentation. In comparison to conventional presentations, presenter can benefit from heterogeneous visualisations that can be used to show multiple perspectives of data, can take multiple paths on the presentation due to interactivity such as summoning up visualisations on-demand or program interaction or animation to visualisations to enhance its effectiveness on spectators. In order to perform this session type, either a complex presentation application should be built for carrying presentations or existing apps should be utilized and admins should be enabled with scripting functionality over app deployment and control of deployed apps. For example, admin loads a scene with certain visualisations, trigger interactivity or actions on deployed visualisations on that scene such as animations or change of data-sets. Admin can move to next scene (slide) through loading another set of applications. Thus, admin can perform a presentation on system level rather than inside an app.

D. Debugging Session: This session is used to debug and test apps and components by developers. Developers may not always get same results for their codes that they build on single screen setups in comparison to distributed display environments. Therefore, they may need to deploy and test on the environment itself.

Developers use various utilities not available to users to ensure correct operation of their coded parts. They test for:

- Generality and scalability
- On screen discrepancies, undesired artifacts and anomalies
- App or component launch and clean up
- Interfaces
- Touch controls if any
- Distributed operation
- App risk for epilepsy or other user health or comfort issues

In order to test and debug these, developers should be given certain distributed testing utilities. First of all, they should be able to see what is happening on each node and their status. They should be able to see what is happening on user devices where UI are displayed and also on the server. Furthermore, they should be able to react to problems by executing actions on the distributed locations and components where the problems can arise during distributed operation. This emerges the requirement of API based distributed operation, so a developer can execute dynamic actions on distributed locations.

E. Partitioned Session: Partitioned session is where different sessions are performed on different partitions of the screen space. This is possible only for tiled and large displays. This is highly recommended for medium and large screen spaces as they may be highly desired for use and thus it might be required to schedule them and this schedule is usually busy. Thus solution is to group non-priority activities that does not require full screen space or environment area together. In the partitioned session, users and admins are given control of different partitions of display area to operate. This can be determined verbally under same server or through other mechanisms where multiple deployments made possible under same server. On each partition, the team given the partition performs their session. If partitioning is tried to be performed digitally, than the system should be flexible on setup and thus multiple server deployments can run on cloud and power different partitions of display nodes without manually reconfiguring display nodes. If this is powered by a single server instance, than multiple user control/management interfaces should

work concurrently and they can interact with apps on screen space without having conflicts while they can agree on areas verbally.

F. Remote Session: Remote session is where two remote sites collaborate using these environments. Generally, there can be one server in charge of displaying visualisations on both sites, or two servers in communication between each other to display visualisations on their site only. On the first case, remote site has to connect local server and be able to duplicate display nodes on local server in their location. So in this case, both sites are showing the same display area by duplicate operation of nodes. So, each node display is open on all sites, and server broadcasts data to display nodes on all sites. Another case is, there is still a single server, but it is configured to host display nodes on both sides. For example, site A has 16 nodes and site B has 32 nodes, thus the server is configured to operate with 48 nodes, powering displays on both sites. Also, collaborators on both sites can deploy visualisation on both sites. In the final case, where there are two servers, one for each site. Servers has to communicate with each other to exchange commands for this dual operation. Another issue on this topic is, individuals interacting from remote sites. In this case, an individual should be able to open a control interface from a remote location and control visualisations on the local site. This session type also related with interoperability, where remotely hosted visualisations deployed to local displays and interfaces. The challenges raise by this session type is regarding flexibility of deploying clients of the system. For example, in a web based solution, the screen interfaces on display nodes can be deployed anywhere in the world or have duplicates operating at the same time. Same goes for user interfaces deployed to web clients on user devices. Thus, using web clients and flexible deployment, first two cases discussed here can be supplied. For the final case, the servers should be able to communicate and this has to be manually implemented on server code.

3.1.3 Use-cases

We have explored various use-cases that LHRDs under Visual data Analytics use can be utilized for. This exploration helps us to understand system and visualisation needs and requirements for these use-cases.

A. Exploration: In this mode, users can deploy several data visualisations to visualise their datasets. They can use any visualisation on any section of the screen

space on demand. Furthermore, visualisations can be controlled through control interfaces. Moreover, visualisation can be linked together on a logical level for coordinated visual scenes. This mode enables users to explore data from many perspectives. Data exploration mode is a generic mode and can be utilized by many areas. Basically, any entity that has to research on large amounts of data can make use of it. Furthermore, this mode can be utilized as a complementary to other modes as data exploration can be required time to time to the remaining modes in this list. The advantage of using LHRDs for this purpose is the screen space and dynamicity. Users can be subjected to both details and big picture, perform on screen comparisons as many visualisations can exist simultaneously and thanks to the dynamicity provided they can move visualisations around, open and close, dive in or focus out at will or script them to do various animations. Finally, they can change how data is connected and processed and visualised on-demand, such as changing datasets, visualisation styles, processing elements on will. In order to enable this mode, users should be able to deploy general data visualisations with any configuration or data on any screen area and be able to control and interact with them. Furthermore, these visualisations should be able to sync and operate collectively on-demand.

B. Collaborative Analytics: In this mode, users of the environment use the system as a digital visual aid to collaboration. They can project any visualisations on screen-space to back up their discussion, demonstrate a problem, present an idea and conduct many other collaborative activities. They can control collectively the visualisations in the screen-space, changing parameters, data and other variables. They can interact with visualisations using various interface and interaction methods. Various collaborative activities and sessions can be performed in LHRDs. Therefore, users can use the visualisation space and interfaces as a supplementary system for collaboration and utilize it as a medium for communication using data, visualisations and analytics. This mode is suited for data science and multi-disciplinary projects as it enables researchers from different backgrounds to express themselves to other researchers through data visualisations and perform analytics not based on abstract conversations but using visual screen-space. This mode requires, in addition to the requirements of exploration, a management functionality on the system that is purposed for visual data analytics. Furthermore, since it tar-

gets multi-disciplinary projects, it would require custom app development in order to accommodate those projects' visualisations on the LHRD.

C. Design and Simulation: In the first part of this mode, LHRDs can be used as a 3D design studio. This can be industrial design, architecture, city planning, physics, video games, etc. Main advantages over conventional visualisation techniques is if the design project contains data, this data can be visualised at the same time on the screen space and effectively monitored during the design session, through using other apps and overlays. Furthermore, renders of different angles or filters can be displayed simultaneously for comparison. Being able see multiple angles and filters alongside with data on same screen space greatly enhances and speeds up the decision making process on every step. As a result, 3D design mode will provide unique capabilities especially for design projects that has to consider lots of perspectives during design process, such as AR, VR development, architecture, system design, industrial design in many sectors and many other areas. In order enable this part of this mode, 3D visualisation libraries should be enabled in the system, where they can create 3D scenes using distributed displays with proper scaling and without any performance issues. Moreover, the apps should operate over API, and thus accept stream of contents and control.

Simulation part, takes design mode one step ahead and enables the simulation of design (Not necessarily have to be 3D). In this mode users can visualise simulations. Through this mode, users can utilize large screen space to display the results of the simulation or simulations compared to the actual data on the same screen space. The main advantage here is that LHRDs allow researchers to compare results and existing data on many aspects. Furthermore, the on demand deployment of visualization modules allow researchers to investigate events of interests. Moreover, these simulations can be easily connected to research data and over API can be used to enhance or program simulations. In order to enable this part of the mode, this 3D visualisation library or app can be inherited or customized for further interactivity rather than being a 3D demo or design studio and hence have specific app logic and interaction functionality designed for simulation.

D. Operation and Control: There are many organizations that have to manage large operations like traffic, factories, hospitals, cities, space technologies, etc. Since these operations have an ever evolving nature due to their size and cost, they have

to be actively monitored, researched and improved. If LHRDs are connected to the systems that manage or store operational data, these studios can act as an operation and control center. However, unlike the other operation centers, due to their versatility the research on the operation and development on the systems that the operation is based on can be performed in the same studio. Moreover, data exploration can be used to solve problems and anomalies day to day.

This mode requires display of heterogeneous data visualisations such as maps, graphs, charts or video streams simultaneously. Furthermore, these applications should be able to receive data as a stream and display visualisation of this data dynamically. Admins of this mode should be able to focus on certain visualisations or change layout visualisations or datasets or configuration on-demand thus the platform should support these capabilities.

E. Research and Development: LHRDs can also be utilized for research and development. This mode makes use of other modes like design, simulation and exploration. It will be deployed in a research center for being a hub for that research area. It can be used for various custom apps for research or use of common apps for research purposes. Furthermore, through the interaction devices the studio itself can be transformed into research environment where spectators can be turned into subjects of the research.

If the research is on a digital product or device, the studio can also behave as a development center for that research. Devices can be designed, simulated, operated and tested using the LHRDs, considering the programming elements can easily be connected to visualisation modules. The large screen space will enable the display of data the devices produce at the same time the current states of the programming. This enables the developers to see correctness of their programming through monitoring the data. This mode can act as a hub for device and system research, especially for devices and systems that operate in physical world through sensors and cameras, where their view can be visualised in the system alongside data and algorithms.

The requirements of this mode is very similar to previous modes with only addition of integration of research devices such as sensing devices or external systems to the platform and enable utilization of these devices and systems by the applications of the platform.

F. Presentation: The presentation of any mode can be performed in the same screen space. Thanks to versatility and programmability of such environment can provide. This presentation can be dynamic and interactive, which is up to the creativity of the developers. Users can create dynamic presentations that can take multiple paths. Presentations can be animated on multiple visualisations across the screen-space. For example, a presenter can start from left to right and during that animations activate on the region presenter explains. These presentation can be connected to direct or indirect interaction devices. Spectator data can be collected and can affect presentation. Audience can connect through mobile devices to presentation quizzes or feedback. As a result, complex, dynamic and interactive presentations can be performed.

This mode requires a specific presentation app or system functionality that enable presentation through other apps. In either case, presentation should be able to deploy and layout visualisations, control these visualisations, program or script these visualisations through a collective logic. Furthermore, this logic should be connected to an interface where users can design a presentation within the app or through system utilities with desired data scenes on every slide, and also interaction capabilities. This requires distributed scripting, coordination, API based operation on general data visualisation apps and an underlying management system.

3.1.4 Conclusion

In conclusion, we have explored collaborative side of visual data analytics, types of users, types of sessions and variety of use-cases. We have identified some general and specific requirements arising from these issues. This aspect provided and prepared us in terms of what kind of software would be required for Visual Data Analytics on LHRDs for multi-disciplinary data-driven research. In the next aspect, we will delve into specifics further by exploring immersive display and management of data visualisations which is one of the obvious requirements of this aspect.

3.2 Immersive Display and Management of Data Visualisations on LHRDs

This aspect focuses on mechanisms for deploying, rendering, controlling and managing data visualisations on LHRD, in an immersive way for the purposes of visual data analytic. First of all, a software for visual data analytics on LHRD should

enable scalable and heterogeneous data visualisations to surround users of the environment. In the first section, we would be identifying various display mechanisms regarding this. Furthermore, such system should provide management mechanisms for deployment and arrangement of data visualisations, also administration of the system as we identified in the previous aspect. In the second part of this section, we will explore requirements in management area of the system. Moreover, in a system like this users should be able to interact with visualisations in order to configure, control or steer them. In the third part of this section, various interaction mechanisms will be discussed to determine the interaction requirements of the system. Finally, we would explore remaining topics such as reliability, inter-operability and remote operation.

3.2.1 Display Paradigms

Immersive data visualisation environment for analytics on LHRD should come with certain display mechanisms. We have identified most mandatory ones in this part and divided them into three categories: Distributed Rendering, Heterogeneous Visualisations, Layering.

A. Distributed Rendering: First of all, on a platform such as this, visualisations should be deployable to multiple distributed displays. In the literature, this can be achieved through various hardware and software solutions. In this thesis, we are only interested in software solutions that are compatible and suited for visualising data. Among various methods that achieves this in the literature (discussed in chapter 2), we are only interested in ones that can achieve visual scalability. Therefore, the key challenge here is to be able visually scale data visualisations, where apps on the system will be able to scale to large number of nodes while displaying millions of data elements without having major overhead on system resources that can affect load time or responsiveness of the app. Failure to do this will cause spectators of the environment to wait considerable amounts of time for data visualisations to load or wait when they interact with visualisation by re-configuring or steering them. Data visualisations should be responsive enough when scaled. Failure to achieve this will cause most use-cases to fail that is identified in the last section. As a result, we would need a system to scale general data visualisations to screen-space composed of large amount of distributed display nodes.

B. Heterogenous Visualisations: Secondly, users should be able to deploy various visualisations on to the display environment in any layout. First requirement is to be able to deploy more than one data visualisations to display area and in different layouts. The key here is the flexibility, hence allowing users to deploy any visualisation to any area and any combination of visualisations in any layout. Second requirement would be to allow users to be able to deploy and unload visualisations safely through interfaces or through API. Finally, multiple instances of the same general data visualisation with different data and configuration should be deployable on the same display area. As a result, the system should enable deployment, arrangement and unloading of general data visualisations through a management mechanism in any layout to any area on the screen-space of LHRD.

C. Layering: Another paradigm that we are interested would be layering of visualisations. This mechanism could be used as overlay of data on existing visualisations, overlay of system utilities or AR functionality. This mechanism would work through synching spatial coordinates of visualisations that stack on top of each other through z-indexes on display area. Through transparency, visualisations with higher z-index would serve as overlays and provide data enrichment or system utilities over other visualisations. System level of the platform should allow system level overlays and also apps should be able to benefit from layering for displaying data with multiple layers or displaying other information such as labels or legends over the visualisations. Therefore, the system should enable layering on its interfaces and display nodes both at system level and app level.

3.2.2 Management Paradigms

Another important aspect for enabling immersive use of LHRDs for visual data analytics is management mechanisms which enable management of distributed display area and visualisations on it.

A. Unified Display Area: First mechanism is regarding displaying a unified display area. This mechanism is about running system logic on individual display nodes and enable overall unified operation on cluster of display nodes. Display area would be blank by default. Users should be able to deploy various visualisations on selected areas through coordination of system logic on cluster of nodes and server.

B. Visualisation App Deployment: Users of the system should be able to deploy any visualisation app to any area in any layout onto the display nodes. Furthermore, users should be able to select configuration of apps and be able to interact with deployed apps. Finally they should be able to unload apps from display area safely without left overs. These should be made available on easy to use interfaces to enable sessions and use-cases described in the previous chapter, so an user should be able to select various apps, select their configurations and be able to deploy them in any layout through this interface.

C. Saving and Restoring States of the Environment: Our experiences from KPMG DO showed us that different sessions require different content as the environment is used for many diverse projects. Therefore, mechanisms will be necessary to save the current layout as a system state and be able to transition between these saved states. Therefore, the system should be able to store or restore visual layout of the apps with their configurations. Furthermore, system should able to trigger interactive elements on restoration of states, such as starting animation, syncing, etc.

D. Debugging and Testing: Another important issue identified in the previous aspect is debugging and testing. Since this is a distributed environment, developers cannot always develop on the environment itself. Therefore, they usually develop on a single computer and use the LHRD facility for testing. Since testing on LHRD is performed on a distributed and clustered system, it requires special utilities to aid developers in this process. First of all, distributed elements should be able to show their states and current events for developers to see what is happening. Furthermore, developers should be able to test events or take actions on certain distributed elements to investigate further. Hence, remote command execution mechanisms and a maintenance mode that shows on-going events and node status on nodes of the system is required to aid this testing process.

3.2.3 Interaction Paradigms

Last part of immersion is regarding providing interaction capabilities to users and admins in the system, thus they can interact/manage/control/configure or steer the data visualisations and overall system. Many other LHRD platform does not provide such capability due to difference in purpose or objectives of their era. How-

ever, in a modern visual data analytics platform, these capabilities should be enabled for effective use of the system for analytics.

A. Management Interfaces: Management should be performed through an interface. It should provide configuration to system and utilities for operation of the system. These include deployment of visualisations, tracking system health, debugging, saving and restoring states, and executing scripts. Through these, admin role determined in the previous section could administer the system accordingly. Furthermore, this interface should be available on mobile devices such as tablets, so administrator can do his/her job on foot while overseeing users and spectators.

B. Interaction Interfaces: In visual data analytics platforms, visualisations should be interactive. This could be performed through control interfaces which are assigned for interaction with deployed apps. These controls are app dependent but from our experiences they can be used for changing parameters of the app, loading or selecting data to visualise, steering the visualisation and many other. These interfaces could be used for controlling all deployed instances of an app, a single instance of a deployed app, or deployed instances of a group of apps. Through these interfaces, users can interact with data visualisations and thus can conduct various roles, sessions and use-cases.

C. On-screen Control: On-screen control is where users interact directly with visual elements on display area. This could be on various levels. On system level, a pointer could be made available that acts like an OS mouse pointer and thus users can interact with the system like a desktop. An on-screen menu can be opened at pointer location for advanced interaction. For example, SAGE2 uses this mechanism. On app level steering of visualisation can be performed through pads, interaction devices etc. On-screen control is an optional capability where it is not directly required by visual data analytics. However, it can be used by exploration sessions where the display environment can be used like a desktop.

D. Casting: This mechanism is ability to cast a control interface of a visualisation to another user or casting a visualisation directly to another user's screen. This would be used for collaborative supervised sessions. Using this mechanism, admins can cast controls of an app to other interface devices or cast private visualisations to other users that has interface capable device. This requires dynamic user interfaces

or nodes that can receive and act on remote casting.

E. Interaction Devices: Various interaction/sensing devices can be deployed in LHRD environment to aid visual data analytics. They enhance interaction with the system and also be used for scientific purposes such as data collection over the spectators for running behavioural studies.

Most common devices are:

- **Common Control Devices:** Many devices for interaction like joysticks can be connected for steering visualisations.
- **Stereoscopic Glasses:** for 3D viewing of data visualisations
- **AR Glasses:** For data enriching through personal overlays to users on elements on the display area.
- **Motion Sensing:** For detecting users in the LHRD area and either collecting data or enabling gesture control.
- **Voice Recognition:** For interacting with system or visualisation through verbal commands.
- **Image Processing:** Similar to motion sensing, actions and emotions of spectators can be processed.
- **Eye Tracking:** For detecting where the users are gazing, so it can be used for behavioural study.
- **Biological Sensing:** Similar to eyetracking, other biological data like EEG or EKG can be processed through mounted devices.

Therefore, the system should enable connection of these devices and make their data or controls available to apps in the system. As a result, apps can utilize these devices, use data generated by them, or take actions on the devices. Developers should be given the option to make these devices available globally or locally to an app. Thus, multiple apps can utilize the same device or an app can be built for specific device use.

3.2.4 Other Paradigms

There are also some other issues regarding immersive display of data visualisations and management. These are insuring reliable operation in this distributed

system with external interactive, dynamic and distributed applications; interoperability with other systems and remote or duplicate deployment of clients and server of the system.

A. Middleware and OS: The platform should have a robust middleware and operating system that manages and enables operation of distributed elements and also logical elements of the system. It should structure communication between display nodes, control interfaces and server. It should also create logical scopes for apps and system in these distributed locations and structure communication between these scopes. As a result, system should be able to supply concurrent communication between all these logical elements on the distributed physical locations in LHRD and enable distributed display area, app deployment, operation of apps and utility functionality that works on this distributed system.

B. Reliability: This system will be utilized by vast variety of projects which needs many general or custom applications. Therefore, safe guards on reliable operation should be provided, so the system should not encounter problems when a new app is developed and deployed in the future. Therefore, it should force apps to operate on pre-defined scopes on both server and clients, where operation on these scopes does not conflict with system or other app operations. Furthermore, creation and termination of these scopes should be safe. Moreover, apps should communicate on different channels thus they do not conflict with the system or other apps in terms of concurrency. As a result, a general and instanced distributed app paradigm should be established that places system and apps on different operation scopes both on logic, storage and communication.

C. Inter-operability: Another functionality is inter-operability with other web based visualisations or solutions. First of all, integrating a web-site, a web visualisation or web-based remotely steered visualisation should be made easy. For example, a static web site or web visualisation should be easily scaled (preferably through pseudo-scaling). If that web visualisation have a control panel, it should be displayed on an user control interface. Secondly, other web based LHRD solutions like SAGE2 should be integrated as they should be able to display their node clients inside the system's node clients and their control clients inside the system's control clients. As a result, system will become efficiently interoperable as it can

easily integrate or instantly display many web visualisations or content on LHRD with pseudo-scaling or host other web-based node-control client logic based LHRD software, thus users of the environment can operate all of these under the same roof.

D. Remote Operation: Another challenge here is supporting remote operation. Node or control clients of the system should be flexible and enable duplicate deployment, concurrent deployment and remote deployment. Thus, node clients can duplicate on remote locations, and hence a single server can power two sets of screens: one on a remote location and one on a local location, in a mirrored style. User control interfaces can have concurrent instances on both local and remote locations, thus a user on a remote location can control the system on a local site. Finally, if the system clients can be deployed remotely, a server should be able to power remote nodes or user control clients on multiple locations and enable interaction between each other.

E. Flexibility: Final challenge is flexibility. Flexibility in our context is ability to deploy this software to almost all LHRDs of today or even future. Therefore, the software should be able to work with many different setups, number of screens, different resolution overall or per screen, different network setup or interaction devices. Furthermore, the clients of the software shouldn't require prior installations of neither the system or the applications, as they should be able to work with any selection of apps immediately over the network.

3.2.5 Conclusion:

In conclusion, we have gathered main features and capabilities for display and management side of this new approach. As we can see, these focus on a system where users can deploy various data visualisations on the unified distributed screen-space in any layout, arrange those visualisations and individually control them. System should make use of functionality like layering and casting. It should enable management mechanisms for session management, presentations through system level utilities and finally debugging and testing. Furthermore, the system should have various user control interfaces and enable integration of devices to the system. Finally, it should provide safe-guards on its mechanisms for reliability, enable interoperability for versatile uses and enable remote deployment scenarios for cross-site collaboration. Through these, system level features and capabilities become clearer

for a system that targets this type of use. In the next aspect, we will focus further on data visualisation apps and how they should be shaped.

3.3 Data Visualisation Apps and their properties on LHRDs

This aspect focuses on providing data visualisation capability in LHRD environment for visual data analytics use. Users of these environment will collaborate through data visualisations surrounding them. Therefore, these environments should provide a framework for delivering vast range of visualisation options for users to conduct visual data analytics effectively and it should also provide a framework to develop these visualisations by external developers. This can only be achieved through app orientation where visualisations are implemented through apps. Users would mesh functionality by deploying a selection of visualisation apps with given data and configurations. As a result, apps could be developed separately, a selection of them included in deployments, than users can deploy included apps to display area with given configurations and data, to enable any data visualisation scene to conduct visual data analytics. In this section, we would explore properties of apps in general that the software approach should support. We gathered this list through the requirements and others mentioned in the previous and next aspects, also from our experiences.

A. Versatile Apps should be versatile. The platform should support development of many data visualisation and other supplementary apps with different needs. It should support general data visualisation apps that can visualise various data in a certain visualisation type. Also, it should support custom apps that uses either these general apps to visualise or other libraries to visualise, designed specifically for custom needs. Thus, we have gathered a list of potential general apps which through meshing functionality can supply needs of versatile use-cases of multi-disciplinary data-driven research in general.

First of all, the platform should enable development of basic data visualisation apps like Charts, Graphs, Maps, Spreadsheets, Flows or Diagrams. Furthermore, it should support apps that can display Images and Videos as supplementary use. It should support apps that can visualise 2D/3D content both generally, also for custom use. It should also support a dedicated presentation app that can use other

visualisation apps as parts of slides on LHRD, where each slide is a layout of visualisation apps. It should also have mechanisms for overlays such as heatmaps, labels, QR codes, pointers or drawing illustrations on visualisations. Another useful app or functionality is screen-share that users can cast their screens as visualisations. Finally, it should support generic or static web pages to be directly displayed on the display-area. In addition to all these general apps, it should allow any custom app to be developed for any specific research need.

B. Modular: Apps should be modular. They should operate as plug-ins. This enables them to be independent from the system. Thus, different system deployments can work with different selection of app libraries. They should also have instancing mechanisms where multiple instances of them should be deployable, safely operated, and terminated without left-overs. Instancing also will enable certain generality where an app can be deployed with different configuration and data simultaneously to screen-space.

C. General: Apps should be general. They should support visualising with different data and configurations. Through these, versatile use-cases of the platform can be satisfied. For example, a graph app should accept different data and configuration for visualising different graphs for different projects. These can be provided to app on deployment to display area, where users can upload these data and configurations through UI, or app can be loaded with given data and template through configuration files, or app can be deployed by another app through API with given data and configuration. This enables apps to act as a general visualisation app of that specific visualisation type and thus enable them to be re-usable with versatile use-cases.

D. Distributed: Apps should be distributed. LHRDs run on display server, cluster of nodes and control clients. Therefore, app logic should extend to all these distributed elements and operate in a distributed manner. These components of the app should communicate with each other and operate robustly, where each component would be implemented by app developers. This enables developers to decide how the app should visualise under a distributed scenario and overcome scaling and other problems through implementation for that specific use-case. Furthermore, this enables developers to select which libraries to work with on each distributed

component of the app.

E. Scalable: Apps should be scalable. They should be deployable to tens even hundreds of displays. They should support various sizes and shapes. They should be able to scale the number of visual elements or data elements within the app for any given data or configuration, all without major overheads on the system. As they scale, there should be minor increase of overhead, since the app logic on each client handles load in a distributed manner as resolution increases. Thus, there should not be major differences in deploying an app to 1 vs 64 nodes. Furthermore, scaling and instancing allows creation of many different layouts of apps through using existing apps with new configurations and data.

F. Interactive: Apps should be configurable, controllable and interactive. Use-cases determined in previous sections showed that data visualisation needs require specialized interaction with apps, rather than standardized interaction which is common to all apps, and usually only support basic arrangement or basic navigation. Therefore, they should have user interfaces for controlling, configuring or steering. Through these, users should be able to select data sources, select style, configure the app or steer the visualisation. They should also be collectively controlled or interacted with. As a result, a group of visualisations that should act together can be interacted collectively. They should be connectible to interaction devices and should be able define how these interaction devices should connect to the logic of the app. Thus, through these devices users steer the app or interact with the app with a more suitable way, such as using a joystick to steer a 3D simulation.

G. Customizable: Apps should be customizable in addition to being general. They should have configurations, templates, styles and so on. They should allow selection of customization of visual elements through configuration files or control interfaces. Therefore, different apps can be styled together to show an immersive data visual scene in a collaged way. Or an individual app would be styled to show a data visualisation that can highlight certain properties. For example, a graph visualisation where certain type of nodes are shown in different size or colour to emphasize their importance.

H. Extensible: Apps should be extensible. They should be able to use various external libraries, data and systems to add or implement functionality. Since these

apps will be distributed, this property can be used to aid different purposes on server or client side of the app separately. On client-side, app should be allowed to use many visualisation libraries on the web to visualise or other supplementary libraries. On server-side, app should be able to use various libraries for coordination of the app, data drivers for connecting to data sources or processing libraries for processing data.

I. Web-based: Apps should enable rapid development through web technologies. In a platform like this every day many other visualisation needs arise. They should be able to keep up with this. Therefore, they should support rapid development of visual elements and interfaces by app developers using web-technologies. For examples, using web development to easily building apps, using generality and modularity of apps to combine apps together and create custom visualisations through composite apps.

J. Standardized: Apps should be standardized. Even though they allow developer freedom on server-side, client-side and communication API implementation, they should have certain standards for enabling operation of various utilities and core system functionality to interact with them through these standards. Thus, they can be deployed, terminated, scaled through APIs. Furthermore, their states can be saved or loaded.

K. Reliable: Apps should operate robustly and be reliable. They should be able to recover from client failure, restart or connect. They should be able to switch data sources on-demand. They should be re-deployable. They should clean on termination. They should operate on the scopes given to them and not conflict with other apps or the system itself on the server, clients or on middleware.

L. Castable: App's interfaces or visualisations should be castable to other screens and control devices. This enable individuals with admin roles to cast UI of app to other users during supervised sessions.

M. Data-Driven: Apps should be data-driven. They should allow access to remote data sources both on client and server side of the app. They should allow distribution mechanisms to be implementable by developers within the system. They should enable storing temporary and permanent data within the app and be able

to use many libraries to process it either on the server, or client side of the app. They should allow streaming of data to constantly update the apps for use-cases like operation and control.

N. API-based: Apps should be linkable to other components, apps, data, core logic through APIs. They should both benefit from implementing functionality through other elements and data or enable other elements to implement functionality through them. Thus, different apps can utilize same datasets, and also they can utilize other base apps or mechanisms, all through APIs. They can use system logic API to load and unload apps or execute on-demand scripts for changing system states or creating animations. As a result, complex sessions and use-cases can be implemented through meshing functionality and implementing collective logic using multiple apps.

O. Collaged: Apps should be collagable. They should be able to talk with other apps or visual components and be operated in a coordinated way with them to implement collaged visual scenes. Thus they can be synced on different parameters or have collective logic operating them.

P. Multi-type: There should be multiple types of apps to satisfy different scenarios. First of these is service type apps that works globally and supply resources to other apps in the system. These can be used to represent device integration or another systems API or a data service. Thus other apps using this app can utilize these resources. Also there should be composite apps that can deploy and use other apps for visualisation needs. Their sole purpose would be to create visual data scenes. One major use-case would be a presentation app that can create slides out of layouts of visualisations.

R. Inter-operable: System should also support inter-operability. This means any web based visualisation project including web-based LHRD software which follow remote control logic to be integrated to the system. For example, a web visualisation with a separate control interface should be representable as an app where the visualisation is scaled to screens which the app is deployed and control interface is displayed inside app control.

Conclusion: In conclusion, we have explored properties of data visualisation and other supplementary apps under a visual data analytics platform on LHRDs. These

properties will enable versatility in app development which can be used in general on data-driven multi-disciplinary research that we set our objective to aid. As a result, basic use-cases can be actualized through applications based on these properties. In the next section, we will look into more in depth concepts for creating complex, dynamic, interactive data visualisations or synced collages of visualisations or complex data scenes.

3.4 Data-Driven, API based, Collaged Data Visualization System on LHRDs

The findings in the three previous aspects can provide basic use-cases for a visual data analytics platform on LHRDs. However, there are more complex use cases that users of this platform may want to utilize. First of all, the apps of the system should be data oriented and various methods have to be available for them to satisfy their data needs in terms of data availability, processing, distribution and load balancing. Furthermore, a desired functionality would not be tied to a single data-visualisation app, often it may require collective approach where multiple apps are used for that visualisation need. As a result, the system should provide methods to coordinate apps on API level, collage them using API, sync them on parameters again using APIs and allow development of composite apps that can use other general visualisation apps to visualise and offer a composite control over that specific visualisation need. As a result, we will explore data-driven, API based scripting and coordination, collaging, syncing and composite app development in this section.

3.4.1 Data-Driven Apps

In a platform targeting to perform versatile general visual data analytics on LHRDs, that platform should enable use of data in the system freely. Therefore, this aspect focuses on how the data sources should be made available to visual elements and logic of the apps in the system. This topic is interested in accessing to data sources, distribution of data within the system and load balancing on nodes. It explores variety of common data sources, how generally these can be accessed by the system, how they should be distributed to elements of the system such as app components, in addition how data should be shared between apps.

A. Data Sources: The system should be able to use various data sources for powering visualisations. These could be files that can be uploaded by users or downloaded by the app. These could be databases that reside on the server that the platform is hosted or externally somewhere remote. First of all, apps can query data on the server or on an external location and distribute the results of the query to display nodes. Another option is using cloud resources where drivers for these could be used on the server that transfers map-reduce and other queries of the apps to these resources and thus enable apps to utilize these resources. Data streams also can be utilized, these can be traffic data, social networks or video streams. Through API, app logic can connect to these resources and stream the data. Last option is data services that work over API. Apps can connect to API of these services and query requested data. As a result, all these options be available to app developers to utilize for their data needs and system and app paradigms should provide this availability. The system should allow apps to connect to internal and external data sources on both server and client scopes. Furthermore, it should allow apps to accept uploads from interfaces for users to provide data.

B. Data Processing Options: From our experiences, data either processed locally or globally. Globally means data is processed by the app logic of the server and distributed to nodes. Locally means, data is distributed to nodes or nodes acquire data by their own and process locally in a distributed manner. Different apps would be requiring different processing mechanisms and app development mechanisms on the system should allow all possible scenarios. Therefore, system should allow processing of data by the app on all the locations of the distributed system.

C. Data Distribution Scenarios: Another issue here is how data is distributed to nodes. Again there are two main scenarios. First one is data is originally stored or accessed by the server-side logic of the app. In this scenario, app logic distributes the data either in raw or after processing depending on which nodes will require the data. If data size is small and visualisation is highly interactive or dynamic, in which case data points that are needed by nodes changes rapidly, server-side of the app may choose to broadcast the data as a whole and app logic on nodes query the local copies. In the second scenario, app logic on the nodes directly access to data. They may choose to download if the data size is small, or query as they need it. Again in this scenario, data may require processing prior to visualisation, thus since

clients handle data acquiring and storage, they have to perform processing too.

Usually, distribution and processing should be aligned with node partial visualisation. This means, which data elements does that part of visualisation needs. Node should acquire the data for their partial visualisation only. If app is too dynamic, and this need changes too rapidly, nodes should provision data in advance depending on the dimension these changes occur. For example, in spatial plane, where users pan and change location, nodes may need to perform overpulling few units ahead in the plane. This buffering could make the app more responsive and prevent visual artifacts due to loading. As a result, the system, the middleware and the client paradigm should allow or enable all distribution scenarios mentioned, thus an app developer could implement any of these scenarios as they need, since different apps may require different distribution scenarios to be implemented.

D. Data Sharing: Apps in this platform may need to collage. Other than coordination or syncing of collaged apps there is one more issue. This is multiple instances of an app or multiple apps using same data sources. Instead of downloading or querying these data sources in a duplicate way, apps or distributed components may choose to share a data source between each other. This can happen in a way that one app downloads the data and provides the other apps server logic the data. One app's server logic can provide data to client logic of multiple apps. A composite/parent app can have access or store data and distributes to multiple child apps. All these scenarios would require inter app communication and communication over API between distributed components of the system including components of different apps both remotely and locally.

E. Load Balancing: Another issue regarding processing and distribution is load balancing. Apps should be able to load balance processing or rendering tasks. This should happen in apps with heavy client side processing. Display nodes with high loads on an app should be able to send these tasks to other nodes with low load, either over server or directly through P2P, to perform the processing and bring back the processed data/rendering. App logic on the server can coordinate this and broadcast to apps with information on which nodes are with low or high load. Also, through P2P, nodes can create a swarm to exchange this information and perform load balancing. This option should be enabled to apps by the system, so visualisation needs that come with heavy loads on certain nodes can use this to overcome their

performance problems.

3.4.2 API based Distributed System Operation

Both system and their components operate on a distributed manner on nodes, server and control clients in a system like this. Users, apps or system utilities need to create functionality using various distributed components through API and sometimes in an on-demand fashion. This requires on-demand distributed operation over API on the system. Examples to this use is, apps needing load balancing and form P2P swarm to exchange tasks and results, then visualise. Another example is, admins executing scripts or switching states between sessions, certain sessions require constant transition of states both on overall system and on individual apps during the session, which can be done by distributed programming over elements of the system. Utilities may access to certain distributed logic of apps on certain nodes or clients to perform tasks over API. A database may be connected to multiple apps and data may shared over API. As a result, utilities, apps and users can define functionality over distributed components by programming/scripting interaction between them on an on-demand fashion using APIs on a logical layer. Through this, Users Interfaces, Apps, Scripts, System Utilities should be able to access and command over API Apps, UI elements, System Utilities and Functionality, Nodes, Interaction Devices, Data Resources.

3.4.3 Collaged and Synced Applications

One of the most significant use cases for API based operation is collaging and syncing. This is basically linking visualisations on-screen space through certain methods so they coordinate and present us with a collaged data scene. For example, a self-driving car Research and Development scenario: On display area various data of the car in map app, chart app and so on is shown; at the same time showing car cameras and sensors views; at the same time showing algorithmic states of logic that read the sensor data; all synced in temporal. Thus, allowing researchers to easily identify a sensor defect or algorithm problems. There can be many reasons behind coordinating on-screen visualisations in use-cases of this system. This aspect provides numerous ways to collage visualisations. It is about linking one visualisation app to other app or a group apps to each other.

First way to achieve this is parameter sync. Visualisations can sync on temporal dimension. Data on two or more app can be synced on a time-based parameter. This allows apps to show different aspects of data on different visualisations all synced on time. They can be showing different datasets or different perspectives of the same dataset or same data with different visualisation apps, but all synced in time.

Apps can also sync on spatial dimension. This means, two apps can show the same location. This can be used for heterogeneous apps side by side or overlaying apps on top of each other. Each app can show different data but they are synced on coordinates. These can be instances of the same visualisation like two map apps showing same location but different data, or they can be different visualisations like one map app showing data on a location and another app like graph showing the same data on a logical visualisation but for the same coordinates.

Apps can sync on other data parameters. For example, a database may have 4 datasets, each app showing the same dataset from a perspective such as graph, chart, etc. Switching between data sets cause all apps to switch. Another example is that one app is showing a flow of a series data animated and another app is showing anomalies in the data and focus them as they appear in the series. In other words, as series move, anomalies pop up on the other app. There can be many examples how data on a single or multiple datasets can be synced.

Moreover, these syncs should respond to interaction. On temporally synced apps, starting animation should start animating on all apps; or on spatially synced apps, panning a map on a control interface should affect other apps that are synced too. Furthermore, apps can be linked through interaction devices. An interaction device can be linked to interact or control app or collaged scene. For example, there are two 3D apps and a controller controls view on both. However, more complex interaction scenarios can be implemented. For example, an interactive and dynamic presentation. Users through interaction devices directly or indirectly affect the path presentation takes. This can be done by creating the presentation through links and connecting interactive devices to logic.

Apps also can be linked through their states and other parameters rather than data. For example a chart app has pie chart and a bar chart state and on two linked chart apps switching state affects both apps. Other complex links can

be formed between apps on their logic or states. This is basically programming how each app, state, logic, visual element, user, interaction device coordinates and operates between components and apps. Furthermore, there are various locations coordination and linking can occur. Coordination can be on multiple visualisations or components, or it can occur on multiple dimension including AR dimensions. Moreover, interfaces of coordinated apps and components can be aggregated and combined to a single interface.

Finally, combining previous items, composite parents apps should be implementable that can deploy and unload apps to display area, control them and coordinate them at user will through APIs. This enables creation of specific use-case apps that utilize other base apps such as graphs, maps, charts, images, etc for that specific use case. Composite app itself can determine layout and coordination of child apps or users using parent app can create complex data scenes for specific purposes such as advanced data presentations using multiple apps and layouts in slides on overall system, where layout is determined on-demand.

As a result, system should allow certain paradigms to enable collaging, syncing, combining apps over the distributed system using APIs.

3.5 Conclusion and List of Technical Challenges and Requirements

As a result, we have discussed 4 aspects of Visual Data Analytics on LHRDs. These are collaborative use, immersive display and management of visualisations, visualisation applications and their properties, and finally data-driven, API based and collaged application paradigms and mechanisms. We also briefly summarized and listed these requirements and challenges in Table 3.1. What we learned from this exploration is the challenges and requirements we face in designing and implementing a platform with this purpose, which have not been implemented before. As it can be seen, this novel approach comes with many challenges and requirements that were not present on other examples in the literature. However, these challenges also shapes our design and architecture, since it specifies how a platform like this should be actualized and thus help us on implementing this platform.

In comparison to the literature, we have observed that the use scenario in our case is very different in nature than examples on the literature. Due to it, the

general architecture and paradigm of our platform therefore should be different than both older and newer generations of LHRD software to enable these use scenarios.

If we compare to the older generations, the objective and use style has to be more up-to-date to changes in literature in this target platform. Therefore, as you can see, there are many challenges emerged on implementing a web-based platform with app orientation where applications are distributed and operate over API, so does the system. It is close to impossible to conduct identified use-cases with older platforms compared to this new approach. The way older generations approach to displaying data visualisations is simple. They provide OpenGL or other graphic distribution mechanisms to be used. They do not have management system, app deployment mechanisms, app development framework, do not use web technologies and thus lack flexibility on many aspects. There are no mechanisms for advanced interaction with the system or apps. There are no methods to combine, collage, coordinate apps. There are no mechanism to ease connection, streaming, processing or distribution of data to apps. Therefore, they are obsolete in terms of our use scenario and objective.

If we compare to the newer generations such as SAGE2 and others. We observe that there are similarities in collaborative use of the platform in terms of roles and session types and some use-cases. Although, the way these users and sessions utilize the LHRD is very different. Instead of deploying collaborative and productivity applications and interacting with them on-screen, this platform focuses on data visualisation deployment, interaction, control, configuration and steering of visualisations. Therefore, many challenges arise on the management of the system, structure of the system, applications of the system in comparison to later generation LHRD software. What is different is the way the system have to be managed, how data visualisation operate and interacted, how system operates is different in nature. System has to implement, scaling data visualisations, different deployment and arrangement mechanisms, different interaction and configuration mechanisms, different data connection, processing and distribution capabilities. It should also implement collaging, syncing, coordinating mechanism for data scenes. Due to all these challenges, even the architecture and base paradigms of the system should be different in comparison to examples in the literature, in order to accommodate these requirements.

As a result, we can conclude that this new approach should overcome great amount of challenges in given aspects and come with a new architectural, system and application approach to enable them. As a fact, other LHRD platforms may look similar in some use-cases, like visualising a spatial data on LHRDs, when we take a deeper look, we see vast differences on operation, capabilities and functionality compared to our desired level of functionality, and that is what this thesis intends to solve by introducing a new software approach for conducting immersive and social visual data analytics for multi-disciplinary data-driven research.

Table 3.1: List of Summarized Challenges Resulting from Each Topic in this Chapter

Collaborative Use of LHRDs in Visual Data Analytics		
	Roles	
	Admin	System requires Management interface preferably on mobile device
	User	System requires App deployment, layout and control mechanisms
	Spectator	System requires grid based distributed screen-space
	Developer	System requires distributed debugging interface and tools
	Sessions	
	Unsupervised Session	System requires concurrent user control interfaces
	Supervised Session	System requires limited user interface admins can cast control
	Presentation Session	System requires advanced presentation apps or utilities
	Debugging Session	System requires distributed debugging interface and tools
	Partitioned Session	System requires flexible server/control instances and clients
	Remote Session	System requires web-based clients for nodes and user interfaces
	Use-Cases	
	Data Exploration	System requires general, interactive data vis apps and deployment
	Social Analytics	System requires management mechanisms for layout control
	Design&Simulation	System requires 3D based custom apps and development
	Operation&Control	System requires app collaging and data streaming
	Research&Development	System requires integration of sensing and other devices
	Presentation	System requires presentation apps, utilities, also interactivity
Immersive Display and Management of Visualisations on LHRDs		
	Display	
	Distributed Rendering	System enables rendering on distributed displays
	Multi-Visualisations	System allows visualisations to be deployed on any layout
	Layering	System allows use of layers for apps and utilities
	Management	
	Unified Display Area	System provides the distributed displays as a single display area
	App Deployment	System allows deployment of apps to the display area
	States	System allows saving and restoring sessions
<i>continued on next page</i>		

<i>continued from previous page</i>		
	Testing&Debugging	System has various distributed testing/debugging tools
	Interaction	
	Management UI	System has an user interface for app deployment and control
	App Control UI	System has individual user interfaces for app control
	Casting	System allows casting of interfaces
	Device Use	System allows integration of devices for interaction or sensing
	Other	
	Middleware/OS	System needs a robust middleware and operating system
	Reliability	System needs to support reliability with variety of apps
	Inter-operability	System should support inter-operability with web-based projects
	Remote Operation	System clients should be remotely deployable
	Flexibility	System can work with different setups and technologies
Data Visualization Apps and their Development on LHRDs		
	Versatile	Variety of apps should be developable through the app framework
	Modular	Apps can be developed and work as plug-ins and can be instanced
	General	Apps can visualise general data with various configurations
	Interactive	Apps can be interacted/steered/controlled/configured
	Distributed	Apps are distributed and have server/node/control components
	Scalable	Apps can be scaled to many screens in any layout
	Extensible	App can use external libraries, data, system or APIs
	Customizable	Apps can use templates, styles, formats
	Web-based	Apps are web-based and rapidly developed
	Standardized	Apps follow certain standards to be subject to global utilities
	Reliable	App can be recoverable, re-deployable, clean on termination
	Casting	Apps can be cast to other interfaces
	Data-Driven	Apps can access, process, distribute data freely
	API-based	Apps communicate and coordinate through API, can implement API
	Collagable	Apps can be collaged and synced on parameters
	Multi-type	Different types of apps for use various purposes
	Interoperable	Apps can integrate various web visualisations or projects
Data-Driven, API based, Collaged Data Visualization System on LHRDs		
	Data-Driven	
	Data Availability	System supports access/host to various general data sources
	Data Processing	System supports processing of data on both server and clients
	Data Distribution	System support various distribution scenarios
	Data Sharing	System supports sharing of data between apps
	Load Balancing	System supports load-balancing between nodes
	API Based and Synced	
	Programming	Components can be commanded over API and coordinated
	Syncing	Apps can be synced on parameters and Collage
	Composite	Composite Apps can be created that visualise over others

Chapter 4

GDO Architecture

The GDO Software Framework is a novel approach to perform Immersive and Social Visual Data Analytics on LHRDs for data-driven multi-disciplinary research. In our survey to existing solutions in LHRDs. We have discovered that LHRDs software platforms in the literature were incompatible for this purpose, due to either lacking crucial functionality or capabilities, or they are purposed for other uses. Therefore, we have identified the problems these software had and explored the design landscape for implementing a new solution for performing Immersive and Social Visual Data Analytics on LHRDs. This solution is called GDO Software Framework. In this chapter, we would be introducing the architecture of this platform. Afterwards, in the following chapters we would discuss implementation, novel software paradigm that powers it and its contributions.

4.1 Infrastructure for GDO

GDO is a distributed software system that enables deployment of general data visualisations to the screens of LHRDs. It contains a management system that enables users to deploy visualisations with various configurations on screens and interact with them. These visualisations could be scaled up to tens of displays. Multiple visualisations could be deployed to screen-space in many different layouts.

Deployed visualisations could be synced in various parameters. Users of the system could interact with these visualisations through control interfaces on tablets and PCs. Finally, it allows developers to freely implement these distributed visuali-

sation apps using a large selection of online libraries web provides and with the help of distributed testing utilities that the system provides.

The system of GDO is composed of web-based node clients that run on each screen on LHRDs, web-based user control interfaces that run on tablets and PCs and finally ASP.NET [14] web project deployed on IIS web server that hosts the pages these clients display and registry of the system for coordination. This server also deploys SignalR websocket communication libraries[15] for asynchronous communication between clients and the server of the system which forms an API based middleware. In addition, the system runs PeerJS [117] libraries on clients for P2P communication using WebRTC. The visualizations in the system are distributed general and interactive applications built by external app developers. These applications run parts of their app logic on display nodes, clients and server and operate using API based messaging between their distributed elements. These applications work as plug-ins to the system software and thus the system is not dependent on any of them and can work without any application included.

4.2 Distributed Display Nodes

A LHRD is generally made of tiled-displays powered by computer nodes. In our software architecture, these nodes run web clients on each screen rendering the part of the overall screen-space of tiled-display. Hence, collectively these display nodes make up the unified distributed screen-space of LHRD. The node clients in the system are web pages/interfaces running inside kiosk mode web browsers on each screen. These web pages are hosted by the web server of the system. Each has javascript libraries that serve as the client-side of the system software. These libraries communicate with the server of the system through websockets for distributed and coordinated operation. Through these, the system is able to load and unload partial visualisation apps on each node. For example, in an LHRD with 16 nodes (4x4), a visualisation app can be deployed to top left corner in 2x2. Each node in that 2x2 grid in communication with the server loads 1/4 of the visualisation that is being deployed and thus collectively make a 2x2 distributed visualisation. Other than loading and unloading partial visualisations, the app logic inside these nodes communicate with the server logic of the app to render dynamic changes to the visualisation.

There are several reasons why web technology is selected for the front-end of the system. First of these is flexibility and ease of deployment. This enables site admins of LHRDs to launch GDO by just running full screen web browsers without any software installation. Secondly, web-technologies are easy to develop on, in comparison to other visualisation rendering techniques in general, such as OpenGL based solutions, which we mostly encounter in 2nd generation of LHRD software. Furthermore, large amount of visualisation and supplementary libraries can be found online for composing LHRD based visualisation apps. Through these properties, web-based front-end enables ease of development, lowers the bar on skills' of developers that can develop apps, enables rapid development of apps for visualisations that may be required in a short amount of time, and through large availability of online web-based libraries, makes the process of providing versatility in visualisations easier. Moreover, it provides interoperability with existing web-based visualisation systems or projects and enable easy integration of them to LHRD use.

4.3 Server Software

The server of the system is an ASP.NET web project that runs on IIS web server. Server-side uses C# as programming language. The task of the server-side of the system is to coordinate distributed screen-space and other clients, host app deployment mechanisms and run back-end of the visualisation apps. Server utilizes a websocket library, SignalR, to enable clients of the system to asynchronously communicate with the server. This means each client also runs a javascript client-side library of signalR inside their client-side system logic to enable client-side of the communication. Thus, server communicates with clients through API gateways built over signalR websocket communication. This enables client-side elements and server-side elements to communicate with each other through asynchronous messaging. Server also hosts a registry of distributed components on the system and their states. This registry is used as a coordination point, where clients of the system determine their current action or function based on records of the registry that is distributed to clients. Server dissipates these records reactively or proactively and as a result clients coordinate their operation.

We have selected ASP.NET as server-side technology for stable operation, development easiness, plug-in mechanisms, organization of code, performance, and

availability of database drivers and data processing libraries. First of all, through MEF [100] and Autofac [17], modular distributed plug-in style apps could be developed and loaded to server. In the server, the server-side logic of apps can run as threads and safely managed, they can utilize more hardware resources through multi-threaded approach or by use of many libraries .NET provides. Through these, they can offer better performance in general for a logic tasked with coordination and potentially data processing. Moreover, this infrastructure makes available drivers of many data sources on .NET to be used for accessing and processing data. ASP.NET web front-end also allows server to host client web pages used by node clients and user interfaces. SignalR websocket library enables asynchronous communication, provides different broad or group casting options. Through this middleware commands can be dissipated, data can be streamed, or files can be sent to modular distributed elements of the system.

In conclusion, server hosts registry of system elements for system level coordination and server-side app logics of deployed apps for app level coordination. Furthermore, it hosts web pages for clients to display. As a result, any web browser can connect to the server and become a client of it, such as being a node or user interface, start communicating with server through websockets and do their duties in overall LHRD distributed system.

4.4 User Control Interfaces

System also provides web interfaces for control and management of the system, which is another important requirement identified in the previous chapter. Through these interfaces, users can manage the system, deploy visualisations, interact and steer deployed visualisations. These web interfaces run on tablets and PCs on web browsers. Their infrastructure is similar to node clients. They run javascript libraries of the system in communication with the server through websockets.

Through opening the management web page on user controlled devices, users can connect to management interface of the system, which is hosted on the server. This interface runs similar to node clients where it is connected to server API through web sockets over signalR. Therefore, current system state can be received from the server and user actions can be sent to the server registry. Furthermore, similar to visualisations deployed to nodes, an individualized control and interaction page can

be loaded for deployed apps inside management interface. We will denote this as the control interface of an app. Thus, the users can control apps alongside managing the system through management interface.

As for the software selection, use of web technologies on this front-end enables developers to easily develop interactive and dynamic control pages for their apps. Developers can benefit from interaction and visualisation based web libraries. These pages can be individualized for deployed instances of apps and be dynamic rather than being static and having common controls throughout all apps. Furthermore, control page of apps may require scaled down version of the visualisation for steering and other purposes. Common libraries can be utilized for this purpose. Such as a maps app can show the full sized partial map on node page and interactive global map on the control page using most of the same code and resources on both types of clients. Moreover, the web-technologies enable all these functionality without any prior installation, as they can run on any browser.

4.5 Visualization Apps

System enables visualisation apps in form of plug-ins to system. These apps are distributed and they run their logic on server, nodes and control clients. So, they have web-based control panels, web based node clients and server-side logic for coordination as a thread which is created on demand. The system enables general and interactive apps that can accept various configurations and data and display the data in that configuration for that specific type of visualisation. For example a graph app can be deployed with given data and given configuration which determines what it visualise and how that is visualised. Furthermore, apps are scalable to multiple screens in LHRD and any location within the tiled display grid. Apps in the system are instanced, where multiple copies of the same app can be deployed with different configuration and data. They are also interactive as users can interact with different instances of different deployed apps through user control interfaces of those instances. Through these, users can configure, interact or steer the apps.

Due to the developer freedom, apps can use many web libraries on client-side. These can be for visualisation purposes or other. They can utilize ready to use visualisations libraries for mapping, graphs, charts, 3D rendering. They can use other supplementary libraries on clients. They can use interaction related libraries

on control interfaces for user interaction. They can utilize data access drivers and libraries on backend to connect to various data sources. They can also do this on client-side for accessing data. Due to the distributed nature, apps can use many data acquisition, processing and distribution scenarios as for their needs.

Apps operate through running a server-side logic as a thread for coordinating distributed app logics on clients, such as rendering visualisation on nodes or enabling users to control through individualized control interfaces for each instance of deployed app. These distributed elements of a deployed app again communicates through websocket libraries provided by system logic of GDO. Apps also utilize ASP.NET to host their javascript code and web pages of the apps such as control page and node pages. Apps again do not require prior installation on clients as their pages and libraries transferred through HTTP as part of loading web pages and their data and commands are transferred through websockets.

4.6 Clustered Visualisation Services Paradigm

In the system of GDO, we are operating with large number of node clients which are where app components are dynamically loaded, interactive and dynamic control interfaces that loads and runs components based on user decisions and coordination server logic that hosts system registry and logic and resources of deployed apps.

This is a diverse and dynamic distributed system with many logically distributed mechanisms on many physical locations. Most significantly many functionalities of the system is generated by dynamic composition of deployed apps and user interaction on distributed components. For example, different distributed components could coordinate to provide on-demand functionality like collaging visualisations, where multiple visualisations' logics coordinate with each other either directly or through a parent (composite) app; or displaying a scaled visualisation where data constantly streamed, or a presentation app that uses other app layouts as slides and changes are based on user interaction, and many other.

In order to enable such system functionality, many logical and distributed components should be able to have structured communication between each other, which they use to create combined functionality. This is enabled in GDO through API gateways on physical elements of the system such as server, nodes and control

clients. However, further mechanisms was required for multiple distributed logics to coordinate on the distributed system on component level. In our system we call this Clustered Visualisation Services Paradigm. This mechanism enables components of apps and system to communicate with each other locally or remotely within the system through APIs and each of them to have their own scope of operation. As a result, each distributed location have components of system and apps which have their own scope, can receive API commands and can send API commands to any other component on the same location or another location.

This enables, distributed and dynamic system logic, distributed app logic, coordination and scripting between apps and system elements thus enable us to overcome many challenges listed in the previous chapter which we will discuss in depth on the 6th chapter.

4.7 Conclusion

In conclusion, this chapter demonstrates our architecture for providing a software framework for visual data analytics on LHRDs. It introduces our approach for distributed display of LHRD where we utilize flexible web technology on display nodes to create a distributed display. Furthermore, our approach for server-side of the system where we create coordination point for both system and apps and provide API based middleware. Moreover, it shows our novel individualized user control interfaces of apps for users to have more extensive control over applications. We also introduced our app paradigm which is general, interactive, scalable and contain many other attributes in order to provide desired data visualisation functionality. Finally, we introduced our Clustered Visualisation Services Paradigm (CVSP) to enable advanced functionality discussed in the previous chapter such as data-driven collaged or composite application paradigms which provides API based communication between local or remote components within the distributed system.

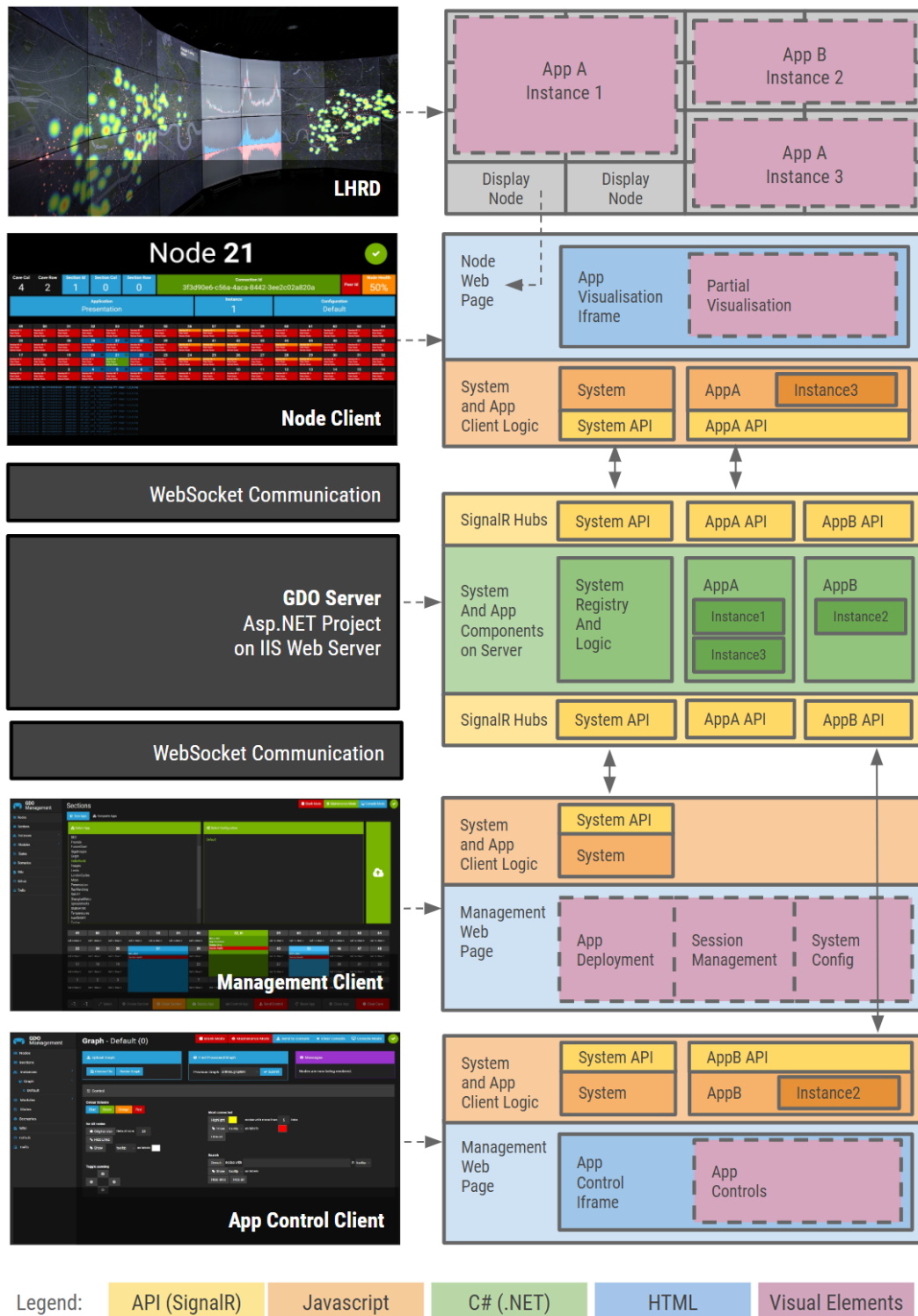


Figure 4.1: Project Architecture consists of a ASP.NET Web Server communicating and coordinating Node and Control Web-based Clients through SignalR Websocket communication where apps and system logic exists in a distributed manner and loaded dynamically.

Chapter 5

GDO Implementation

In the previous chapter, we have introduced the architecture of GDO Software Framework. In this chapter, we will be discussing the implementation of GDO Software Framework. Our approach in this chapter will be dividing each core functionality and discussing how that functionality is implemented. These are: Creating a distributed display area on LHRD using web-based node clients, Management system of the software and app deployment mechanisms, Saving and restoring system states, application paradigm, instancing of applications, controlling and steering applications, types of applications and development of applications. We will leave the main contribution of the thesis, Clustered Visualisation Services Paradigm (CVSP), to the next chapter.

5.1 Creating a Distributed Display Area

One of the main features of implementation is creating a unified display area out of distributed tiled displays connected to computer nodes. Usually, each of these computer nodes power one or more displays. We will refer these as display nodes. In GDO system, each of the display nodes run a web page which the server software of GDO hosts. This is the node web client. This client is responsible for rendering that part of the display area in LHRD.

This page loads the client-side logic of GDO as javascript libraries and run them. These libraries are in communication with the server and listen to commands on their individual API gateways we created through websockets. Through system messages over API, system can command the state of this web page on each node.

This state can be vacant, where page displays a blank web page or just the node id; or it can be in app state where the page displays partial visualisation of an app in that node. Also, it can be in a maintenance state where debugging information is displayed as an overlay.

As a result, GDO creates a unified screen-space by distributed displays of web pages, which are controlled and coordinated by the server logic of the system in LHRD through web-socket communication. Server can tell any group of nodes to go to the app state and start displaying a visualisation, where as a result each node in that group displays a part of that visualisation.

5.1.1 Display Nodes

A node client is composed of a general web page and javascript libraries of the system software, which both hosted and provided by ASP.NET server of GDO. A display node runs this web page inside a kiosk mode web browser in full-screen and provides it an id as URL parameters.

After a display node initializes with given id, the system libraries on the node creates a local registry and object store of system information regarding node itself and other data such as overall layout. These libraries also host a signalR client (javascript) tied with a client-side API gateway of the system. Through this, node and its components can call defined functions on the server-side API and server can call defined functions on the node API. Thus, server can send information and commands regarding the node, system operation and app deployment. These messages are translated into actions which updates the local node registry. Most significant of these actions is app deployment. When server commands a group of nodes to deploy a visualisation app through messages, these nodes load app libraries and load app visualisation page from ASP.NET server and deploy them to a maximized iframe inside node web client. As a result, the nodes in that group can start to visualise their partial visualisations of the app and make the overall visualisation through the collection of those partial visualisations.

In Summary, we can deploy various visualisations in many layouts to web based node clients across the LHRD through deploying app visualisation page and libraries into node web page on nodes.

5.1.2 System Registry

The system through its registry knows the location of a node in overall node grid, since every node upon initialization registers with the server with their unique ids. Furthermore, if the node is part of a visualisation, the server knows which part that node have to visualise. Therefore, through using this information, app libraries on each node can be commanded regarding which app and which part of an app they have to render.

For example, in a graph app with 100 graph elements rendered in 4 displays, each node client (app logic in node clients) loads a graph visualisation and then display graph elements in their corresponding sector. Thus, making a unified visualisation out of partial visualisations. If an app logic is designed to operate like this, it does not matter which nodes you deploy the app or how many nodes you deploy it to. Apps can be deployed to a single node and display all 100 data elements or 64 nodes where each node displays data elements in their sector.

Through this mechanism, we achieve heterogeneous layout of apps. So in a setup of 64 displays. 1-64 apps can be deployed in various layouts to screens. The server groupcasts to nodes of each app to load that app in given sections. Each group of nodes (per app) loads locally their partial visualisations. Therefore any layout of data visualisations can be created with simple web pages on an LHRD in an on-demand fashion.

Furthermore, this is possible in a flexible way, since there is no need to install any software in any of the clients. All they have to do is connect to ASP.NET server with an id. Node page and system libraries are loaded through HTTP, communication and coordination is achieved through websocket and a visualisation is loaded on that display; html elements and javascript libraries of that app can be loaded again through HTTP from the server and deployed to the node. That partial app again communicates and coordinates through web-sockets.

As a result, almost any type of visualisation can be deployed dynamically and in an on-demand fashion to tens of browsers in an LHRD in any layout, thus enabling users of LHRD in creating various visualisation scenes composed of heterogeneous visualisations.

5.2 Managing the System and Deploying Apps

The system of GDO consists of a server software that manages the distributed system deployed to clients in LHRD. This system is controlled by user control interfaces. This management interface is a web page where users can manage the system, deploy visualisations, interact with visualisations, and use utilities. These pages can be opened in any tablet/PC within a network that can reach the ASP.NET server of the system.

These pages contain status of screen-space such as layout of nodes and their status, live logs of system and a terminal for executing commands. Most significantly, it contains a canvas for selecting and deploying apps to group of display nodes in the screen-space. This is performed by a user selecting an app and its configuration and deploying it to any selection of nodes in the screen-space. Equally important, this interface contains individual control interfaces for deployed apps, where users can interact with app instances they deployed on the screen-space. Finally, this interface contains other utilities such as saving states of the system and executing scripts.

In order to enable control over the distributed system, the system software on the server hosts a registry. This registry holds the system state. This state changes by user actions and app events. As a result, any client can request current registry and load according to the current state. For example a node can join the system late, query the server regarding its current state, such as node status and app deployment status. Then it receives an update from the server with this information, which enables it to load accordingly. So, if there is an app deployed to a section containing that node, node loads app libraries and start displaying corresponding partial visualisation. This is same for the management interface, where it queries and receive the current state of the system and load its UI according to it, thus showing up-to-date information on the system elements in the UI.

Clients and server share this information, send commands and actions through an API based messaging system built over signalR websocket communication which shapes our middleware. On the server, server hosts a signalR hub for system API gateway and hubs for each apps API communication. On these hubs, there are defined functions for API based operation such as deploy app, clear screens, load parameters. App hubs can define their own functions too, this is why server and each app have separate hubs. On the client side, signalR library is registered with

client-side functions of the system and current selection of included apps that the server can call. Again each app adds its own functions to client-side too. This gives independence to server code and app codes, while enabling app developers being able to implement their distributed functionality.

These functions on server and clients are connected to their corresponding registries. On the server, system hub is connected to system registry and app hubs are connected to server-side logic of apps. On the client-side, system defined functions are connected to client-side system logic and app defined functions are connected client-side app logics. Thus, when a message is received such as activate X on server hub, it changes registry of the system, take corresponding action and distributes these changes to clients. Each client then changes its own local registry and take according action. For example, in activation of maintenance mode, clients load maintenance overlay on top of their web interfaces using this method. The server may also chose to unicast commands to clients, groupcast or broadcast them. So a server can respond to a query of a single client or cast an action to group of clients. This enables server to respond or command whichever clients it chooses to.

Based on this general paradigm, node clients register and periodically ping the server, control interfaces receive current system state and allow users to take actions according to it, through user interfaces apps can be deployed, modes can be activated, system states can be stored or restored or scripts can be executed. On each of these activities, either through UI or logic, API of system can be called, which takes corresponding actions, update registry and distribute commands to clients.

5.2.1 Core Communication Cycles

This is a proactive mechanism that enables clients and server to be aware of the states of all clients. It enables mechanisms that require system state to operate and run. Nodes register and fill the server registry with their status as they load. This is done by nodes sending an update regarding their status to server hub. After initial registration, they continue to update periodically. This periodic update is spread to a time window, so it has balanced use on network resources and server hub. Server accumulates this data in registry and broadcasts back to clients. As a result, nodes and control clients have up-to-date state of nodes and the system.

Clients who receive this update changes their own local registry and take

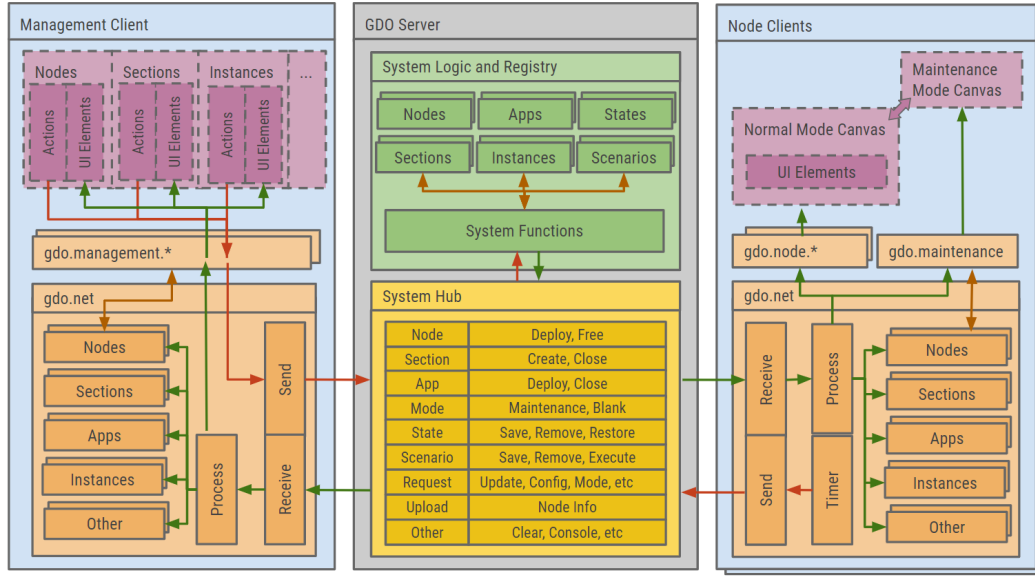


Figure 5.1: General Communication Schema for System Management

actions. So, this update may trigger events the node have to take. For example, if update includes an app deployment on that node, than this turns into action, where it results in node loading app libraries and app visualisation page. Also, app and other system components on the node may require this data later for other uses. For example, an app logic without querying the server can learn state of adjacent nodes by looking into local registry.

Other than periodic updates (that includes the most recent event), newly joining nodes may require current state of the system. Therefore, whenever a node joins to system, it requests an overall update and server sends a unicast with latest copy of system registry (serialized) to that the node. Thus each client joining now or staying on the system is always up to date local registry on the system.

Through this mechanism, user control interfaces show up to date state of the system, thus preventing logic or UI elements from taking actions that cannot be performed or errors due to non-existing elements or already changed states. So a user cannot deploy to a section of nodes which is already part of an app. This also enables live display of system status on maintenance pages in nodes and on management interface in user devices with live logs of the system.

5.2.2 App Deployment

Another management functionality system implements based on this server-client paradigm is deploying apps to nodes. Users through UI or other apps through utilizing API can deploy and terminate apps. This is a two-step operation. First is creation of a section. Section is a logical group of nodes that are put into the same signalR groupcast. So when server receives command to create a section, it puts all nodes in that section into the same communication group. Server can receive this action by a user through user interface, or app logic on other clients.

Afterwards, users or app logic can select to deploy an app to that section. They also have to include which configuration that app will use. After receiving this command, server through an app factory creates an app instance for that app with selected configuration and broadcast this as an update to clients. Unrelated clients will only update their UI to reflect changes regarding that app deployment. While nodes in that section with that command will load app libraries and request the selected configuration data. After receiving the configuration data, these nodes will initialize node components of the app with parameters on that configuration.

This process loads the app visualisation pages into that group of nodes in that section within an iframe that is maximized. Each node also loads the app libraries with given configuration. Similarly, upon receiving terminate command on the server hub, server destroys the corresponding instance and sends all clients of that app instance with commands over API to stop visualising that app. These clients unload app libraries, clean scopes that app uses for temp/data variables and unload DOM of the iframe and clean DOM elements app loaded.

While an app is active on a node, it is given a web page within an iframe and a scope of operation in local registry. Apps may choose to load code to their given scope or they can load libraries and code to DOM of their iframe. They are also given access to local registry and signalR functions of the node. As a result, local app logic operate using these local resources and information plus remote communication to server over API to visualise their partial region of the app that node needs to visualise.

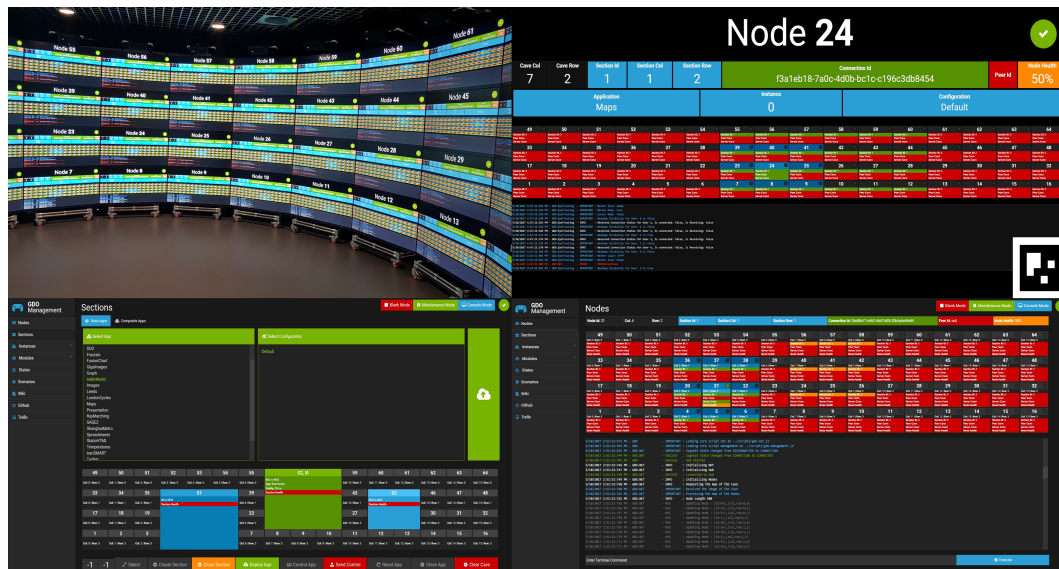


Figure 5.2: System Management Interfaces: Node Clients in Maintenance Mode (Top), App Deployment (Bottom Left), System Health and Config (Bottom Right).

5.2.3 Modes

Another system functionality is system modes. Modes are built above client side web page states mentioned in 5.1. These are general modes of the system that can be activated from management interfaces, such as maintenance mode, minimal mode, normal mode and etc. The way they work is: server activates a mode upon user interaction and broadcasts the activation command to all clients. All node clients after receiving this command show appropriate mode. For example in maintenance mode, a debugging interface is shown on each node; while on normal mode, it shows node ids for vacant nodes and app visualisations for deployed nodes.

Most significant of these modes is the Maintenance mode. In this mode, nodes display debugging interface that shows various information regarding the state of the node such as its location, app deployment status, its configuration, or other information like connection id or node health. It also shows a live log of on-going events such as received commands or app events. This enables developers to track their application in a distributed display area and identify problems during testing.

5.2.4 Terminal

Similar to the maintenance overlay on nodes, another terminal is shown within management interface, showing events of the system. Unlike the ones on nodes, users can execute API based commands on this terminal. These can be local commands, server commands or relayed execution where server relays a command to be executed on another client or clients. This happens through messaging over API between client-server. Through this, developers have advanced debugging functionality where they can execute commands on all parts of the distributed system during testing.

5.2.5 States

System also have a state saving and restoring mechanism for overall LHRD. The state can be stored in a configuration that keeps the location and size of apps. Thus, admins can save states and restore them later on. Admins can store different states or set of states for each repeating session or save states prior to a new session, prepare session and use the state in the session. States mechanism also operate over API. While saving, it looks at the server registry and save its state. While loading through UI, it sends server to restore state command. Server sends clear LHRD command and load apps with configuration at given locations and sizes according to the state. Through states, admin of the system can store layout of apps for different sessions and use within the desired sessions.

5.2.6 Scenarios

Scenarios is a scripting mechanism that works throughout the distributed system of LHRD. Users can create scenarios on management interface, save, load and execute them. These action scripts executes a set of commands on system API. These commands can be made iterable, or they can pause, wait user action, or timeout. These can be utilized to create sessions where selection of apps and what they are displaying can be programmed and interacted in a collective way. So, an admin can create a scenario to first load a system state, than change datasets of apps, then load another state. Admins can do this with timeouts, user inputs and with other parameters. This functionality provides many opportunities for admins to create complex and dynamic sessions that can be interacted on a higher level during the session. These commands the scenario executes are again based on API commands in the system, thus they can target server, or a client, or group of clients.

5.3 Visual Applications and Development

In GDO, LHRD displays visualisations through general, interactive and distributed applications. These applications are designated to run on any group of nodes, that we refer as a section, when deployed. Visualisations run partially on the nodes of a section and with coordination of the server they make up to a single unified visualisation on that group of nodes. A visualisation app in GDO is composed of three parts: control, server and node components. These parts operate in a distributed manner over web socket communication.

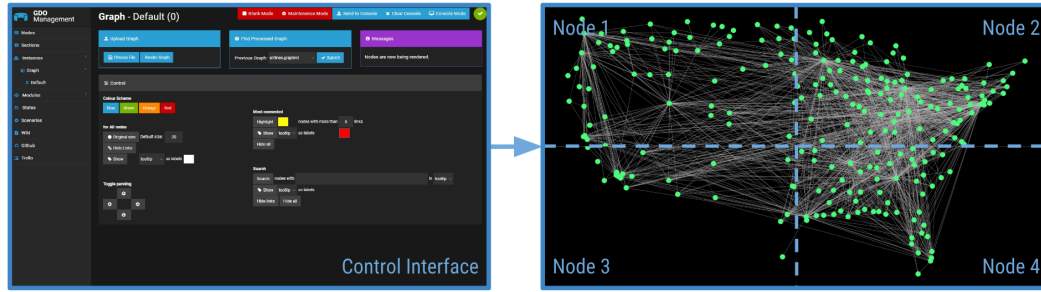


Figure 5.3: Apps on GDO visualise on a distributed manner through cluster of components on LHRD display nodes and can be controlled through web interfaces on tablets.

5.3.1 Node Component

The part of the app that is deployed to nodes consists of a general app visualisation web page and app logic library written in javascript. When an app is deployed to a display node in LHRD, the system logic of the node loads this general app visualisation page inside nodes web page as a maximized iframe. This page is similar to node page and hosted in the GDO ASP.NET server. Then the node loads app logic into the node and initialize it with parameters that provides the information to node component of the app regarding its location in the overall display grid. So the node knows where app is situated inside LHRD screens and where that node is situated. With this information node component of the app knows which part of the distributed visualisation it should render. It starts to communicate with the server component of the app and coordinate to render visualisation on that region of LHRD. By each node of that section rendering their corresponding partial

visualisation, a distributed visualisation is created in that section.

During operation, node component of the app communicates with the server component of the app through websockets to coordinate. They receive parameters, changes and data from the server component of the app. They can also load, download, stream data and resources from the server or external sources to visualise.

Lets look at an example of a multi-layer mapping application. Each node is provided with parameters such as zoom and location of the map that partial visualisation should be rendering. They are also given which layers are displayed currently. So each node component of the app loads the current map library with those coordinates, zoom and data layers. If these change during operation, server component of the app broadcasts new coordinates and zoom or layers to render. The data for the layers can be from the server or external resources on the web. As a result, a dynamic distributed map visualisation out of general app visualisation web pages in iframes in distributed displays can be created.

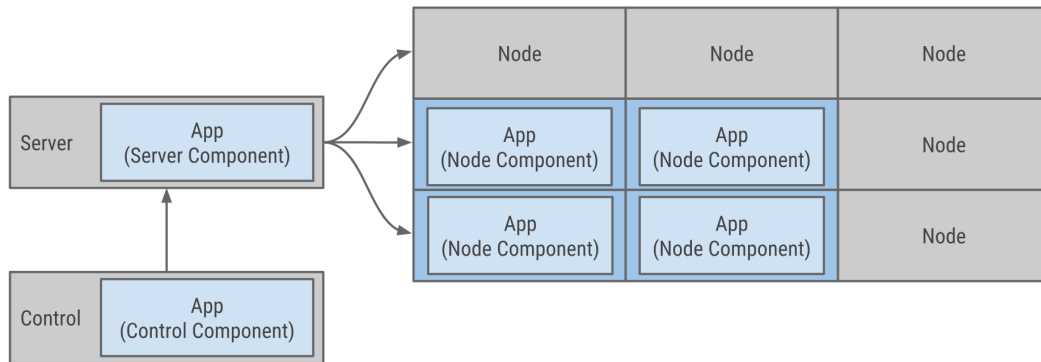


Figure 5.4: App Components on Server, Display Nodes and Control Clients.

5.3.2 Server Component

The server component of the app is a class in the system registry. This app class is a factory for instances of that app. It creates and destroys instances and keep track of them. Therefore, GDO can deploy many copies of a single application to different sections on LHRD with different configurations through instancing mechanism.

When an app is deployed to a group of nodes, the system creates an instance of the app. This instance is a thread that runs a copy of app's server side logic with given parameters. These parameters are the location and size of the app in GDO and configuration of the app. Location and size enables server side logic of the app to coordinate distributed visualisation on the group of nodes in LHRD that the instance is deployed. Configuration enables apps to be deployed with saved or created templates.

App configurations are generic json files that enable app developers to store states of the app which can be retrieved and used in other deployments. They can be very simple or very complex which is up to the app developer. For example in a 3D app, a configuration can store which scene is used, which models to be loaded, what are the user preferences or where is the current view located at. When a 3D app is deployed with this configuration to any section of LHRD, both server and clients of app logic is provided with this configuration and they can load the libraries with these states/parameters.

Through these mechanisms, we are able to deploy multiple instances of an app with different configurations in LHRD. Enabling this configuration data to each instance on start-up allows us to deploy apps with more extensive parameters on each instance. It also allows state saving and restoring mechanisms to work, since we can save state of an app as a configuration and later reload it on another session.

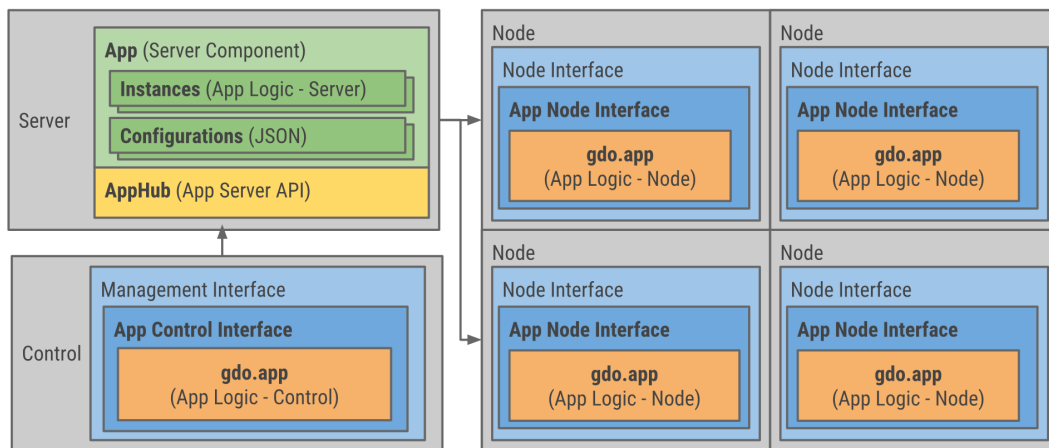


Figure 5.5: Implementation of Distributed Applications on GDO

Another component of the app on the server is the API hub of the app. We have multiple hubs on the server, one for system and one for each app. This allows app developers to implement their communication schema between clients and server side components of the app. And thus implement a distributed application. Developers can implement various distribution, rendering methods between clients/server as that type of app may require. For example, App hub of an image app may implement API with actions such as load image, zoom image, pan image on the server-side; and client-side API can implement actions such as receive image part, pan and zoom. So a control client can load image to server and then server can send partial images to clients through receive image part function, also control client can call zoom image on the server and server can calculate coordinates for each client and can call zoom functions on those clients' API.

There is only a single hub per app due to restrictions on signalR libraries. SignalR only allows creation of hubs on the ASP.NET server start-up. Thus, the only information server has on start up is which apps are included with the solution. Apps overcome this restriction through relaying messages to dynamically created instances. Therefore, dynamically deployed instances of app are reached through their common app hub for that app. They perform this through relaying the commands to corresponding instance of the app by specifying instance id. For example, when an user interface over API tells the image app hub to load an image, it refers to which instance to execute the command at. So, if there are more than one instance of the image app on the LHRD screen-space, server-side components of those instances can be targeted individually. The same goes for client-side API.

This restriction is turned to an advantage where it enables us to target an app in general or its instances individually. For example, there is a data animation app with multiple instances, where the animation can be started, paused or stopped. A user can target the app in general and command to start animation, where all instances are targeted in this case as a group and all of them start the animation. Or, individual instances can be targeted to start or stop separately.

Overall, apps are plugins on the server represented as app classes, which has the task of creating and destroying app instances of that app on-demand, where these instances are working copy of applications server-side logic in a thread. These classes also host configurations for those instances and have a hub which implements

server-side API functionality of the app and acts as a relay to messages coming in and out from clients and relays them to corresponding instance or group of instances of that app. Using this functionality, app developers can create coordination logic on the server. This logic can distribute parameters to nodes, process data, respond to user actions coming from user interfaces and many other functionality.

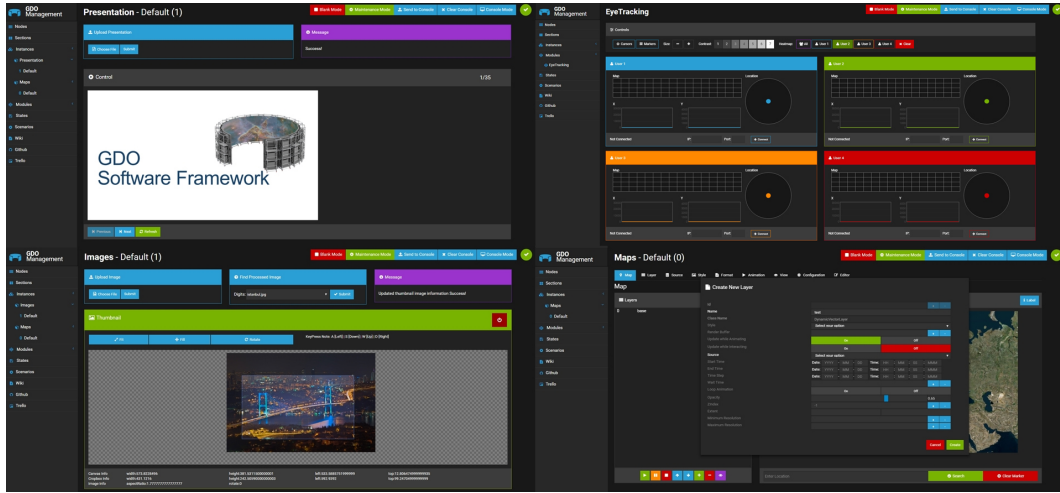


Figure 5.6: Various App Control Interfaces Displayed Inside Management Interface

5.3.3 Control Component

Application paradigm in GDO allows control over the instances of apps. These instances can be configured/controlled or steered through user interfaces. These interfaces are loaded dynamically to management interface. They are similar in nature to how we load app visualisation pages inside node clients. In this case, we load app control page into management client in an iframe. Through this interface, users can control, configure or steer each instance of each app individually. This interface is also designed and implemented by app developers similar to app visualisation page. It is also very similar in structure to app visualisation page, where it uses common app libraries with node component of the app.

When a control interface is opened in a management client. Client in an iframe loads this general control web interface of the app with given parameters. It uses the same javascript library of the app, but it initializes to act as a control component. The reason it uses the same library is for code re-usability. Both web

pages of app (control, node) tend to use similar elements commonly. For example, in a mapping application, the app uses a map library to render partial map in the node, and in control interface again uses same mapping library to render whole map but scaled down for interaction. Instead of forcing developers to duplicate work and implement two separate libraries, we have chosen to use one library as entry point and if developers chose to use two different set of libraries, they can select which libraries to load in that entry library depending on if the client is a node or a control. Otherwise, they can use the same libraries with different parameters.

Control interfaces are interactive web pages that contain control/steer/configuration web elements that users can interact with. For example, in mapping application, a whole map with navigation can be showed where users can select dynamically the visual style and data layers. For charts application, it can be template and data selection on control interface, which is up to the developer.

Through these, users can control the app. This happens through users actions being translated to API commands that control interface calls through our websocket API middleware. For example, a user moves the map and makes a layer invisible. This interface calls corresponding functions for new coordinates and setting layer visibility on app hub on the server for that instance. App logic of the instance on the server makes changes and distributes changes to other relevant clients such as group of nodes displaying the instance and control interface controlling the instance.

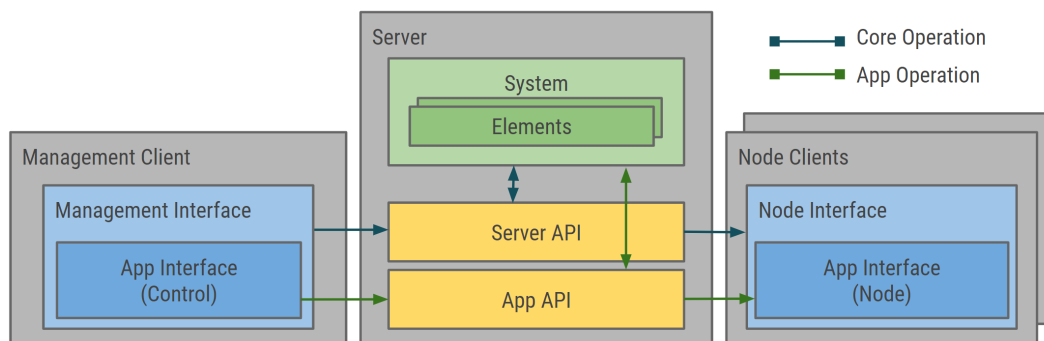


Figure 5.7: Base Communication of System and App Operation

5.3.4 Communication Schema

Apps distributed elements communicate through API, where on the clients and on the server, app developers implement API functions that other elements can call. On the server, a signalR hub is implemented for this functionality. As discussed before, app developers can implement overall functions that target all instances of the app, target group of instances or target an individual instance. On the clients, app developers implement another set of API commands for client-side functionality. These functions again can target the app as a whole or instances of it separately. Also, they are divided by control functions, node functions and common functions. So, API functions of a control panel may differ from a node client. Also they can have common functions that are used both by node and control clients of the app.

For example, a basic video rendering app. The API of server may have functions like load video, start, stop, pause. The control interface may have API for receive video list from server, receive overall video. The nodes may have receive partial video function. And since there might be multiple control interfaces, start, stop, pause may be common API for both nodes and control interfaces. Thus when one of the control interfaces stop the video, other control interfaces as well as nodes of the app receive the stop command over API. As a result, these three components of the app and its instances can communicate with each other using API.

5.3.5 Instancing, Configuration and Layering Mechanisms

App instances are used to enable deploying a working copy of the application at any section of nodes in LHRD. To reiterate, through instancing paradigm, we can have multiple copies of an application running on different sections without any conflicts. On server side, each instance have its own elements, since we instantiate the class developed by the app developer on the server and use corresponding App Hub to redirect requests to those instances. And on client side, global app variables, objects, functions are stored under a scope dedicated to that app on nodes and control interfaces of the app. However any temporary variable, object or function is stored under scope dedicated to that instance on client-side and this is initialized when the instance is created and is destroyed when that instance is terminated. As a result, we provide individual and common variables and functions to applications without any overlap, while individualized scopes to instances to operate. Following

sections tries to explain how multiple apps and multiple instances of the same app, or multiple layers operate inside GDO.

A. Multi App Operation: In case of multiple apps, each app has its own unique hub that is active all the time. When a management client deploys an app to a section, an instance of that app is created inside the App element on the server, as mentioned before. In this scenario, clients of that app connect to App Hub to communicate with that App Instance through their id. Afterwards, if clients have to be updated, the App Hub calls all clients on that section again with the id. So in case there are two apps, a chart app and a 3D app deployed to different sections of LHRD, each of them loads node components in their group of nodes, creates instances for their server component and have different hubs for websocket communication. Nodes of the chart app communicate with app hub of chart app which is connected to that instance of the chart app. Same goes for that 3D app. As a result, instances of different apps can be deployed to server and can communicate with their node and control components without overlaps.

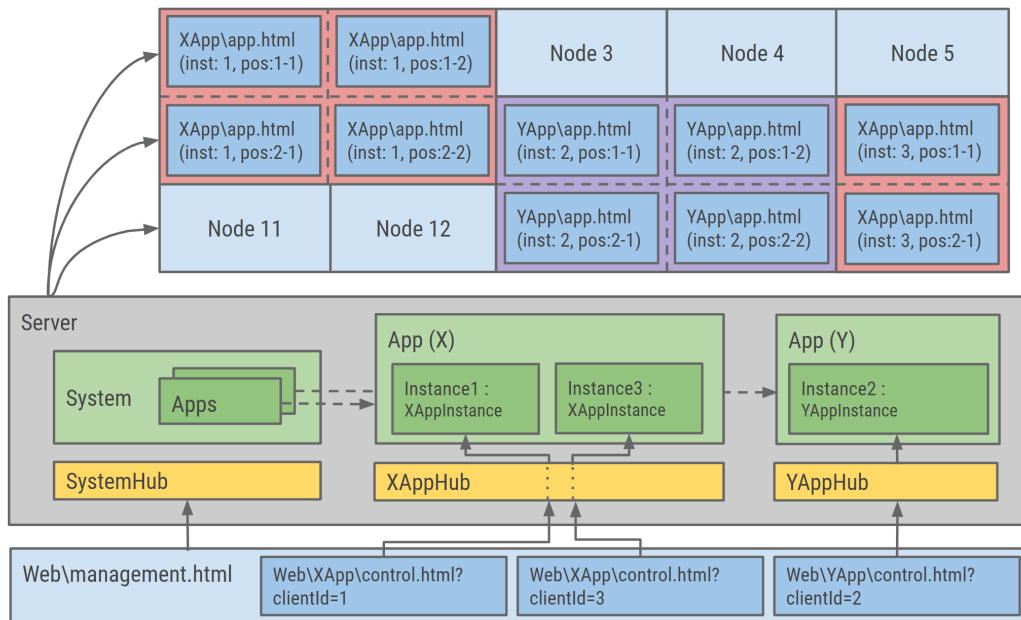


Figure 5.8: Multi-App and Multi-Instance Operation Differences

B. Multi Instance Operation: In case of multiple instances of the same app, server creates these instances in the app's factory as threads with selected configuration for each instance. Each control client and group of nodes belonging to a instance access to their corresponding instance through the common app hub for that app. Their messages are relayed to that instance. On the client side this is similar, global app objects and functions stored at app scope, and instance specific variables are stored at instance scope, and API relays actions and change variables accordingly.

So at no time, different instances' variables overlap. In case of visual elements, that app visualisation pages are loaded as iframes and when an instance is closed those iframes are destroyed too. Also instance scope is cleaned as a result. Furthermore, external js classes loaded inside iframe, unlike the app specific objects loaded to gdo structure (local registry) are destroyed with the iframe. Neither in the web pages document nor in javascript structure (local registry) any instance specific or external remainder of a instance is left over after termination. Therefore a client (node or control) becomes ready for deploying a new instance of another app.

As a result, if a video app is loaded with two instances, there are two threads on server, 2 group of nodes on the LHRD. Each group of nodes communicate with the corresponding instance through relaying over the hub. In each node, the node component of the app is loaded and given an app scope, an instance scope and have an iframe for rendering, where iframe and instance scope is destroyed when that instance is unloaded. Thus, two instances of the same app can be present and deployed to LHRD simultaneously.

C. Configurations: GDO has a configuration mechanism for apps. App instances are deployed with different configurations. Configurations are JSON files/data that app developers can implement a schema for. So, app instances read these configuration data on deployment and initialize accordingly. These configurations can provide differentiation in functionality between app instances in terms of loading with different data, templates, logic, format and other parameters. Configurations determine many aspects of that app instance. A map configuration can declare location, zoom, layers, data. A graph configuration can have template, style, type and data. Presentation configurations can hold slides, transitions, layout. Overall, apps can use config mechanism to hold data sources or links, file links if data is local,

current view parameters if app is navigable, other template and styles for data and many more parameters or data. Configurations can also work aligned with control interfaces. So, App X can be loaded with Configuration A. During operation, users can use the control interface of that instance to make changes. These changes can be saved as Configuration B and can be used in another session if app allows.

D. Multi App Layering: GDO also supports multi-layer app operation. This is enabled through multiple app pages being active working with different instance scopes. These app pages operate at different z-indexes, they can have transparency and can be switched or made invisible. Since instances have dedicated scopes and app visualisation pages are deployed in isolated iframes with separate DOMs for visual elements, this works without conflicts. Messages received from the server components are directed to given apps or instances. Moreover, apps can utilize layering of web technologies internally. Therefore, an app instance on a node can have multiple layers or overlays using html z-index since it renders over web technologies within its iframe.

5.3.6 Types of Apps

There are few applications types we have already implemented and these are: Apps, Modules and Composite Apps.

A. Apps: First type is apps. These are the apps we discussed up until now. They have three components, which are on server, control interface and nodes. They are instanced. They have configurations. They operate in a distributed manner through API. They are deployed through management interface. Each instance have their own control interface. They are deployed to the nodes through their node components and show distributed visualisations. When they are deployed, they have an instance created for them in the server. Although, their hub is always active. All other details regarding them are discussed in previous sections.

B. Modules: Second type is modules. They work in a similar manner and are structured similarly to apps. They are applications that are designed to run at background continuously and have a single instance. They are designed specifically as background services for external devices or data services that GDO uses. So other apps or users can utilize these external devices or data sources as a global

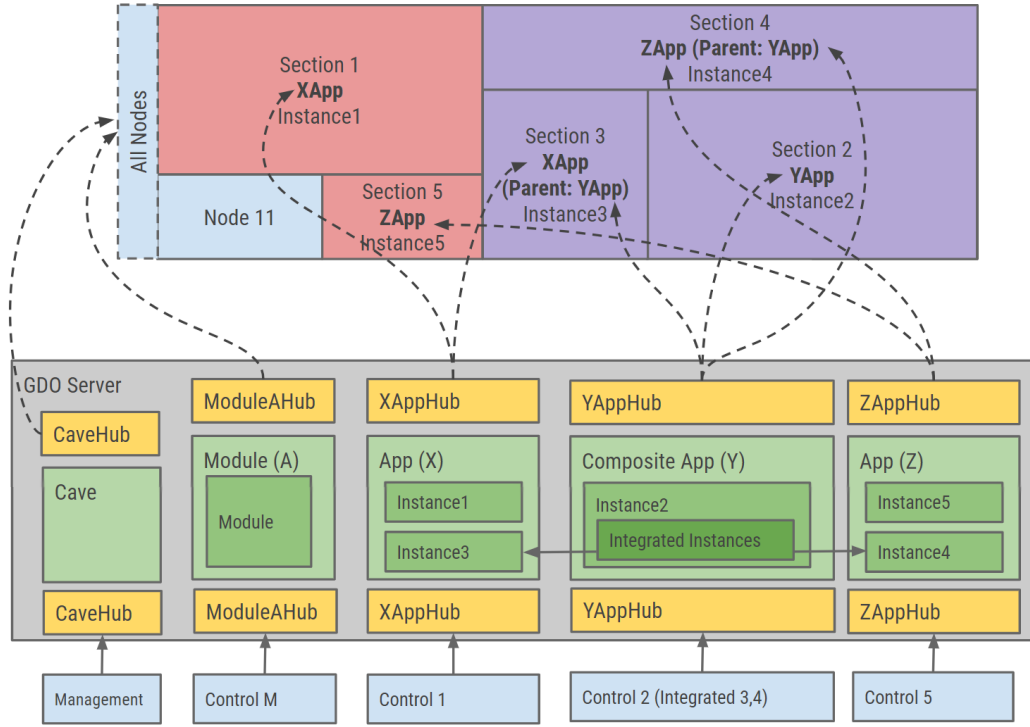


Figure 5.9: Operational and Distribution differences of Modules, Apps, Integrated Apps and Composite Apps.

resource within LHRD. They are stored in the server similar to apps but have a single and always active instance. They have a single control interface since they are not instanced. They can create overlays on all the nodes as visualisations. Similar to apps they have an always active hub, but instead of targeting multiple instances, they target always active module instance.

C. Composite Apps: Composite apps are a type of apps that use other apps to visualise data. Their main purpose is to use multiple apps in coordination to create a collaged data scene. They are similar to normal apps in structure. However, they load other apps to other sections. These children apps are given configuration by the parent composite app. Throughout their operation, children apps are controlled by parent apps. Parent app uses API to control instances of the apps it deployed. On a node where a children app is deployed, both parent and children app node component is active where they talk to children and parent app hubs.

So children apps operate normally through their node and server components.

Parent app server component can command server component of the children apps, node components of the children apps over API or it can command its own node component which can command node component of children app. Thus, parent app becomes a background coordinating system for children apps, through many coordination options.

In addition, parent app can retrieve data and process it and distribute to children apps. For example, parent apps control interface can be regarding data importing and processing . In the control interface of a parent app, it can also load tabs or panes that can contain control interfaces of the children apps. As a result, this mechanism provides coordinated and collaged visualisations to be deployed to GDO using composite app logic, where parent apps are tasked with coordinating and meta operation and data provision, where children apps are used for visualising the data parent app provides with parameters given by parent app. On children nodes, both parent and children node components exist. Parent node component can be used for data provision, coordination or as overlays where children node component is used to visualise data.

5.3.7 Development of Apps

App developers implement GDO applications through using .NET and web resources. An app developer only has to implement few main components to create an operational app: App Instance, App Hub, App Client Library, App Control Page and App Visualisation Page. Also depending on the skill of the developer, he or she can implement a distributed operation of the app between these, following our simple guidelines.

Table 5.1: Developer Template for App Components in GDO

Element	Language	Function
AppInstance	C#	Server Logic of the App
AppHub	C#	App API on Server that relays to Instances
App.js	JavaScript	Client Logic of the App
Control.html	HTML	Control Interface of the App
App.html	HTML	Node Interface of the App
Configurations	JSON	Configurations of the App

Apps are user created projects that are plugged into the software framework.

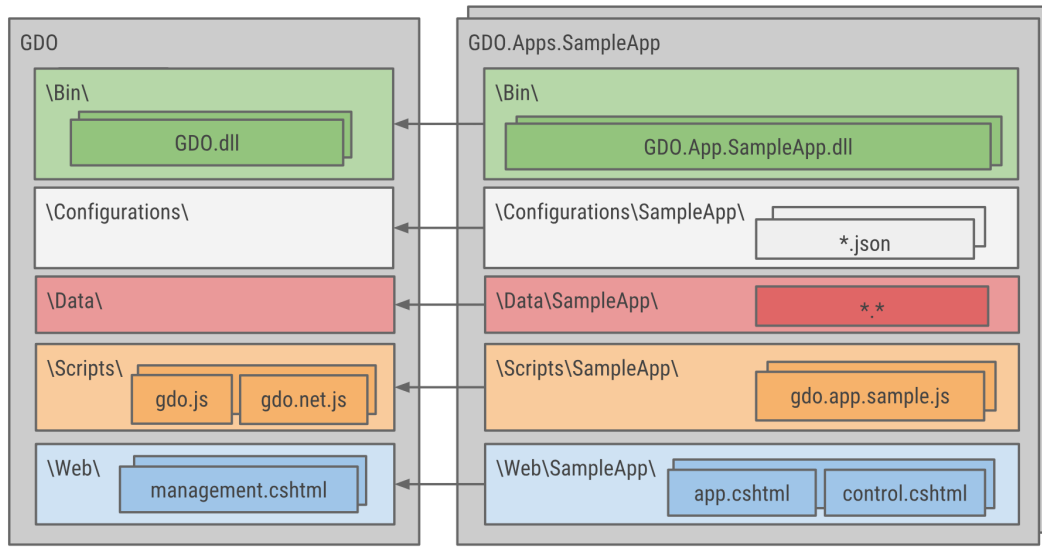


Figure 5.10: App File Structure

Core project does not depend on app projects and can run standalone. Any combination of apps can be loaded on the server during start-up, if they are present in the plugin directories. Apps have their own project folder which the framework looks for. From this folder, the project loads app executables and main files, configurations as json files, other libraries app uses, and also data from data folders.

Therefore, App developers also can use additional .NET libraries, js scripts and web pages as long as they put them under this folder structure and put data manually into project folders. As a result, apps can individually developed using web technologies and libraries freely on front-end, .NET libraries on the back-end, where all of these can be packaged as plug-ins that can be included with the overall project of GDO.

5.3.8 Performance of GDO

We have conducted a few benchmarks on the system of GDO [39] and also applied similar benchmarks to SAGE2 platform[127] under same or similar test conditions. For these benchmarks a mediocre computer with i7 3630QM processor and 16GB DDR3 RAM is used to host the server logic for these platforms. For benchmarking load on the server, we have tested for memory, CPU and network usages.

First Benchmark is memory and CPU usage of GDO and SAGE2 with different applications loaded. For this benchmark, no clients connected to both server logics. We have tested for memory and CPU usage, with base server deployment, with 4 different basic apps like (graphs, images, maps), and with 16 basic apps. It should be noted that apps on these systems are very different in nature, therefore we have to select similar ones but not exact. On all figures, Both GDO and SAGE2 had stable load without an increase while applications are loaded. GDO had 120MB memory usage on average while SAGE2 had 100MB on average on all test conditions. For CPU usage, during app deployment minor pikes occurred on both systems. Although after app deployment is completed, CPU usage was stable and it was under 5 percent for both systems.

On our second benchmark, we have tested for effects of node connection on the server. In terms of CPU usage, we observed similar stable CPU usage which was under 5 percent for both SAGE2 and GDO, independent from how many nodes are connected. However, on memory usage, GDO memory did increase on a logarithmic scale with number of nodes connected. On the other hand, SAGE2 memory usage on the server stayed stable around 100MB under same test.

Table 5.2: Memory Usage on Server for GDO vs Number of Nodes Connected

Num of Nodes	1	2	4	8	16	32	64
Memory Usage	120MB	127 MB	140MB	161MB	199MB	227 MB	265MB

Furthermore, we had a combined test for GDO. In this test, we have connected 32 nodes to GDO and loaded various apps to GDO. Under no nodes connected, GDO has 120MB memory usage and under 5 percent CPU usage. With 32 nodes are connected but without any apps loaded, this raises to 230MB, but still under 5 percent CPU usage. We have loaded a single app to 32 nodes. We also loaded 2, 4, 8 apps in different layouts to 32 nodes for the rest of the test. What we find is that 230MB memory usage stays stable with different app layouts, however there is minor increase in CPU usage, where this increase is still under 5 percent.

In terms of network load, we have tested GDO with no apps and no nodes, obviously resulted in 0 mbit/s on the server-side communication. We have tested with 32 nodes without any apps and we have observed 2mbit/s network usage for base operation on the server with nodes connected. We also tested with 32 nodes

and dynamic apps that span to all nodes with constant interaction, which resulted in 4mbit/s network usage on the server.

On the client side, both SAGE2 and GDO have similar memory usage. Base client of SAGE2 used 7.2MB of memory and base client of GDO used 8.4MB of memory. Since GDO and SAGE2 apps are very different, we can not have a detailed comparison under app usage. However, we have tested for the only similar app: maps, and we have observed that SAGE2 and GDO have similar memory increase, where SAGE2 memory usage increased to 12.4MB and GDO memory usage was increased to 15.5MB.

As a result, we have observed similar server and client loads on both SAGE2 and GDO with the exception of GDO minor memory usage increase on server-side with number of nodes connected. However, even with hundreds of nodes, this does not affect overall performance, because it is logarithmic and minor. On other aspects we compared GDO and SAGE2, we have found they have similar performance usage which is minimal as both systems are capable even running on medicore PCs. This was the expected results since 3rd Generation LHRD software focus is web technologies, and due to using web technologies, these platform are lightweight.

5.3.9 Conclusion

In conclusion, we create an unified screen-space on LHRD using basic web pages that are in communication with the ASP.Net server of the GDO through SignalR websockets. We have web-based control interfaces that users and admins can manage the system using the same paradigm. These control interfaces offer novel features specifically designed for visual data analytics management. In this system, we can include many externally developed apps that operate in a distributed manner. These apps have node components for visualising data, which are coordinated through their server component over API and have individualized control interfaces for configuring/interacting and steering the app. Apps operate on a distributed manner which developers can decide on. They have instancing mechanisms, different types for different uses. Apps are developed using web resources on front-end and .NET and C# resources on backend.

Through all these, we implement the basic set of functionality we intend to cover on an LHRD, this set includes, creating an unified screen-space, deploying apps

to any area in any layout on screen-space, have a management system for LHRD for visual data analytics, have a distributed app paradigm to operate general, modular and interactive data visualisation apps on the system and application development framework to develop these apps. However, we have not went into detail to our middleware and framework which also enables collaging and syncing apps, various data distribution and processing scenarios and many other functionalities. In the next chapter, we will discuss this from a different perspective. We will introduce this mechanism as a software paradigm and list all the contributions it enabled.

Chapter 6

Clustered Visualisation Services Paradigm

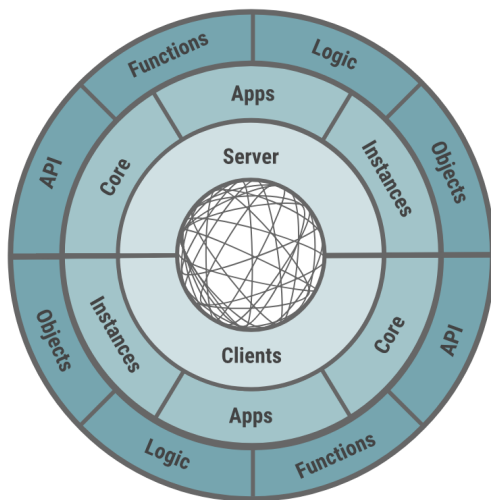


Figure 6.1: Clustered Visualisation Services Paradigm

In order to enable and operate this dynamic and versatile distributed system, we have come up with a new software paradigm, which you may have seen parts of it in the previous chapters. We call this Clustered Visualisation Services Paradigm, since it has certain similarities to MicroServices and Service Oriented Architecture.

In this paradigm, we have modular system and app components on distributed locations of the system that communicate over API to create dynamic logical functionality that span across physical locations of the system.

Through this mechanism, users or logic can call API functions on other components of the system and apps both locally and remotely, and therefore enable many dynamic functionality like app deployment, app operation, utility operation, collaging and syncing, device integration or composite apps on the cluster of LHRD clients and the server.

CVSP is composed of 5 rules:

1. System and utilities have components on the server, cluster of display nodes and control interfaces.
2. Apps, modules, composite apps have dynamically loaded instances on the server, dynamically loaded components on the nodes or control interfaces. They have operational independency within the scopes of these components.
3. System operates an underlying communication schema that enables communication between all these physical locations and logical components within those locations. As parts of this schema, all these components can call each other locally or remotely.
4. System mechanisms and apps coordinate between components of theirs over network, so they can create distributed functionality.
5. System and app components can communicate and coordinate between each other both locally or remotely to create composite functionality.

Overall, we were able to create a distributed system structure composed of distributed components of apps and system that communicate and coordinate over API to enable the target functionality. In this system paradigm, distributed components and communication schema between them divides the physical distribution in the system into further logical boxes that operate separately. Furthermore, these physical and logical boxes can be created dynamically. For example, a new node can join as a physical element to the system, or an app is deployed to a section of nodes, where these nodes create instance scopes to that instance of app, that can be accessed over API dynamically. This creates a dynamic, modular permissive API based distributed system. In this system, apps or system utilities can create many types of distinct distributed operation through being able create many proactive or reactive communication cycles between their server, node and control components.

There are four important aspects of this paradigm: modularity, dynamicity, communication and coordination:

In terms of modularity, we create different scopes to these components on both server and clients of the system. So system has its own scopes on the server and on each client. Apps have their own scope and a hub on the server, and their

own scope on the clients. Also, instances have their own threads on the server, and their own scopes on the deployed clients. These scopes enables apps to have operational independency within those locations. So, an app component on the server or on a client have vast freedom in terms of operation within its scope. It can use many libraries inside. On the server it can use .NET and C# libraries for coordination and data processing. On the clients, it can use many online web libraries for visualisation, interaction and other supplementary uses. Therefore using these, apps can implement many types of operation within these components.

In terms of dynamicity, the distributed system is dynamic. Applications of the system can be created or destroyed dynamically on each of the distributed locations and where they can run separately within their scope. Furthermore, these physical locations can be dynamic too, they can be reloaded or join and leave at will. For example, a client can be connected to the system, join the network, load system components, receive its parameters, load app components it should be displaying if any, and start to operate, later can unload and load another app component; and finally leave the system. This enables us to operate the system flexibly and furthermore deploy any app to any part of the system at will.

In terms communication, connecting these components to each other through a communication schema enables us to create, operate and manage distributed applications and utilities in the system. For example, control component of an app can call the app instance on the server upon user interaction, which can call group of nodes the instance is deployed to change a template. Or, app logics on group of nodes could individually query data of the server component of the app as they need, while server component sends broadcasts of parameter changes. There can be many types of distributed operation that could be implemented on these logical applications and utilities that span across LHRD dynamically through these 3 types of components and communication between them.

Finally, in terms of coordination, this paradigm enables creation of composition of functionality between apps themselves or between apps and system. This includes apps or users to be able to call system API to change or dictate system state. For example, an app component on a node can call a system component on the same node or a system component on a control client can call an app component on the server. Through this, an app can load other apps, which lead to creation of their

dynamic components on server and nodes, and through API can command their server component or these node components of apps. There can be many types of coordination, syncing, collaging or composition of functionality implemented using these mechanisms.

As a result, we achieve a dynamic service oriented mechanism tailored for a LHRD that displays many data visualisations. In comparison to the MicroServices where smaller distributed services function together to create applications, we used a similar paradigm to distribute and coordinate app and system logics on the LHRD distributed system and cluster. So, apps can create functionality over distributed components. Furthermore, combination of apps and system can mesh or create functionality through communication and commanding each other over API on-demand. This is first of its kind in LHRDs and through it we achieved many novel features overcoming the challenges we stated.

In this chapter, we will explore each of these novel or critical functionality or capability this software paradigm enabled, which it did through overcoming challenges stated in the 3rd chapter; but first we will introduce how implementation of CVSP works from its own perspective in comparison the previous chapter.

6.1 Implementation of CVSP

We can divide the implementation of this paradigm into two areas: Server-side and Client-side.

Server-side: Server-side of this software paradigm is implemented through a collection of hubs connected to registry of the system. Hubs are permanent during deployment, although number of hubs are determined based on apps. Hubs determine the access level to all server-side elements. They can forbid access to certain objects or variables. Behind hubs, we have a registry. There is a core side of the registry dealing with core operation and utility operation. Also there is apps side registry, where users or logic can create instances of apps. These instances are dynamic and internal coding of them are up to the app developers. Furthermore, their internal operation can vary greatly on the configuration they have been given or user actions on control panels. So in server side, we have core system hub which acts as the API to core functionality on the server and behind it we have server-side core

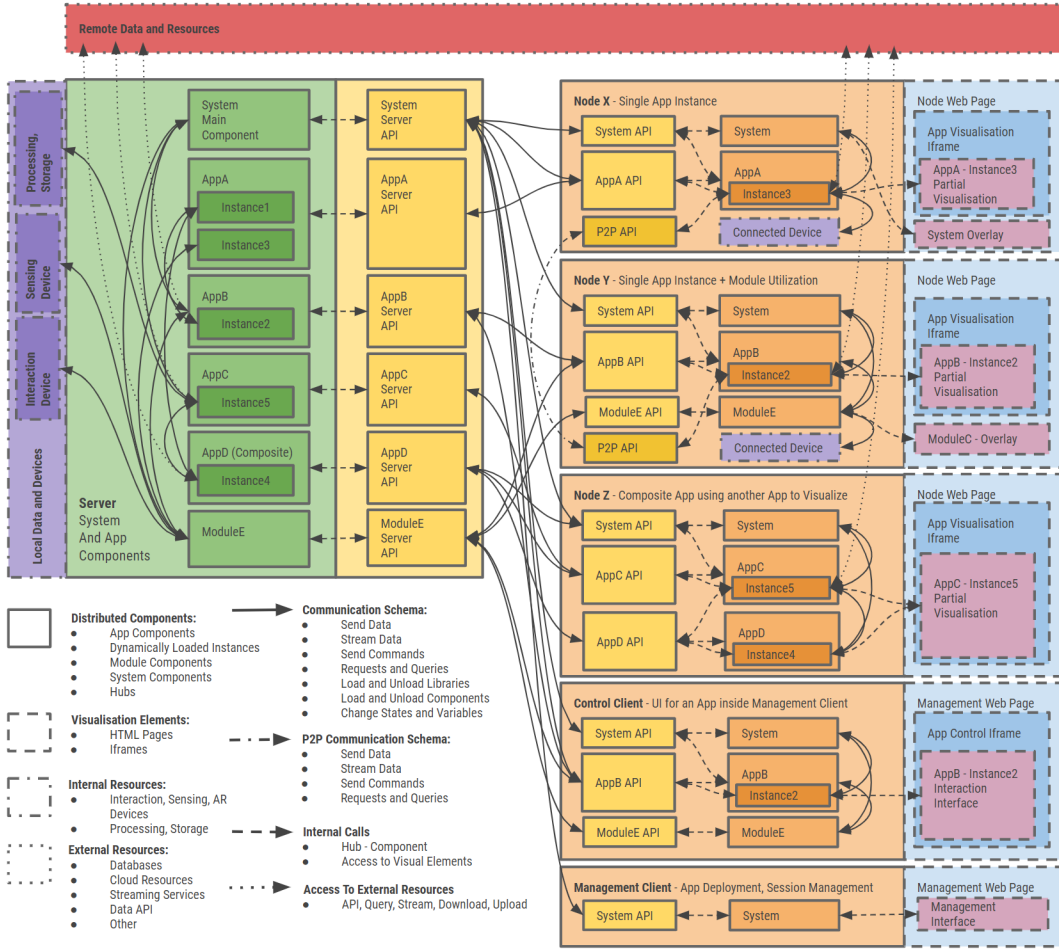


Figure 6.2: Clustered Visualisation Services Paradigm Communication Schema and Architecture consisting of System and App server and client components. In a 64 screen LHRD there will be more than 64 clients when we include multiple management and control clients and there can be tens of apps, modules in operation any second.

functions and variables. We also have app hubs for each app, which are developed by different app developers, where behind them is an app registry for that app that holds configurations and instances. Through core hub, these instances can be created. Each instance is a thread. Through app hubs, these apps and their instances and their accessible functions can be called and thus they can create their own operation. Therefore, we have core system and utility components and dynamically created app components on the server where each can be accessed through APIs and have their own scopes and operate.

Client-side: On client side, the usage of this mechanism is even more flexible due to javascript. Clients have a similar structure, but this time the hub functionality embedded to scopes themselves. We have a javascript structure to create this. The structure is similar to a tree, and starts with GDO core library itself, which implements core and utility functionality. It also creates scopes to apps and instances on its branches. It can load libraries on-demand to any scope on the structure. This structure also has access to web interface itself, thus be able to dictate what is on the screen of that client. This structure has API gateways both for the system and apps implemented through signalR. So on each client (cluster of nodes or control clients), we have javascript structure that has access to web interface, which has API gateways for core and app operation, which has both static and dynamic scopes for system, utilities, apps and instances. Overall, the components (system and apps) on that client can be created, terminated, accessed over API; and furthermore, the components on that client can access to each other and to the web interface; and thus operate. So overall, we achieve the aspects and rules described, where we have many components of apps and system on physical locations and API gateways to communicate between each other. The next sections will discuss challenges that we overcome using this paradigm.

6.2 Immersive and Scalable Display of Data Visualisations

One of the challenges GDO approach solved is immersive display and visual scalability of data visualisations on a LHRD. This challenge is both mentioned in Immersive Display and Management of Visualisations on LHRDs topic and Data Visualisation App Properties in the third chapter. It is about being able to scale a deployed visualisation to any node group using current generation's technologies and be able to deploy many heterogeneous data visualisations that are scaled in their sections in any layout.

The issue here is not just achieving scalability or displaying heterogeneous visualisations. There is considerable amount of work for distributed graphics in the literature which can achieve certain scalability in LHRD like environments or deploy multiple visualisations. Older generations even though providing scalability, they failed on scaling and arranging dynamic, interactive, web-based, general and

modular visualisations. Therefore, the problem is achieving it for versatile and web based data visualisations that are general, modular and interactive. As a result, our counterpart here is the recent generation of LHRD software. In that generation, we have identified scalability and arrangement problems that we will try to solve for data visualisations.

The most significant example is scalability and arrangement issues in SAGE2. These issues are result of their software paradigm. Since their objective is oriented around collaboration and productivity, SAGE2 software paradigm is designed without taking into consideration support for scaling and arrangement of data visualisations. First, we will take a look on scaling. In GDO, through Clustered Visualisation Services Paradigm (CVSP), we provide a solution to this challenge. In order to elaborate this, we will explain how a conventional application scales on SAGE2 and later on how a similar application scales on GDO.

In SAGE2, applications runs parallel app logics using javascript on SAGE2 display nodes. There is no coordination point like a server-side logic to an app. There is no communication between app nodes. There is limited communication between apps themselves which is unrelated here. The only communication exist is: that system software of SAGE2 sends system events to SAGE2 nodes, which results in actions on nodes. So a client-side logic of an app, has to visualise only with these system events like app deployment, pointer events and other. This prevents app level coordination inside the cluster. As a result, the only reasonable solution is pseudo-scaling, which is creating the app visualisation in full-size on all display nodes and only showing the corresponding part and cutting excess areas, which may not be problematic for productivity applications, however is problematic for scaling data visualisations.

On the other hand, in our approach, the applications run distributed with app's server logic to coordinate. So, each node component of the application could operate differently depending on its location and the data or parameters it receives from its server component, where this component could send the data as a broadcast, groupcast or unicast. So, it can send different data or set of actions to different nodes in that section using CVSP. Furthermore, the nodes in cluster themselves can send data/queries or have different communication cycles with the server-side logic of the app or even use P2P to communicate in a swarm with each other.

So to demonstrate this, we have conducted an experiment with the one of the similar applications SAGE2 and GDO has: Maps. For this test, we will use SAGE2's google maps application and for GDO we will use basic maps application. We will scale the application to 64 screens. This means the application will have a size of roughly 132 million pixels, when it is largest. In SAGE2, applications have predefined sizes, and for this test we had to manually scale this application (removing limits) to 64 screens for matching corresponding map application on GDO and its scaling capability.

So in the largest try in the experiment, which is scaling to 64 screens; in our maps application, a global canvas in smaller size is displayed on user control interface. This sends coordinates to server logic. Server logic sends these global coordinates to nodes and app logics on nodes calculate local coordinates. Each node then creates a local canvas in the size of the node display area which is 1920 by 1080 in this case and load a partial map with those coordinates. 64 nodes doing this simultaneously results in 132 million pixel sized map. In comparison, SAGE2 google maps app which tries to create a map on each node by the size of 132 million pixels while GDO creates a 2 million pixel partial visualisation on each node. In total for a visualisation that has 132 million pixels, GDO creates a 132 million pixels canvas in 64 nodes, while SAGE2 has to do 8.4 billion pixels, where almost 8.3 billion is in excess.

Lets take a look of the effects of this experiment on the performance. To reiterate, in this experiment, we have tried to load googlemaps app to SAGE2 and BasicMaps app to GDO with different scales. SAGE2 app uses google maps API and GDO app used bing maps API. Since these APIs use different compression for map tiles, Base map resources required for 1920x1080 map visualisation is around 3mb for BingMaps that GDO used and 1.5mb for GoogleMaps that SAGE2 used. We made sure that neither internet connection nor quota on map tile APIs were causing a bottleneck on the operation of the visualisation. Both SAGE2 and GDO were configured to have 64 nodes. Even though they were configured to have 64 nodes, this experiment was conducted on a single computer while only single node active and single server active for both GDO and SAGE2, and on each setup node and server communicate on localhost. As for apps, Maps are loaded with similar zoom levels and centred at New York City coordinates on both applications. We

measured total amount of downloaded resources and total time that was required app to load the map. We took 3 results for each try and average it. The results are for a single node trying to load their partial visualisation while changing the total visualisation area for overall map. For example, in 32 screens, the total map is for 32 screens while each node on GDO or SAGE2 try to load their partial visualisation which is $1/32$ of the map area, which the results try to measure load time on one of these nodes. Table 6.1 shows the results of this experiment.

Table 6.1: Performance figures of a single node on similar maps application scaling by GDO and SAGE2(There is no variation in transfer size over runs)

Resolution		GDO		SAGE2	
Screens	Pixels	Transfer	Latency	Transfer	Latency
1	1920x1080	3.4mb	$2.1 \pm 0.1s$	1.6mb	$1.6 \pm 0.2s$
2	1920x2160	3.2mb	$1.8 \pm 0.2s$	2.7mb	$1.5 \pm 0.1s$
4	3840x2160	3.3mb	$2.2 \pm 0.1s$	4.5mb	$3.2 \pm 0.7s$
8	3840x4320	2.6mb	$1.8 \pm 0.2s$	8.6mb	$5.3 \pm 0.6s$
16	7680x4320	3.3mb	$2.4 \pm 0.1s$	15.5mb	$8.7 \pm 0.7s$
32	7680x8640	2.5mb	$1.8 \pm 0.1s$	23.6mb	$21.1 \pm 0.8s$
64	15360x8640	2.8mb	$2.2 \pm 0.2s$	38.7mb	$120.6 \pm 2.5s$

In the results of this experiment, we can observe a stable data transfer and load time in GDO as the map scaled throughout the LHRD. However on SAGE2, due to the reasons we have stated, single node load time increases exponentially with total map scaling, as well as resources it uses, since it tries to load the whole map on each node. This resulted in unresponsiveness on the UI after 8 screen size map as panning actions resulted in delays. It also resulted in crashes on the browser after 16 screen size map and as we approached 64 screen size map scale, most of the tries crashed before loading the visualisation on SAGE2.

This experiment shows us app and system paradigm used in SAGE2 is unsuitable for scaling data visualisations. And with the constraints of these paradigms, writing an application that has coordination on the cluster which has difference in operation on its nodes such as having partial visualisation tasks has many difficulties in SAGE2. On the other hand, one of CVSP's primary goals is to enable this, where visualisation nodes can coordinate and visualise through displaying partial visualisations.

As for the arrangement challenge, SAGE2 offers windowing with on-screen pointer to interact which is more befitting for collaboration that is conducted for

productivity. However, deployed applications in SAGE2 can not scale freely to desired sections on LHRD, due to many of them having restrictions on scaling and thus desired layouts can not be created by SAGE2 on any selection of applications. On the other hand, GDO uses grid arrangement where applications can be deployed to any section on LHRD divided by screens, thus any dashboard styled layout can be created, while all applications scale accordingly to their corresponding section, thus creating a seamless data visualisation scene with heterogeneous applications. In comparison to SAGE2, SAGE2 creates these kind of scenes with only certain applications and with manually arranging them on-screen if they allow. On the other hand, GDO offers this for all applications guaranteed by its app paradigm and allow arrangement prior to deployment through its user interface.

Moreover, SAGE2 deploys all apps to all nodes. So, in a 64 node LHRD deployment, every node of SAGE2 creates a canvas for 64 nodes, and deploys the apps that shouldn't be on that node to off-screen areas of that canvas. This creates a serious problem in deploying scalable heterogeneous visualisations. On the other hand, GDO deploys apps only to nodes that they should be displaying on.

As a result, we see that for diverse web based data visualisations, the ability apps to be distributed and their components to freely communicate and even implement their communication leads to easily scaling these visualisations, since each app logic on each node in the cluster can have partial jobs of visualisation and can communicate with the server, receive or query data or parameters, even communicate with each other, or external data services to perform its task. If applications are designed to simply create a visualisation on each node and deployed in parallel, where they are broadcasted through system events like in SAGE2, they have to use pseudo scaling which is not performant over certain number of screens. Moreover, this kind of general scaling capability on all applications of the system leads easy arrangement of applications since each application is guaranteed to scale to sections they are deployed, thus enabling creation of data visualisation layouts of heterogeneous applications on GDO using any of the applications developed to GDO, unlike SAGE2 which cannot provide this.

6.3 Data Analytics Oriented LHRD Operating System

A platform built for Visual Data Analytics should have an underlying operating system and management mechanisms in the lines of requirements that we described in third chapter. We can see in the literature that most of the software in the literature does not offer an operating system of this kind or management mechanisms tied to it. Instead either they have a simple visualisation deployment mechanism in a program or offer a code library that developers use manually for visualisation needs or other unsuitable mechanisms for this task.

On the other hand, SAGE2 and other recent generation software offer distributed operating systems that can be controlled from management interfaces similar to ours. SAGE2 system is the most relevant to ours due to the similarities of 3rd generation. However, since SAGE2 and GDO has different objectives, we see that their management mechanism and underlying operating system are incompatible to our design and objectives in terms of supporting general and distributed data visualisations and many other functionality we listed in the third chapter. There is two aspects to this topic, where the first one is the underlying operating system and the second one is the management functionality provided in UIs built above this.

Operating System: In GDO, CVSP principles lies beneath GDO operating system. CVSP structures the system, defines the rules how it should operate. In this sense, GDO operating system builds above implementation of CVSP; while CVSP provides modular design of distributed system and communication schema, GDO implements system functionality above it such as app deployment, app communication, app management, session management and unified screen-space. Furthermore, through CVSP, GDO deploys 3 different types of components to server, cluster of nodes and control clients dynamically. These components can use any libraries within their scope, can coordinate and even implement that coordination within their own communication schema which CVSP and underlying operating system enables, structures and manages.

In SAGE2, SAGE2 also use a similar distributed operating system to GDO in terms of implemented functionality, however it does not use CVSP principles. In SAGE2, SAGE2 distributed system spans from server to its clients and operate through API based messaging. It deploys apps similarly to GDO, where user ac-

tions from management UI triggers events on the server, which is broadcasted to nodes, which deploys apps. Afterwards, app logics on clients operate using reactive communication coming from the server events.

There are two problems in SAGE2 in this area. First one is the app deployment mechanism. In SAGE2, all apps are deployed to all nodes as mentioned before. So if there are 10 apps on screen-space, all of these apps are deployed to all nodes. So, even there is a node where no app should be showing; on that node, all 10 of the apps are loaded and displayed in excess background. This is unmanageable for general, modular, data-driven, scalable apps while being acceptable for productivity and demo applications.

Moreover, there is a problem regarding what OS of SAGE2 provides to operation of the apps. Deployed apps in SAGE2 start with default state. User interaction on the system, such as clicking on the app through a virtual pointer, or using app menus registered to system, creates common system events on the server which are broadcasted to app logics on nodes; thus changing the state of app logic on each node. This is how apps operate in SAGE2 due to their underlying operating system. So, no other types of intra-app coordination or operation is possible. Apps have to operate in parallel without a coordination point, and based on broadcasted common system events. Only apps exception to this are system hardcoded apps in SAGE2.

As a result, SAGE2 has 9 base functionality missing, in comparison to GDO. These are:

1. Coordination Point for Apps: Apps should have coordination point, where data or parameters can be hosted and distributed.
2. Selective App Deployment: Apps should be deployed to nodes that they should appear on.
3. Persistent States: Having a coordination point enables a persistent state for the app during operation, which can be updated through logic or interaction. Thus, reloading clients can show up-to-date state instead of default state. Furthermore these states can be saved or restored afterwards.
4. App based Communication: Apps should implement their own communication to have intra-app coordination.

5. Partial Querying of Data/Parameters/State from the Server: Node components of apps should be able to request partial or whole states or data from server components of apps on-demand.
6. Proactive Communication: Nodes and other clients or server should be able to have proactive communication cycles.
7. Selective Communication: App components on the distributed system should have selective communication between each other, where they can send different data or parameters to different nodes.
8. App-System Communication: Apps should be able to use system API.
9. Inter-App Communication: Apps should be able to use other apps' API (SAGE2 implements Inter-App variable sharing but no direct API communication).

These base functionalities enable many advanced properties of GDO such as general, scalable, data-driven, collaged or synced data visualisations. For example, data-driven apps often use proactive communication to query data hosted on the server component of the app during app operation. Or, a general app requires a coordination point to host data and live configuration. Or collaged apps, requires both app based and app-system communication. Unfortunately, these are missing from SAGE2, due to constraints on SAGE2 operating system.

Overall, these base functionality are critical for data oriented apps, and thus prevents many features like general, scalable, data-driven, collaged or synced data visualisations to be implemented on SAGE2. On the other hand, CVSP based operating system enables creation, deployment, operation and management of many advanced distributed apps, due to enabling all these base functionalities and properties through CVSP.

Management Mechanisms: In terms of management mechanisms, SAGE2 and GDO offer capable web based management above these operating systems of theirs. GDO management is visual data analytics oriented while SAGE2 management is prioritizes desktop like use. SAGE2 offers an interface which has a canvas that enables users to deploy apps and move them around or arrange them. GDO have similar mechanism. GDO's difference is: it enable users to select configurations for apps and apps can be only deployed in grid layout, which we prefer for data

visualisations, and also which uses screen-space more efficiently and is presentation ready.

In comparison, SAGE2 focuses on collaboration and productivity, which requires more on-screen rearrangement of productivity and basic apps. Furthermore, SAGE2 interaction is very limited on apps and only on-screen interaction is possible. In our system, apps can implement their own interaction interface that users can use to configure/control or steer the app. Moreover, in GDO management mechanism, GDO offers saving and restoring states and execution of scenarios. This enables sessions and session management mentioned in 3rd chapter to be conducted. This utility is again implemented through CVSP, where over API current state of apps and screen-space is saved and then any saved state is restored using the same API functions that is used for app deployment, all through either from control interfaces or from logic.

Overall, both systems offer web based management mechanisms and underlying operating system in comparison to 1st, 2nd generation LHRD software, which does not have and does not provide mechanisms for deployment and distributed operation of apps and utilities. However, our system is oriented in data analytics with heterogeneous app deployment and distributed operation in any layout and saving restoring states and scripting them. Through extending API based distributed operation to apps and utilities and enabling API based distributed coordination between system, apps and utilities, we provide complex and versatile use-cases the diversity in data science apps deems. And thus, we have created a more suitable operating system for data analytics on LHRDs, in comparison to SAGE2, which its operating system constrains many critical functionality required.

In terms of management interfaces, our system is more suitable for visual data analytics, since it is interested in more structured deployment of applications using grid layout, and also instanced and differentiated deployment of general data visualisations for specific use-cases, moreover enabling session management using saving and restoring system states and scripting. On the other hand, SAGE2 focuses on productivity related interaction and desktop like use, its operating system and management mechanisms focus and provide functionality related to these topics.

6.4 Modular and General Data Visualisations

Another major challenge on this objective is creating an app paradigm where apps are modular, general and standardized. Modularity provides any selection of apps to be included to a system deployment as packages. This is a requirement to create versatility in visual data analytics since many general or custom apps could be built for these environments. On the other hand, generality of applications is required due to being able to provide generic data visualisation apps that can be re-used with different data and configurations. Finally, standardization enable us to operate the system robustly and allow us to use utilities over 3rd party applications built for the system. Overall, we need all three of these properties to provide modularly deployable and manageable data visualisation applications that can accept various data and configuration on each deployment.

Modularity: In SAGE2, applications are also implemented modularly, similar to GDO. They can be included and excluded from deployments easily. In SAGE2, apps are js files that should contain certain functions the system calls. So, any js file having those functions can be implemented as an app and included in deployment.

Moreover, SAGE2 apps are also standardized. They respond to certain system events broadcasted from the SAGE2 server and render according to these events. However, these apps are not general alongside apps of many other software in the literature, which is a critical functionality for modern visual data analytics for given objective. Therefore, these applications do not have configurations which enables them to be deployed with different data, templates, styles or parameters. As a result, they cannot be used as general data visualisations, which can be utilized and customized by users or other apps to visualise specific use-cases diverse research areas may have.

In GDO, we have implemented applications to be modular, general and be standardized at the same time. We have implemented modularity through dividing our app into components and using these components as entry points of programming for app developers. So, an app has server, node and control components, which are created and deployed dynamically to LHRD system. Each of these components should implement certain classes or scripts or web pages. However, these components can make use of other external libraries freely after implementing those

interfaces on the given scopes. This is how we applied developer freedom alongside standardization. Even though, our app paradigm is distributed and uses multiple programming languages, and even implements its own API per app, we were able to package it to work as a plug-in to the system using this structure and command it over API with common functions they have to implement. We used MEF, Autofac, signalR and our own middleware to do so.

Generality: In terms of generality, we have implemented a configuration mechanism. App instances can be deployed with configurations. Each component of app, both on server, cluster of nodes or control clients are given these configurations over API. Therefore, these components can initialize and operate using parameters or data stated in the configurations. Furthermore, since apps work over API, their components can implement functions to change datasets, templates, styles and other parameters during operation too.

For example, a graph application is a general distributed graph visualisation, which accepts graph data and distributes the data to nodes and each node component visualises the data. These and visualisation options can be stored as configurations which can be used in each deployment diversely. So different instances of this app can be loaded with different data, templates and styles. Furthermore, during operation, users or other apps can command the API of this app to load other datasets, change templates, show labels, select graph nodes to focus and many other options that the app developer may implement. Thus, an app can easily be further configured to visualise given specific data in desired template or style in order to satisfy diverse use-cases of Data Science it may be used for.

As a result, in comparison to SAGE2 apps, which cannot be configured upon deployment and further during operation, GDO provides deep configuration and interaction capacity to apps using CVSP. At the same time, again using APIs, the same application and its components can respond to utilities which system components may call locally or remotely thus providing standardization. Finally, since apps can be packaged implementing our code interfaces, we can deploy them to server, instantiate their instances and load their client components and libraries dynamically, or terminate all their components dynamically, through using CVSP paradigm and OS based on it, thus providing required modularity, generality and standardization.

6.5 Interactive and Dynamic Data Visualisations

Interactivity and Dynamicity are important aspects in data visualisations. Users in determined use scenarios need to interact, configure, steer or control the visualisation during a session. This is both extension of generality, where users may configure the data visualisation through UI by selecting data, or uploading data, selecting templates or visualisation style. Also, this interaction extends to steering or controlling the visualisations actively or working with it. As for dynamicity, dynamic events such as incoming data or events from systems, data sources, logic or devices may be streamed and have to be reflected on the visualisation in real time. This enables use of live data sources in visualisations, device integrations or live processing.

Interaction: In SAGE2, interaction is limited to common interaction and per app on-screen menus. Common interaction is where users have common interaction on apps through system defined pointer events. For example, when a virtual pointer clicks on the app, that event is sent to nodes. SAGE2 also has on-screen interaction menus, which apps may register to. In this cases, apps can register menus that users may interact (The second functionality is implemented in 2016 after our implementation in 2015[39][127]). For both types of interaction, system passes these events to app logics on nodes in parallel.

The problem is SAGE2 apps do not implement web pages but have js files creating divs for visualisation. Translating pointer events to div areas for interaction creates significant developing problems. For example, if you have a chart application where page is created by a js, and system telling you a user is clicked on a pixel, you have to create a js function to translate this to corresponding event. This creates a large if else code structure for complex interactions and is unnatural way for development. In our case, interaction is done by a control web page developed by app developer powered by a js, that translates user actions on html elements to js functions, which translates it to API commands to be called on the server component.

Another problem is on-screen interaction SAGE2 uses vs control interface per app paradigm GDO uses. On-screen interaction is very limited, it takes on screen-space and spectators are subject to interaction itself. On the other hand, in our ap-

proach, users or admins use control interfaces per app and interact/steer/configure app without taking any screen-space. Also, separate interaction interfaces enables admins to interact without being monitored by spectators. Moreover, control interfaces enable apps to have complex web pages for interaction which may have multiple tabs or menus and may contain large amount of interaction elements.

As a result, in GDO, complex interaction mechanisms can be created by developers for each app. This further support generality of data visualisations, since these interfaces can be used to configure the general visualisation by selecting or uploading data, selecting templates or styles and many more. For example, in a complex map application, users can create layers, select layers to display, create styles, use these styles, create animations and play these and conduct many other advanced tasks through UI on tablets and PCs, all without being monitored by spectators.

We have enabled this through using CVSP, an app is divided to control, server and node components. App developers can implement interaction UI for users which communicates with server component and therefore users can configure, steer or control the visualisation through their events translated into API calls over this mechanism. Each app can have general or specific control interfaces that enables complex interactions.

Dynamicity: As for dynamicity, when we look at SAGE2, we see limited coordination and communication within distributed app. So, if a logic wants to flow data or parameters or events to apps, apps have to implement these externally since they lack a server component. This external manual coordination point has to identify each app logic on each node and stream accordingly which is unlikely. So in SAGE2, the natural way is to stream data to all app logics in cluster through broadcasting or they can query individually upon system events using an external mechanism. This creates problems implementing diverse dynamicity required by many data visualisations either general or custom. In our paradigm, apps can stream data to nodes either through broadcasting or individually over API through their server component. They can start, stop animations or other events based on logic in the server component over API. They can stream data individually from external sources given their parameters. They can react to user events in a more coordinated way.

As a result, more dynamicity and interaction options are possible if apps have 3 components on server, node and control clients, and their interaction and dynamicity

can be implemented over API which CVSP provides in GDO, and SAGE2 and other systems don't.

6.6 Collaged and Synced Data Visualisations

One of the major challenges or feature for performing modern visual data analytics on LHRD is to coordinate, collage and sync data visualisations for multi-view visualisations mentioned in chapter two and three. This functionality is not been provided by any platform that has been implemented prior to GDO and therefore is completely novel. We have used CVSP extensively to enable this. Through this, we can coordinate app instances of the same app, instances of different apps, use parent apps (composite) to deploy and visualise through other children apps and coordinate and use scripting to execute sessions where many different layouts of apps are deployed and commanded over a session. Lets look at how each of these are achieved.

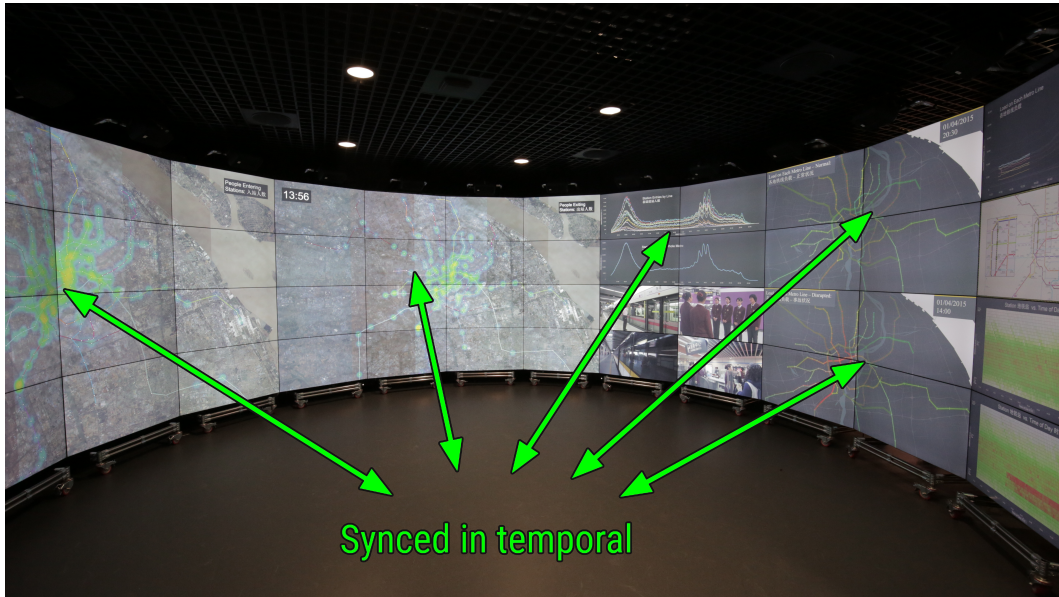


Figure 6.3: Sample Collaged Visualisations that are Synced on Temporal

Coordination between Instances: Instances of the same app can coordinate between each other in GDO. This is performed through use of scopes and hubs. These instances use the same hub and they have a scope for overall app. Therefore,

hub commands both going into server instances and back to clients can work in app scope instead of instance scope. As a result, they have an effect on all instances.

For example, there are two map animation app instances, that when activated runs an animation of data on spatial plane throughout time. The play action on control interfaces of any instance of the app can be ordered to work on app scope. Thus, hub will order all instances of the app to play instead of a single instance. It afterwards distributes that play command to all nodes of all instances of this app in a synced way over API. Hence temporal sync between all instances of the app can be achieved among all nodes and control interfaces. This sync can be further improved by a system heartbeat mechanism we implemented, which reduces the synchronization to less than 30ms, which is required by videos and other apps that desynchronization is noticeable. Moreover, coordination between instances can also be used other than synchronization in time, as it can be used for other parameters such as spatial coordinates, or it can be used to change datasets, templates, styles or parameters, or logical coordination.

So through having a single app scope and hub and instance scopes on both server and client components of an app, we can have collective actions over instances of the same app using CVSP. Allowing us to sync and command them collectively or coordinate them. As a result, we can have many individualized visualisations of same general app on screen-space and we can sync, coordinate or operate them collectively through utilizing CVSP.

Coordination between Apps: Instances of different apps can communicate between each other through CVSP too, and sync and coordinate. Although, this functionality has to be implemented manually on one of the apps or both of the apps, so apps can recognize each other and be compatible to this kind of operation. These apps can communicate between each other and sync on a data parameter in few ways. First one is a control interface of one app can target hubs of multiple apps and send messages to all. Second way is instances of different apps can communicate between each other on the server itself and relay the coordination back to their client components of those apps. This can be used for syncs on data parameters or other supplementary logic and coordination between apps. This coordination can be reflected to control interfaces of the apps.

For example, a chart and graph app may implement to coordinate to use the

same dataset. Actions on one of the control interfaces translate to other. Any selected item on chart app is highlighted on graph app. Furthermore, let's say the data is time series and when played both apps animate. These functions can be implemented on APIs of server instances of apps. So when a data category is selected on control interface of chart app, either control component directly or over its server component calls graph app's API to highlight the nodes in that category. When play button is clicked in control panels of any of the apps, either that control component tells both apps' server component to stream the data or tells its own server component which tells the other apps' server component to stream the data. Thus, time-series starts to play and both server components stream data to their node components.

All these and many more actions can be implemented between apps, since apps themselves and their components work over API, and compatibility and collective operation can be programmed between these components using API, which is made possible using CVSP.

Coordination using Composite Apps: If we take previous item one step ahead, we will have parent apps that coordinate children apps. We call these composite apps in our system. Composite apps can be used to coordinate child apps of theirs through using CVSP. In this case, a composite app runs on all nodes child app instances are running. A user can use composite app's user control interface to dispatch a command to its own hub. This hub can relay the command to all of its nodes, where also child app instances are running. On the nodes, composite app logic can use CVSP to access child app logic or variables. For example, there are few animating child apps coordinated through a composite app. A pause action can be relayed through composite app pipelines to all nodes these child app instances are running. Composite app knowing API functionality of each child app can use API to order each instance to pause. In the same manner, it can do this to server instances of those child apps. Composite app's instance on the server can access to child app instances and communicate with them for coordination.

As a result, composite apps can utilize multiple apps to visualise and coordinate them through CVSP. One example is advanced presentation which can deploy other apps as partial slides in overall scene. Over API, it tells these apps to load certain configurations and even control them during, such as starting or stopping

a video. Overall, this mechanism can be used for synchronization on parameters, activating states, sending data, sending parameters to child apps and their components and many other coordination scenarios; also deploying and unloading child apps directly from parent apps components.

Coordination using Scripting: Finally, the coordination between apps or overall screen-space visualisations can be scripted. This script can be implemented manually or through scenarios mechanism mentioned in the previous chapter. The script either should be designed specifically for a data scene with pre-determined app layout or should be complex enough to determine the apps on screen-space and through logic coordinate them. Of course, scripting is a more temporary solution or primal solution, but it has few use cases, such as make do session management.

As a result, a script using API of the system can load apps, change states of their components using API, provide them configuration or data, animate them, than unload and load a new scene of apps using CVSP, thus enabling some of the session execution requirements described in third chapter.

6.7 Integration of Interaction and Sensing Devices

Another functionality CVSP provides is integration of interaction and sensing devices modularly. These could be controlling devices such as joysticks, or sensing devices such as motion sensors, eye-tracking, or other devices such as AR glasses. In other software in literature, integrating these devices require changing the system code itself, including SAGE2. GDO offers two mechanisms to integrate these modularly.

Module Level Integration: First mechanism is module mechanism. In this mechanism, an external developer may implement a module that has a single instance and always active for that device. The device may be connected to server itself or any of the clients of the system or send data to server over the network (client connection require access to system resources over browser or should connect to device over API). This module than provide the functionality or data of the device to the system over API. Other apps can access to this through module's server API or module's client API. So a component on a control interface can call module's control interface component locally to access to device data or controls; or

an app instance on the server can call module instance to access to device's data or controls locally. Furthermore, an app component on a client can call module's server component over API. So, module can offer these data or controls locally and remotely both on server and clients. Moreover, since module is active on server and all clients, this device or multiple devices can be connected to any of these locations and over the network their data or control can be provided to any other component of the system or apps through CVSP.

App Level Integration: Second mechanism is, apps can integrate these devices directly. So an app, can connect to a device on the server or on one of the clients it is deployed to (client connection require access to system resources over browser or should connect to device over API). Since instances are dynamic, providing this functionality to other apps would be problematic in comparison to module approach, but this has been used too in our framework.

So overall, we can connect devices and implement functionality as part of an app or individually as a module, where other apps can utilize the data or controls these devices provide over API using CVSP all modularly without changing system code, where implementing this functionality requires changing system code on other platforms in the literature.

6.8 Data Distribution and Load Balancing in Data Intensive Apps

Another major challenge for visual data analytics on LHRDs is making many types of data and resources available to the system, allowing many processing and distribution scenarios for them and providing methods to conduct load balancing on cluster of nodes. Through using CVSP, we enable many data access, processing and distribution scenarios and options. This section discusses various ways apps can utilize these scenarios and topics related to these.

Processing and Distribution: Apps can choose to process on the server and then distribute the processed data to the nodes or apps can choose to distribute the raw data to nodes and process on the node or something in between. This 3 options are most common practices and usually selected depending on the app.

On global processing, app instance processes the data and send the nodes their local bit. Each node receives a different bit and displays based on it.

On local processing, app sends the global data; and nodes, knowing their location on the node matrix of the instance and dimensions of the matrix, processes the data and extracts the bits they should be displaying. There can be other practices in between and around these options.

In comparison to SAGE2, where none of these scenarios is enabled since apps operate in parallel without specialized coordination. GDO provides all of these scenarios through CVSP for app developers to utilize. So, in GDO, where apps are composed of 3 components and have custom API between them, developers may implement all these processing and distribution mechanisms depending on their app.

Data Availability: Data Availability is the decision where to enable data on the distributed system of LHRD in this context. So, how much of data is made available and used on the pipeline should be left to app developers. More server-side means that the instance on the server itself has access to data and processes the data and sends the data to nodes with parameters.

The client side of the gradient is that the data is available to clients themselves, and the server either do local or global processing for parameters only and clients with those parameters pull the data directly to clients either from external resources or the server.

Moreover, various apps may use external data or content, like map tiles, or other data. Users may also want to upload data to the server and made this data available. Or app developers may choose to download data directly to clients during initialization, where nodes or server component may download data directly from external resources. Apps may choose to use both, one of them or a composite variants of these scenarios too.

In SAGE2, the only option is to let parallel app logics on nodes to download/query data from external sources. On the other hand, CVSP allows many data importing, downloading and distribution mechanisms through having node, control and server components and implemented communication between them and also allowing all these components to communicate with external resources directly.

Data Needs based on Dynamicity: Data needs based on dynamicity in this context is: change in visual states of the app causing more data that may or may not be present in that display node to be required for the visualisation. So for example, in a spatial visualisation, or temporal visualisation, as visualisation changes on that dimension, more data may be required on the server or nodes due to this dynamicity, to actualize this change. This is another important topic for data intensive apps on LHRDs. Depending on data size and dynamicity of visualisation elements, clients can pull or be provided all of the datasets or just the required bit or something in between. The factors affecting this call are: the size of the dataset, the dynamicity of the app if it affects required data at any given moment or user interaction which changes dynamicity of the app or many other factors.

App may show high dynamicity most of the time or low dynamicity most of the time on data needs. Or they show low dynamicity in most of the time, and exhibit high dynamicity on certain use scenarios suddenly. These can be affected by the data or configuration used, app design or purpose, user interaction and many other factors.

So, if there is high dynamicity on the data needs, clients operates on high volume of data for even small amount of visual elements, than appropriate clustering and pulling mechanisms are needed in most of the time, or they may just distribute the whole data prior to visualisation to all clients of the app. If data is prepared on the server in this scenario, than the server app logic should apply threading to manage with processing overhead. If there is no high dynamicity, or the data size is small, than app may allow clients to query data as they need, or directly distribute data as they update. There are various solutions or decisions that affect app design in this topic.

Most significantly, the software paradigm should allow many different scenarios on data processing, distribution, availability to be implemented by apps, so they can overcome problems that may rise due to this issue by themselves. GDO, through CVSP, is again only software to provide this to apps. Apps in GDO, may choose to implement mechanisms that can statically or dynamically decide on data distribution to nodes, due to enabling coordination. So, apps in GDO, can choose to provide most of the dataset beforehand to nodes for nodes to locally pull data, or distribute over time, or make nodes to query the data as they need. Apps may also implement

certain buffering or overpulling mechanisms for temporal or spatial data to prevent lags due to high dynamicity. For example, clients may pull data beyond canvas or buffer if data is temporal.

As a result, apps may be implemented in a way that client or server components of apps can choose to acquire data based on dynamicity of app usage and its effects on data need or other factors. This can be decided by individual logics on the components or in a coordinated way of overall logic of the app, and all of this is enabled by having components of app to communicate and implement dynamic functionality through CVSP.

P2P Utilization: Clients may also want to distribute and exchange data between themselves. Each node is given a P2P client to access other nodes and put in a group when an app is created (if app activates P2P mode). Through this, each client of the instance has direct connection with other clients of that instance. They can use various P2P mechanisms to share and exchange data between the other clients of that instance, thus further expanding CVSP usage for data sharing and coordination.

Another case of P2P use is dynamic view. In a visualisation, nodes may decide to buffer visualisation data of excess areas, so when view changes suddenly due to interaction or other reasons such as in 3D or maps or graphs, where there is a spatial data visualised, the visualisation continues to be responsive.

So in this scenario, each node may download and or process data points in excess areas over the partial view of the node and keep this ready in case of sudden panning or zooming. For example, this data can be transferred to neighbouring nodes in the direction of panning instead of those nodes reacquiring the data. So if the view is panned to right, then each node transfers their data to left neighbouring node over P2P.

Load Balancing: Another data distribution related capability CVSP offers is load balancing. In GDO, two app components in separate nodes and a server component of an app can have three way communication over API. Furthermore, server component can send individualized data and parameters to different nodes of that section in groups or individually. Moreover, individual node components of apps can query server separately. All this combined, enables nodes to have different workloads, take on others' workloads or transfer workloads.

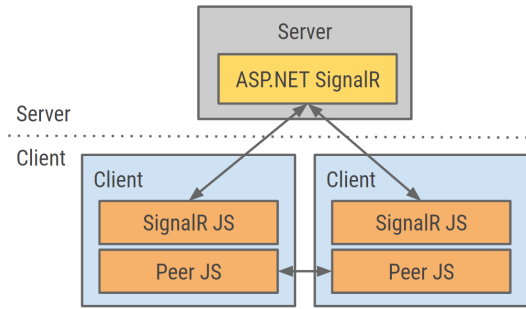


Figure 6.4: Three-way communication in GDO between Server and Clients

For example, if local processing is applied, node components of apps can be given balanced workloads (consisting of other nodes tasks) then later transfer results either back to server or directly to each other where the data is needed. Of course, server either in single location or node components in distributed manner should hold a live registry of workload distribution and thus apply balancing, which can be easily im-

plemented through using our API communication CVSP offers, where workload related messages can be communicated in that three way communication. This is practical in several use-cases. We have found that, it is mostly used on heavy local processing scenarios, where nodes can benefit this mechanism to distribute workloads.

In contrast to SAGE2, where inter app component of nodes communication is not present, neither individualized or group server to app components of nodes communication, achieving this type of load balancing is not possible without help of external server or applications. On the other hand GDO using CVSP apps can easily be developed to conduct load balancing.

6.9 Reliability in Visual Data Analytics on LHRDs

Another major challenge is providing reliability in such a dynamic, diverse distributed system where apps are distributed, extensible and developed by third party, while global utilities are enabled over them. Reliability in this context is to be able to operate with any selection of externally developed apps, where these apps can be deployable to any screen area; global utilities working over them; GDO working with flexible setups as clients can join and leave; but still errors and problems in a physical location or logical mechanism is isolated and stay within that component.

For example, a crash of a client should not crash the system, or an app crash should not affect other apps on the server or nodes or system's core functionality such as app deployment, unloading, on-screen logging mechanisms. As a result,

problems stay isolated and global mechanisms in most locations continue to operate regardless, and thus be able to recover problematic components or unload them.

In order to ensure this, we have made use of modularity provided by CVSP. CVSP makes each component in the system to operate within given boundaries and communicate through a structured and separated API. As a result, app and system components can operate without conflicting and affecting each other. Furthermore, the scalability provided by CVSP in terms of number of clients and apps enable GDO to be deployed flexibly, while providing reliability.

Implementation of CVSP takes certain measures to ensure this reliability. First of these is scoping, where each component is given a distinct area to operate. In the server, this is ensured by instancing mechanism, where each instance of an app is created as a thread and system is given control over these threads. So, in event of errors, system can terminate these threads. Furthermore, when an app is terminated, the thread is terminated too. This ensures that errors of instances of apps cannot spread and when instances are terminated there are no leftovers stay behind on the server.

A similar mechanism exists on clients. Each app and each instance of those apps are given a scope in javascript. Their libraries are loaded dynamically. These scopes can be cleansed at will and done so after unloading. On the visual side, this is done through using an iframe and separate DOM. All visual elements and additional libraries are loaded within the DOM of this iframe. Even if the iframe becomes unresponsive, the main client frame continues to respond, so does the main system component, thus the client responds overall and is able to recover. When an instance is unloaded, its given scope on main DOM is cleared and iframe destroyed with all visual elements and additional libraries.

Second measure is concurrency management on communication. These components communicate with each other through websockets. This may raise various concurrency issues such as sockets being blocked on long duration functions or race conditions creating errors or erroneous data on registries. On the server, we resolve this with separating hubs and introducing locks on each component that hubs are connected to. So system and each app operate on different hubs and thus cannot conflict in this area. So, when a hub function is called, it calls the component behind it, and if that call to component has to be atomic than component prevents

all external (incoming from the hub) or internal (incoming from other components) calls as it locks itself where these calls queue on that API.

As a result, while an atomic operation is taking place, no other element can access to that component. This prevents multiple calls on APIs to conflict on same component. Furthermore, all system hubs and most app hubs are designed in a way to act as a relay. So, the functions inside the hubs are designed to relay the action and complete instantaneously, so they do not block the hub itself. Therefore, we don't have processing code within hub functions, and instead perform processing within the components itself.

On the client side, each client uses same signalR client-side code, so their API is hosted on same location. However, we overcome this through different components using different parts of the API. From our experience, we do not need to separate API in clients since usually clients are not a congestion point as much as the server. Most of the time, only one or two app instances operate within a client since different parts of LHRD run different instances of apps. Still, the signalR client-side relay related API calls to corresponding components. As a result, even on instance scope (app component) have problems, server can communicate with that client and have command over system component of the client as it continues to operate and thus restart erroneous app components within the client.

If the core code becomes unresponsive on the clients, which is very unlikely, KPMG DO have an external restart mechanism that simply reloads the web page in that web clients and since we allow flexible joins to the system cluster, that node recovers and re-joins to the system.

So overall, due to separating system and app communication into different areas on APIs and using sockets to relay messages to corresponding components on the clients, we ensure reliable concurrent operation through the system as apps and system does not conflict and hubs are not blocked.

As a result, GDO can work with many selection of externally developed apps and did so, have sessions where these apps are deployed in different layouts, controlled and redeployed and so on. Logical system and all system components on all locations continue to operate without being affected by app operation and thus preventing app related crashes to crash the overall system.

In summary, reliability is achieved through applying modularity described in CVSP, where we have divided the system to physical locations and further separated it to logical app and system components on these locations with clear and separate scopes and APIs and communication cycles.

6.10 Interoperability, Flexible Setup, Remote Deployment and Casting

There were certain topics raised in challenges regarding interoperability, flexible setup, remote operation and casting, which we find related to each other due to their relation to using flexibility provided by web-technologies. In summary, we want to enable integration of web visualisations easily or be able to integrate web based LHRD software similar to SAGE2 so research centers can both benefit from existing visualisations online and have dual operation with SAGE2 like systems. In terms of flexible setup, the objective is to be able to work with different setups and display technologies in this context. So this system shouldn't be hardcoded for KPMG Data Observatory and should be able to work with different setups, systems and resolutions today and in the future. In terms of remote operation, clients of this system should be able to be loaded on remote locations such as server, node clients and control clients can be on different sites or node clients or control clients within themselves can be on different sites. For example, in 64 screen deployment, 32 clients could be on one site and the other 32 on another. As we mentioned, most of these requirements are related to use of web technologies in LHRD as part of 3rd generation. However, we improved them through using CVSP, and enabled them for general, scalable, data-driven, interactive applications.

Interoperability: So for the first challenge, being able to utilize existing web based visualisations, both SAGE2 and GDO can load web visualisations to LHRD using pseudo scaling. SAGE2 uses webview app (introduced more recently than GDO [39][127]) and we use staticHTML app. Both apps pseudo-scale web visualisations, while displaying the visualisation in iframes on display nodes.

As for integrating other LHRD software, SAGE2 or other systems in literature do not offer any capability to perform so. On the other hand, GDO using CVSP can do so and did for SAGE2 and our internal tranSMART app (see following chapter). Furthermore, it can do so for any LHRD like visualisation system or visualisation

steering system that operate through web based control and node clients. It does it by deploying it as an app and redirecting the nodes and control clients on GDO to their corresponding clients of the integrated system in iframes. So, GDO control clients show control panel of integrated app in iframe and GDO node clients show nodes of integrated app in iframes. Of course in this operation style the integrated app uses its own communication, distribution and processing mechanisms and rely on GDO for encapsulation only. But regardless, these systems can be deployed as an app and unloaded as apps in our app deployment mechanism to any selection of nodes safely, while other apps can operate within LHRD, where this capability only exists for GDO.

We also conducted tests on performance of this kind of integrated deployment as seen in Table 6.2. Since GDO stays dormant during SAGE2 deployment, its only

Table 6.2: Memory Usage on Clients under integration of SAGE2 to GDO

	Node	Control
Only GDO	8.6MB	12 MB
Only SAGE2	6.8MB	8.6 MB
SAGE2 inside GDO	15.7MB	20.8 MB

effect is increase in memory. This overhead is around 7-8MB for nodes and control clients which is exactly the size of base SAGE2 client memory footprint. Other than memory usage, since these are clients, there was no observable increase in CPU usage or Network usage, as GDO does not affect these in a dormant state for the group of nodes that has integrated app deployed. Also, servers of both platforms operate distinctly under this type of deployment, therefore their performance is not affected. As a result, we can conclude that web-based visualisation or LHRD software can be integrated to GDO without significant overhead.

Flexible Setup: As for flexible setups, SAGE2 and GDO act similarly and can be deployed to many types of setups that have server – control clients – node clients paradigm. We can deploy GDO in a single computer with one or more nodes or a large LHRD. In GDO and also in SAGE2, all nodes do not have to be present or active for the deployment to work. They can join later or not join at all. So, in a 64 screen deployment, a developer can only open 8 nodes on single or multiple monitors and work with these. This provides total flexibility with setup, and also ease app development as developers can run the system on single PC and use browsers to open

node and control clients for testing. Furthermore, we can easily change the resolution on clients of the GDO and also GDO clients do not require prior installation since all the system code, visuals, content and parameters are transferred over the network using a web server and signalR web socket communication.

Remote Operation: This type of flexibility also enables remote operation scenarios. Similar to SAGE2 and SAGE which is also used for cross-side collaboration, GDO clients can be deployed remotely and in duplicates when necessary. So in a 32 screen deployment, where there are two sites with 32 screens, both sites can display same set of nodes in mirror style. Or the system can be deployed with 64 nodes and each site displays their corresponding 32 screens. Same goes for control interfaces as control interfaces can be loaded to devices in all sites simultaneously.

This functionality works in GDO through our CVSP communication using group and broadcast functionality and also unicast update request. So server can broadcast to all control clients, or all nodes (even duplicate) or a selection of them through groupcast that display the app instance. In addition, newly joined nodes or control clients can request current state of the system through unicast from server and show up to date user interfaces through this. This communication freedom in CVSP allows many deployment and setup options regarding remote operation as we can support almost all possible remote operation scenarios. However, due to our purpose, we have not created utilities to enhance this kind of operation, as its not our primary focus.

Remote operation works similar in SAGE2. SAGE2 uses API based communication for system events and even though no app API directly, the server of SAGE2 can communicate with clients on any location. However, it should be noted that SAGE2 apps do not have server components and thus do not operate and offer data visualisation functionality similar to GDO. They don't support data distribution scenarios as each distributed app component on nodes individually acquire data. So in a scenario, where a data distribution exists, SAGE2 do not support duplicate or remote operation or may have problems with it, since it does not support that scenario in the first place. Through our CVSP, server component of app can send data or parameter to multiple copies of nodes or remote nodes and clients, and we ensure if the app is implemented within guidelines both functionalities is guaranteed to work even for apps with different data distribution scenarios.

Table 6.3: Remote vs Local Latency of GDO Clients with Base Load Times on Node and Control Clients and Load Time of a Graph App

	Local	Remote
Control Client Base Load Time	2200ms	5300ms
Node Client Base Load Time	1800ms	5000ms
Visualisation Load Time on Node	170ms	1400ms

The only problem with remote operation is latency for these kind of platforms. We have conducted a test for remote vs local operation for GDO. For this test we have measured base control and node UI load time, also for visualisation load time for a graph app with 234 nodes and 2100 edges which for us represents an average amount of visualisation elements on a single node during normal operation. The local location in the test was Imperial College in London and Remote location was in Ankara, Turkey; where remote location has 50 mbits/s internet connection using VDSL. The test shows that there is a certain amount of delay on client and visualisation load time. So, the result is, in order to use this type of system remotely, the network connection between both sites should be high end to prevent this type of latency.

Moreover, we can hypothesise that remote display nodes can show synchronization problems for highly synchronized content such as videos or simulations beside latency. We even observe these kind of problems in local setups. GDO offers an inter-node synchronization mechanism for apps to utilize in this type of use. This mechanism synchronizes clocks on these nodes and enable developers to have commands executed on these synchronized clocks. For example, if we have a play action broadcasted to 32 nodes on a remote location for a video, the arrival time of this message will be much more varied than a local setup, thus the video will not be played in sync on those remote nodes. On our heartbeat mechanism for synchronization we implemented, developers may define commands that execute on sync. In summary, server constantly sends heartbeats to sync local clocks using javascript libraries to under 30ms variance in real time. When a play action is sent with sync option, it is delayed 1-30ms on each node, and executed in real-time sync. So, we overcome synchronization problems on dynamic visualisations in remote operation scenarios in GDO.

Casting: Finally, due to web based operation and dynamic deployment and operation capabilities provided by CVSP, we can cast control interfaces of apps to other clients in the system. This mechanism works through dynamically loading components of an app into an interface by the order of the server. Through it, we enable what we call a console interface, which can be used for two purposes. First purpose is to cast touch capable control interface of an app to a limited user interface on a large touch screen in the LHRD environment, so multiple spectators can become limited users where admin can cast control of an app to this console and enable these users to interact with that app under supervision of the admin without fully exposing system management to them. Similarly, a control interface of an app can be casted by the admin to a tablet a limited user holds, and thus the admins can give controls to selected apps to individual users without exposing full system management functionality.

Overall, GDO can provide interoperability with web-based visualisations similar to SAGE2, but unlike SAGE2, it is the only platform to integrate other LHRD systems for dual operation. In terms of flexible setup, both SAGE2 and GDO are very flexible due to using web-based technologies. In terms of remote operation, GDO can be deployed remotely and apps can use GDO inter-node synchronization mechanism to sync animated content on screens. Furthermore, GDO control interfaces can be casted to other clients for limited user interaction discussed in 3rd chapter.

6.11 Developer Support for Distributed App Development

Another challenge mentioned in previous chapter is developer friendliness. This section explains how we managed this in CVSP software paradigm. Over tens of apps were built in GDO and to ensure easy implementation on these, we implemented certain features through or within CVSP and our general architecture. First of all, for data visualisation apps, we enable web based front-end and C# backend. This enables developers to create easy and rapid web visualisations and interactive control interfaces on front end and use C# for proper backend coordination and processing and data driver libraries.

CVSP architecture structures the components and communication scheme

only. Therefore on these components, the developers are free to develop in given languages extensively. On the frontend, developer may use many web visualisation or interaction or other supplementary libraries or pages. CVSP and management system can easily load these on-demand and clean these safely after termination. On the backend, developers implement applications on C#, where they can use many .NET libraries, utilize threaded approach and benefit from more structured coding, while doing all these separate from the core project of GDO. Furthermore, they can utilize hardware resources for coordination or data processing. Again the system handles dynamic creation of threads as instances and provide clean termination of them after that instance is unloaded.

On the communication side, developers can implement the API of the app which is between its server component (hub and instances), its node and control client components and thus have many coordination, data distribution, rendering, processing and interaction scenarios, which is the most important part. Through this, developers can implement apps with various data distribution, processing, rendering, complex interaction and many other features that are not easily possible or not possible at all on other LHRD software. Moreover, we let developers to implement configuration mechanisms, and thus they can have various system supported JSON based configurations available to them to be used as freely as they like on all components where the system handles distribution of the configurations. Therefore, developers can implement general data visualisations that can be used for various different projects or purposes.

All these help developers to implement diverse general or custom data visualisations with dynamic frontend, interactive control panels and capable backends. On top of all these, due to flexibility of the system, these developers can develop on a single computer, conduct initial test on few browsers. Afterwards, load the app to an LHRD and with the help of distributed utilities such as maintenance mode and remote command execution, which is implemented on CVSP, they can debug their app on LHRD.

Finally with composite application paradigm and CVSP, developers can use system functionality offered on APIs within their apps, thus load, unload and arrange other apps and use them to visualise their data in a coordinated and synced way through CVSP. Overall, developers with various skill-sets can create simple to

complex applications to GDO as tens of apps have been developed internally in Imperial College within GDO project duration as next chapter discusses.

6.12 Conclusion

In conclusion, we have a novel objective of providing web based and interactive visual data analytics techniques for multi-disciplinary data-driven research on LHRDs in order to immersively and socially perform visual data analytics. We have surveyed existing literature on 2nd chapter and explored design landscape for a such platform to conduct this on the 3rd chapter. The findings of these chapters are many requirement and features that has to exist simultaneously on the target platform, which were listed at the end of the third chapter. For this, we have developed a software paradigm to enable the functionality to overcome these challenges. In this chapter, we have introduced this software paradigm and how we used it to overcome unique challenges listed in the 3rd chapter.

CVSP's novel paradigm designed for LHRD, allows us to have system and app components on all physical locations modularly and dynamically and have a permissive API based communication between these components both locally and remotely. This allows us to create functionality dynamically through pipelines, such as a control interface component calling a component on the server calling components on cluster of nodes; or we can have applications to use system API to deploy apps and control them; or have utilities to work over apps and many other; while apps being able to utilize many external libraries behind this schema on server and clients. Overall, CVSP provides an architecture where modular distributed components that can have all to all dynamic operation and thus be able to create mesh of functionality on-demand in an LHRD environment.

These base functionality enables us to create visual scalability, heterogeneous app deployment, modular, general, interactive applications, a novel management system, collaging and syncing applications, creating composite applications, having system wide utilities, integrate devices where control and data of them can be provided to other components, app developers to be able to implement many data acquisition, processing and distribution mechanisms, system to have flexible setups, integration of many web based visualisations and projects, remote operation capabilities and finally offer many development options and utilities to app developers

to implement basic to advanced apps on the system. Thus, achieving most of our goals which are novel or critical, but most importantly never provided together in any other system in the literature, eventually enabling immersive and social visual data analytics for multi-disciplinary research projects on LHRDs effectively.

Therefore, through this functionality, we have witnessed that many apps were developed for GDO both general and custom. For example, a graph app for general graph visualisations, or chart app for chart visualisations, or maps app to create complex spatial data layers on LHRDs. These apps can be deployed in any layout and scale accordingly, be interacted and be used with different data and configurations. Therefore, many research during this project duration, have used many apps developed in GDO to visualise their data for visual data analytics.

For example, a transport research like shanghai metro, uses various apps to visualise its data like maps and charts even sync these apps on screen-space. Similarly, a medical research can use a different set of apps or similar set of apps as it requires. It can use the same map app to show spread of diseases, or chart app to show ratios. These apps can be animated, collaged, synced or interacted. Other online visualisations can be used within the system. Different devices can be connected to the system and apps modularly. Or many other features that we have stated in this thesis can be conducted. None of these are possible with other LHRD software as whole, and some of them are not possible at all. As a result, other systems do not support general, modular, dynamic, interactive, scalable, collagable and synced data visualisations on LHRD with suitable management mechanisms. GDO through CVSP enables all the features discussed in this chapter and thus enables Immersive and Social Visual Data Analytics for multi-disciplinary data-driven research on LHRDs overall. In the next chapter, we will introduce the applications of GDO as a proof of these contributions.

Chapter 7

Applications of GDO

GDO has been used extensively during its project duration. It has been used as the main software for KPMG Data Observatory at Imperial College for many multi-disciplinary research and projects. GDO also have been used in many visits to Data Science Institute (DSI) at Imperial College London. DSI is constantly visited by students, academics, business representatives and government officials. GDO has been used to demonstrate various projects in these visits. We have listed most notable of these visits in table 7.1.

Table 7.1: Important Visits that GDO Software Framework and its apps were used for demonstrations at Data Science Institute, Imperial College London

Guest	Date	Ref
Lord Bates	August 2018	[57]
Environment Secretary Michael Gove	May 2018	[101]
Chinese Secretary Cai Qi	May 2018	[19]
Portuguese Minister Prof. Manuel Heitor	April 2018	[119]
President of Estonia Kersti Kaljulaid	March 2018	[61]
ICL President Prof. Alice Gast	November 2017	[120]
Digital Secretary Karen Bradley MP	November 2017	[120]
The Duke of Cambridge Prince William	October 2017	[121]
The Duke of York Prince Andrew	June 2017	[57]
Japanese Ambassador Koji Tsuruoka	June 2017	[57]
Norwegian Minister Jan Tore Sanner	May 2017	[106]
European Commissioner Carlos Moedas	September 2016	[65]
Central Bank Governors	July 2016	[57]
President of China Xi Jinping	October 2015	[83]

GDO also has been used as a software framework for many student projects as these students base their masters or bachelor projects on GDO.

Table 7.2: Student Projects that have used GDO Software Framework for visualising their thesis or implemented a GDO App for their project

Year	Type	Student	Project Title	Ref
2017-2018	MSc Thesis	Arun Kothari	Machine learning for Bitcoin address classification	[97]
2017-2018	MSc Thesis	Aditya Chaturvedi	Identifying users on the Bitcoin network using machine learning and graph visualisations	[79]
2016-2017	MSc Thesis	Aodhgan Gleeson	EthCon Viewer: Visual Analytics for Classification of Ethereum Cryptocurrency Transactions	[62]
2016-2017	MSc Thesis	Huangsong Wei	A New Method of Twitter Exploration: Time Sequential Hashtag Co-occurrence Networks	[7]
2016-2017	MSc Thesis	Qile Huang	Visualizing Netflow Patterns in Large-Scale Enterprise Network	[165]
2015-2016	MSc Thesis	Jiaxin Qiu	Big-Data Detection and Classification of Algorithmic Behaviour in the Bitcoin Blockchain	[21]
2015-2016	MSc Thesis	Guillaume Paillot	Scaling Data Visualizations For Data Observatories	[134]
2015-2016	MSc Thesis	Yaozhong Zhang	Immersive Video Wall Presentation Tool	[80]
2015-2016	MSc Thesis	Christopher Snowden	Twitter Observatory: A Set of Twitter Visualisation Tools	[156]
2015-2016	MSc Thesis	Phil Brezovszky	Developing a System to Visualise Clinical Scan Data on the Data Observatory	[47]
2015-2016	MEng Thesis	Siddharth Vasu	Distributed Spreadsheet Visualisation and Analysis on Video Walls	[55]
2014-2015	MSc Thesis	Dan McGinn	Bitcoin Observatory in the KPMG GDO	[22]
2014-2015	MSc Thesis	Min Chia	Visualisation Framework for Large-scale Graph using Browser-based Distributed Rendering	[162]
2014-2015	MSc Thesis	Wei Zhang	TimeV: Real- Time Big Data Interactive Visualisation on 100-Megapixel Screen with Spark	[152]
2016-2017	Bachelor Thesis	Aneeque Safdar	Ethereum Transactions Visualisation	[63]
2016-2017	Bachelor Thesis	Tian-Ren Chen	A More In-Depth Analysis of Twitter User and Brexit	[5]
2016-2017	Bachelor Thesis	James Grant	Generative Procedural-Parametric Architectural Design	[72]
2015-2016	Bachelor Thesis	Daniel Thomas	Distributed Ray Marching Library for a 3d Fractals Project	[54]
2015-2016	Bachelor Thesis	Finlay Curran	Towards a 100 Megapixel Game Engine	[153]
2015-2016	Bachelor Thesis	Suryansh Rastogi	BitcoinViz: A Set of Visualisation Tools for Bitcoin Transactions	[23]
2015-2016	3rd Year Project	Student Group*	Hercules: Interactive Data Visualisation on 100megapixel Video Wall	[39]
2014-2015	Internship	Evann Courdier	Distributed D3	[39]
*Group Members: Lorenzo Paoliani, Samuel Budd, Lorenzo Calejari, Ioana Radu, Oliver Robinson, Xingze Tian.				

Furthermore, GDO use was also directly or indirectly appeared in the media or used in media documentaries or show-cases for DSI. It have appeared numerous times in Imperial College News, but also on external media organizations as news articles, analysis and even as documentaries. You can find the list of notable external media appearances in table 7.3.

Table 7.3: GDO Software Framework showcases or use appeared on media

Source	Title	Ref
BBC	The Secret Science Of Pop	[149]
BBC	Mars: Red and Dead?	[57]
Financial Times	Inside Europe's biggest data visualisation laboratory	[85]
Guardian	Why data is the new coal	[170]
Motherboard	Europe's Largest Data Observatory Turns Big Data into Big Images	[66]
V3	Imperial university opens big data observatory to boost analytic insights	[82]
Medium	Digital Strategies in London's boroughs: state of play 2018	[48]
CCN	London Institute Uses 64 Monitors to Demonstrate How Bitcoin Works	[96]
China Daily	Pharmaceutical tycoon's college challenge	[118]

Finally, many applications have been built for GDO during its project duration for diverse purposes. The following sections goes through these applications. (Some of these apps are developed by the author directly, some of them are developed in collaboration with the author and other colleagues or students, some developed by other colleagues or students directly but aided and guided by the author; for the rest, the author provided the system (GDO) and its software paradigm (CVSP) with documentation to enable development and utilization of the apps. So, the author has four levels of contribution in these applications thus indirectly in the many research these apps used.)

7.1 Map based Apps and Projects

Map based apps and projects aim to visualise large scale maps on screen-space. These maps can be interactive, where users can zoom in and out or perform pan action on the map. Furthermore, these maps can be animated through animation logic we implemented. We use OpenLayers library [111] for creating the partial visualisations on node and control interfaces. In almost all map apps in GDO, global map parameters are stored at server and distributed to nodes. Conversion to

local map parameters of partial map can be done at nodes or on the app instance in the server. Nodes receiving local map parameters visualise their part of the map. There are few apps that visualise maps in GDO framework. Two of these are generic and can be used to visualise variety of maps. Others are project specific and implemented prior to our generic advanced maps app. Other than these, there are various public or private visualisation projects conducted in DSI using these apps.

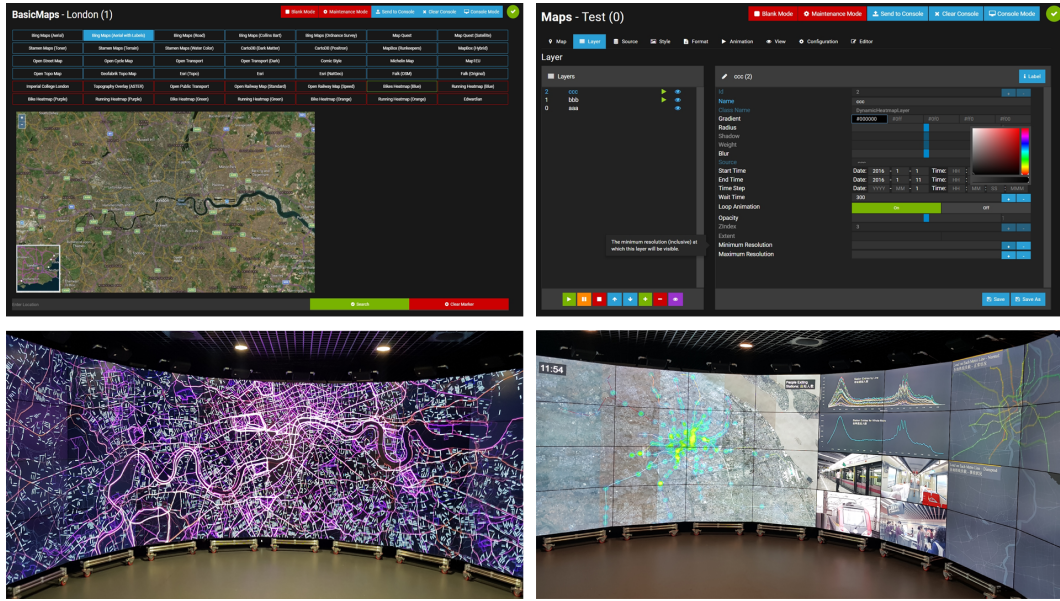


Figure 7.1: Map based App Control Interface (top) and On-Screen Visualisation (bottom) Examples, including Shanghai Metro demo which was displayed at President of China visit (bottom-right)

A. BasicMaps: BasicMaps is our first mapping application. The proto version of this app goes back to pre-GDO period and is actually our first application in the platform. This app visualises various predetermined maps and overlay data. It has two sections in its control interface. First one is for maps used as base. Second one is for overlays displayed over base maps. Users can select any base map and many overlays for a visualisation. Maps in this app, like every other app, can scale to all screens of LHRD. Control Interface also has an interactive version of the overall map, where user actions on this, reflected to distributed map on screen-space, such as zoom and pan. Also, users can search for locations on the control interface version

of the map.

This app uses OpenLayers base JS code on both control and node interfaces as core library. It loads all pre-determined maps when app instance is deployed to this library. Server determines which of these are visible and overall coordinates of the map. This is of course controlled by control interface of the map, where actions are relayed to the server and the server distributes these to nodes and control interfaces. Receiving interfaces, which are both nodes and control, calculate their local parameters through an algorithm we implemented using CVSP of the system and update visibility of layers with information they received. Overall, BasicMaps have easy to use interface and quick demonstration for deploying large-scale interactive maps with overlays to LHRDs.

B. Maps: Maps is the advanced generic version of BasicMaps app. It is implemented after many BasicMaps variants were implemented for specific projects. It was decided that a general maps application is required for preventing implementing duplicates for every need. Instead, users can utilize this general version and just change configuration. It uses the same logic but instead holds much more complex structure on the server for synchronization with clients. It can have many tile sources, map layers, styles and animations. Users can create layers, delete layers, determine order of layers. Users can import new map sources as local or external tile services in many formats and standards. Users can create styles for data overlays on maps.

In this version, we also enable animation of data sources. We created an animated source and layer type. This stores various spatial data in temporal format, so the app can animate through time. This uses global time, thus multiple overlay animation layers can have data through multiple spans of time with different frequencies and overall system can animate all together on sync. Furthermore, users can decide how many steps this span of time is divided into and how much each step takes in real animation time.

This app also allows users to change visibility of layers, or start animation of individual layers or even use an online editor to create data layers. Furthermore, users can save and load configurations of projects they make inside the app. Finally, it allows deployment through CVSP, thus other complex apps can deploy and control almost every functionality of this app through CVSP without the control interface

and use it to deploy every kind of interactive and animated maps across the LHRD. This is one of the complex use cases of GDO Apps, which can manage and allows creation of multi layered, interactive, animated maps for LHRDs.



Figure 7.2: Maps app being used for Anti-microbial Resistance Visualisations.

C. Other Maps Apps and Projects: There are various apps and projects that utilize or modify BasicMaps or Maps apps to visualise spatial data. Here are few public examples implemented in GDO:

- **Shanghai Metro:** is a use-case app which is first implemented for the visit of President of China. It uses Shanghai Metro data and camera feeds to create visualisations of people coming in and out of stations and load on stations. It uses a version of BasicMaps with an animation overhaul for animating people coming in and out of stations. This project also features a DD3 visualisation for visualising loads on lines. This app is also first example of inter instance communication. Users can load two different configurations of this app and it syncs animation instances of it temporally using the app scope.
- **London Cycles:** is a use-case app that animates TfL's data on cycle hire stations across London.
- **Leeds:** is a use-case app that was created as part of Jaywing Collaboration [89] to show footfall data in a shopping center in Leeds.

- **UKTravel:** is a animated telecommunication data across UK from Telefonica.

There are other major and minor examples of how map applications in GDO is used. However, after the implementation of generic maps app, many researchers used this new version to visualise data through this instead of implementing their own app, such as Dr.Ceire Costelloe on the research of Anti-microbial Resistance.

7.2 Graph based Apps and Projects

Graph based apps aims to visualise large scale graphs with thousands even millions of nodes and links on the distributed screen-space. These apps either provide generic graph visualisations where users upload or link graphs in various formats and determine visualisation layout and style through configuration or control interfaces of the apps. GDO has two generic graph apps in its library. One of them uses our own libraries for graph visualisations and format and built just for GDO. The other one uses sigma graph libraries [137] for visualisation while using GDO system for distribution and control mechanisms.

Other than these, there are few crypto-currency apps and projects that primarily use graphs to visualise their data. Most significant of these is Bitcoin app and visualisations. The other project is Ethereum visualisation project which visualises Ethereum transactions using GDO framework. These projects aim to analyse crypto-currency trade in these currencies through searching for patterns, trends and anomalies visually.

Finally, Graphs are used for other purposes such as visualising networks and cyber security related topics in GDO, for example visualising netflow patterns in large-scale enterprise networks.

A. Graph: The generic graph app enables users of GDO to upload graphML files to GDO server and visualise large-scale graphs in screen-space [162]. This app can scale to any node configuration. After uploading the graph data, it processes nodes, links and labels and saves it as temporary files on the server in JSON format. This temporary data is divided accordingly to number of screen nodes the graph app is deployed to. For example, if the graph app is deployed to 4 screen nodes, than there should be 4 files for links that each screen should use. This app also takes

into consideration the links running across a screen where neither of the edge nodes reside in the screen.

After the processing, each screen node reads data from corresponding files to render the partial graph in their screen. As a result, graph data is visualised distributedly. Users from control interface of the app can now control the visualisation. These controls include: color schemes, size of nodes, panning, showing or hiding nodes, links or labels. Furthermore, users can search for a specific node, show most connected nodes and filter the graph through it. This app also has zooming mechanisms which enables users to increase the size of the graph on screen-space. Zooming is performed by reallocating which screen component of the app reads which graph file. Panning uses buffering in order to render SVG elements beyond view of the screen.

As a result, users can easily upload graph files and visualise them in any number of screens. They can make minor style changes and control the visualisation orientation to a certain degree. This application can scale up to millions of nodes and links, although suffer from rendering duration due to limitations in rendering of SVG DOM elements in a browser. Tests have been made both with synthetic and real data scaling from 1000 nodes to 100000 and 3000 links to 300000 for this issue. Partitioning duration is affected by both increases in node and link numbers exponentially. It stayed stable for nodes, if number of displays increased and increases logarithmically for links as number of displays increased. Although, all these numbers are insignificant since they vary between milliseconds to maximum of few seconds. The real latency occurs on rendering process. Eventhough, rendering duration increases logarithmically, it starts with 2.5 seconds for 1000 nodes and 3000 links and goes up to 45 seconds for 100000 nodes and 300000 links. This originates from limitations of number of visual element renders in a browser DOM, thus introducing significant lags for GDO large-scale graph renders.

B. Sigma Graph: Sigma Graph app is another generic graph visualisation app in GDO library [39]. It takes on previous graph app and improves it. First improvement is use of sigma.js for graph rendering. It has other improvements in the back-end for partitioning. However, the main improvements are in rendering and controlling the graph. It allows selection of certain nodes and edges, allow filtering nodes, applying color schemes based on node or edge properties such as tags or weights. Again

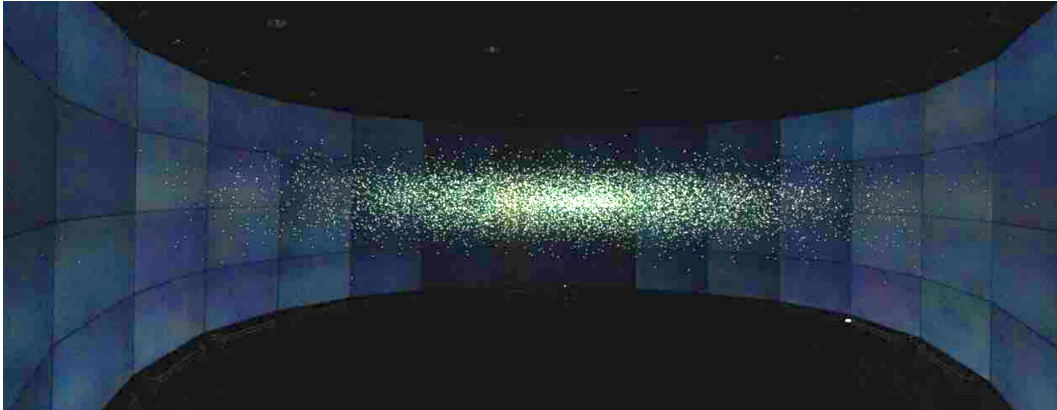


Figure 7.3: 64 Screen Graph with thousands of nodes rendered in Graph App [162].

similar to previous graph app, it allows panning and zooming. One other interesting functionality is that it allows customized functions to be applied through control interface on render pipeline. Overall, it is an improvement over the first graph app and allows more customization and rendering options over large-scale graphs in GDO.

C. Bitcoin: Bitcoin GDO app enables graph and other visualisations on bitcoin transactions [22]. It has two visualisation types. First one is not a graph visualisation. It is a real-time globe visualisation showing the nodes with their transaction activity on a globe. The second one is an interactive graph visualisation for showing both real-time and historical transactions in blockchain. This app uses again `sigma.js` for visualising its graphs.

Through this app, visual exploration of historical transactions of a block becomes possible. It enabled visualising different historically significant events such as first bitcoin transactions to major spam events. Real-time mode allows live transactions to be visualised as graphs. This app also has a `GigaImages` project which enabled visualising thousands of bitcoin visualisations to be displayed across the screen-space using `GigaImages` app simultaneously.

Overall, this app enables performing visual analytics on bitcoin cryptocurrency.



Figure 7.4: Ethereum (bottom-right) [63] and Bitcoin (others) visualisations on GDO (On top left BBC Presenter Spencer Kelly demonstrating Bitcoin Observatory Project [23])

D. Ethereum: Ethereum Visualisation is a group of projects intended to visualise ethereum account transactions and network attacks [63] [62]. Instead of being implemented as a separate apps, these use GDO's existing generic graph visualisation apps and other apps. One of these projects also has its own system to visualise these transactions in real-time. The purpose of these projects is similar to bitcoin projects and it is to explore ethereum crypto-currency transactions, anomalies, attacks visually and performing visual analytics on these topics.

7.3 Chart based Apps and Projects

Chart based apps aim to visualise large-scale charts on distributed screen-space. These apps and projects again are divided into two categories, first one being the generic apps for visualising charts and the second one is research based apps and projects. Generic chart visualisation apps have two purposes different than normal chart visualisation techniques. The first one is displaying these charts distributed across multiple screens. The second one is regarding complexity, and it allows charts with normal than higher complexities to be displayed in the large screen-space.

A. DD3: D3.js in other words Data-Driven documents [46] is a java-script library for interactive data visualisations. Distributed D3 (DD3) [52] app takes this library and integrates it to GDO framework for enabling custom visualisations and infographics to be implemented using D3. Developers using this library and app implements on D3 and this app takes care of loading these data visualisations into GDO and distribution of them to screens. In other words, a generic d3 visualisation can easily be loaded with this library. Users can load these visualisations as a DD3 instance with a certain configuration to any section of nodes. DD3 library distributes this visualisation to those nodes. Overall, this library enables us to implement complex and interactive data visualisations and infographics using d3 and load them to GDO as projects of DD3.

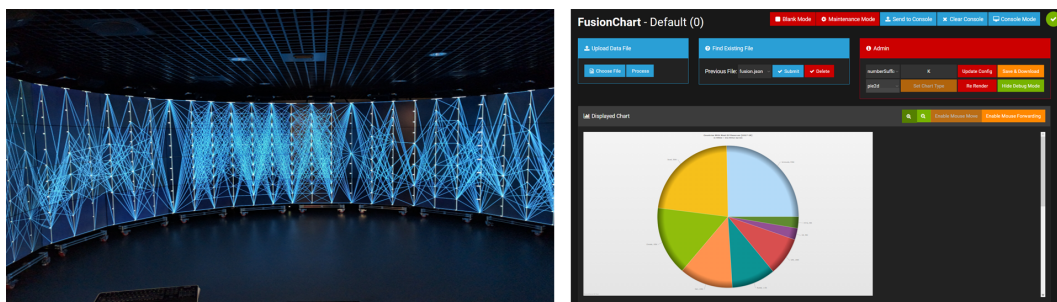


Figure 7.5: DD3 parallel coordinates chart (left) and Fusion Charts control interface (right)

B. Fusion Charts: Fusion charts is a generic easy chart rendering library in GDO. In contrast to DD3 which is based on implementations on top of D3.js and importing of these to GDO, FusionCharts app use fusion charts [70] data specifications and library to render basic charts with ease. In other words, through DD3, complex infographics can be implemented using D3 and be rendered distributedly on GDO and interactively controlled where control interface can also be implemented. On the other hand, using fusion charts format, fusion charts app renders basic charts with given data with limited customization but rapidly. It supports various chart formats and styles. The user uploads a fusion chart specific JSON file. This file specifies the chart type, properties, labels and includes data of the chart in a JSON. This app than visualises this on the screen-space distributedly. Afterwards, users can control certain properties of the chart and even change chart types.

In comparison to DD3, this app does not distribute the chart data but instead each node renders the chart and scales accordingly for partial visualisations, which is exactly how distributed html solution in GDO library works. Therefore, it is an easy to use solution, and provides rapid and fast chart visualisations. However, this app does not support complex programmability of DD3. Regardless, through those option, we can provide both an easy and complex solution for both basic to advanced charts and infographics needs.

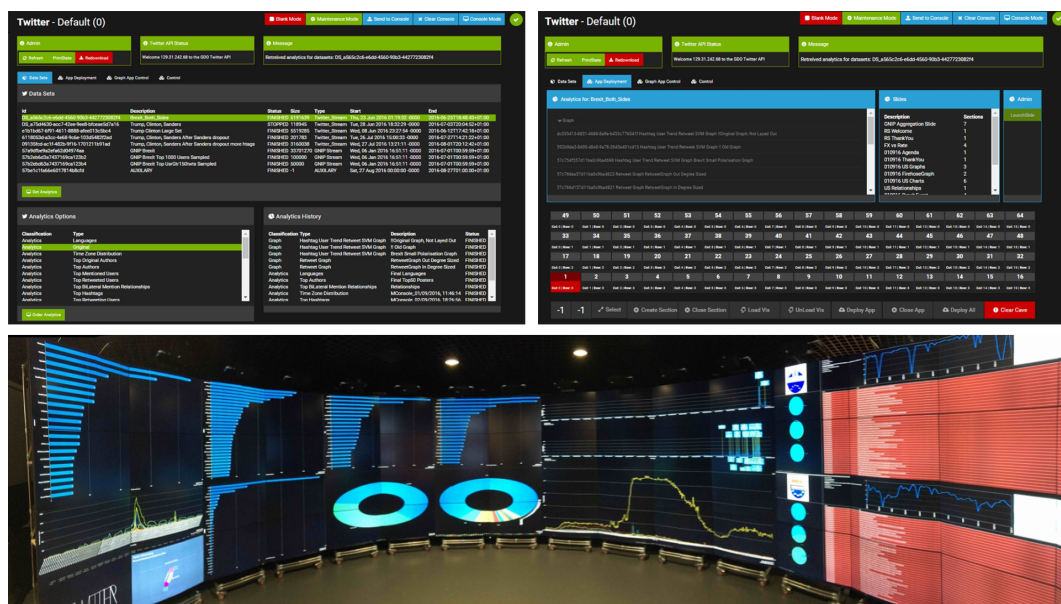


Figure 7.6: Twitter App Control Interfaces (top) and Visualisation (bottom) Example [156].

C. Twitter: Twitter App [156] aims to perform visual analytics on twitter data in GDO. This app uses both charts and graphs to visualise twitter data. It utilizes graph app, distributed html app, images app and fusion chart app to visualise data on the screen-space. It uses GDO CVSP to create sections and deploy these other apps with twitter data this app processes. Through this app, users pull twitter data from twitter API and store it in a mongoDB database. Afterwards, this app allows users to select datasets in the mongoDB database, select analytic options such as top authors, top retweets and many other. Finally, allow users to deploy these analytics as other apps such as charts, graphs, static visualisations as images to selected sections in GDO.

In the interface of the app, there are three panes: DataSets, App Deployment and Control. Datasets pane allows users to select datasets in mongoDB, select analytics options and see analytics history. Users can see the status of the dataset, such as if it has finished streaming from Twitter API or not. In the app deployment pane, users are given analytics from the selected datasets and options. Each analytics is deployable to certain number of sections. For example, one analytics may use only graphs and deployable to one section; another one may require due to its properties charts and graphs apps, thus deployable to many sections. These are called slides in the app. Users create sections and select slides to deploy to these sections. Users also can load or unload visualisations to existing apps deployed by this app. Users are also given options to re-download data. In the control pane, users can control the apps they deploy.

This is an good example of composite apps implementation. It uses other existing apps instead of re-implementing them, and use GDO CVSP to deploy and utilize them. User can select where to deploy and even control the deployed children apps in this scenario, where twitter app acts as a coordinating and data providing parent/composite app.

D. Sharing Economy: Sharing Economy is a visualisation project that aims to visualise longitudinal study of how sharing economy performed in previous years in UK. It looks into Uber, Airbnb other datasets and visualises in charts and maps the analytics of sharing economy in UK [42].

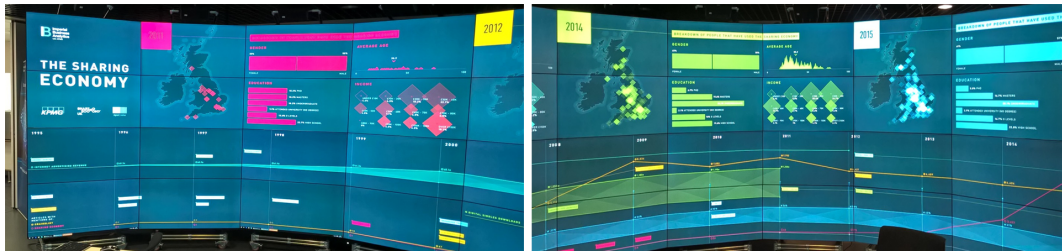


Figure 7.7: Sharing Economy Visualisation

7.4 Multi-media based Apps and Projects

Multi-media based apps aims to visualise media formats such as images, videos and 3D content on GDO. These apps use various distribution mechanisms to distribute these media based content to the screens in GDO. These apps are useful as supplementary visualisations for loading static images of data or relevant content, load videos or presentations to GDO or visualise 3D content distributedly in screen-space.

A. Images: Images app enables users to deploy images to GDO nodes. Users upload via control interface an image file to the system. Control interface transfers this file to the instance of the app. Depending on node layout of the app deployment, it crops and creates partial images of the actual image. It upscales it during the process depending on section resolution. Afterwards, it distributes these images to the nodes of the app. Nodes display their partial image, thus making the whole image appear on the GDO screens.

App interface allows panning the image, fitting or filling the canvas and even rotating. On each event, from the original image on the server, partial images are re-created and re-distributed to nodes. This app also can be controlled through CVSP similar to other chart and graph apps, thus enable it to be deployed part of a composite app. As a result, various supplementary images or static images of data visualisations can be deployed to GDO.

One notable use-case of this app to visualise images sent by NASA's Curiosity robot on KPMG DO as part of a collaboration [57].

B. GigaImages: GigaImages app is an image display app that aims to display images with extra large resolutions such as millions of pixels. These images are stored in different formats such as Deep Zoom Images (DZI) or other tile-based image storage standards for storing large zoomable images. This app uses OpenSeaDragon library [112] to display these images. Similar to maps app, it creates an overall canvas in the control interface where global coordinates are produced for the image and distributes these to nodes where nodes calculate local coordinates for their OpenSeaDragon render. These nodes load the canvas with node coordinates and zoom into local coordinates of the image they calculated. Thus, a user can select a

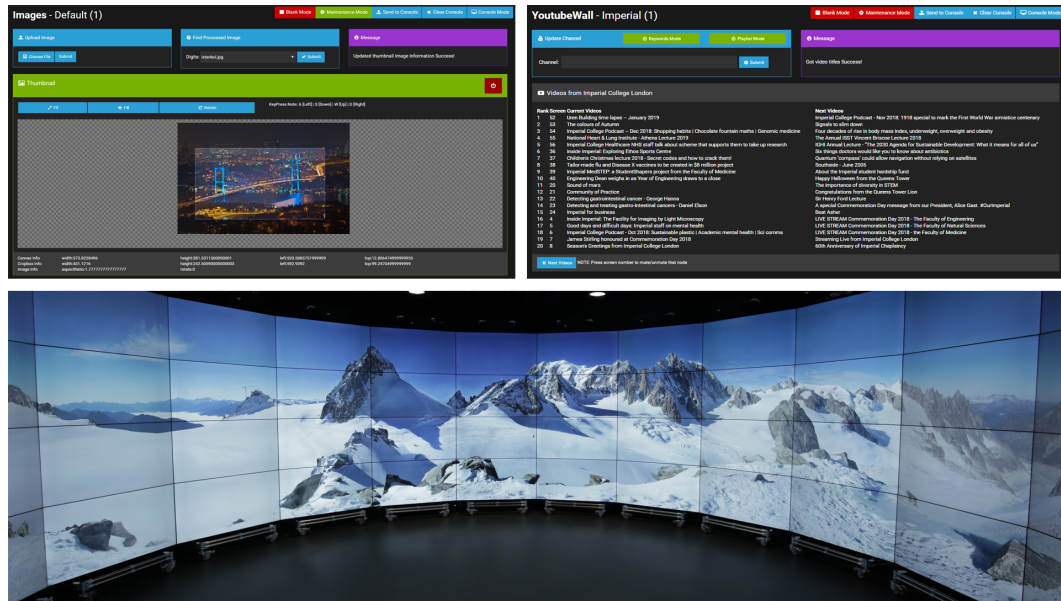


Figure 7.8: Images (top-left) and YoutubeWall (top-right) App Control Interfaces and GigaImages Visualisation (bottom).

gigaimage in the control interface and pan and zoom into it, while these actions are replicated in the nodes smoothly. This is useful for projects that data is catalogued as visualisations such as bitcoin patterns or very large images like stars in the universe. This app enables zoomable interaction with those large image visualisations.

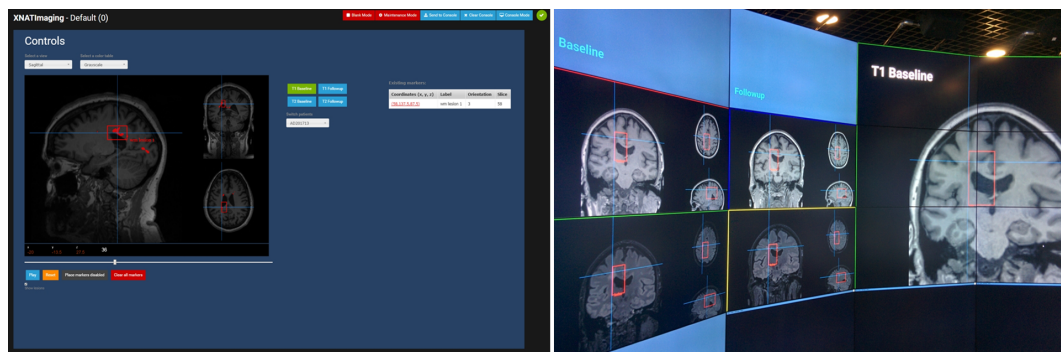


Figure 7.9: XNAT Imaging App Control interface (left) and Visualisation (right) Example [47].

C. XNAT Imaging: XNAT Imaging app is used for analysing medical scans of brain for diagnosis of Multiple Sclerosis (MS) on a patient [47]. LHRDs provide both comparisons and high-resolution displays for these scans and thus enable easier visual diagnosis in comparison to other visualisation options available to radiologists and clinicians. This app allows uploading of MRI images to be visualised in LHRD. Users through control interface can add regions of interest, switch patients, render MRI images, navigate 3D MRI images, animate these images over time, select views and decide color schemes. As a result, this app provides a specialised experience to images app for the task of diagnosis of MS in patients.

D. Presentation: Presentation app is basic presentation tool for GDO. It works similar to Images app; and instead of images, it allows uploading of office formatted presentations to the instance of the app. Afterwards, it distributes these presentations as images to the nodes of the instance. It gives user the options to navigate in the presentation through previous and next buttons. It also displays the slides in control interface for presenter to view. As a result, it provides easy to use presentation app in the GDO with basic options and controls.

E. PresentationTool: PresentationTool app [80] is a complex presentation tool in GDO. It allows images, office presentations and videos to be deployed as a meta-presentation to screen-space. It uses Youtube, Images and Presentations apps as a composite app, and use them to display these media formats. There are meta-slides in this presentation. In each meta-slide, user decides where are the sections and in which section which resource is deployed. For example, in meta-slide 1, user deploys slide 5 of a presentation to section 1, an image to section 2, a video to section 3. In the next meta-slide, user decides again what will be on the overall screen-space with these resources. So, app allows individual office presentation slides, images and videos as unit resources and in each meta-slide allow users to deploy them to any section in any combination. After the presentation is ready, the presenter can navigate through meta-slides which changes the overall screen-space.

Users can also zoom in and out of resources on sections and individually control them. Furthermore, users can use a paint tool to paint over individual resources during the presentation. In other words, presenter on meta-slide 3, can use touch screen or mouse to paint on a resource that is deployed to the screen-space. This



Figure 7.10: PresentationTool App Control interface (left) and Visualisation (right) Example [80].

allows presenters to highlight important aspects of a slide. Moreover, there is a built in voice assistant that presenter can give verbal commands to. It listens to commands such next, previous, load, play and others and takes according actions.

As a result, this app allows complex presentations to be given using GDO framework on LHRDs. It can distribute any resource (slide, images, video) to any section on a meta-slide and allow presenters to navigate on meta-slides. It is another example of composite apps as it uses images, youtube apps to display images and videos; and built upon existing presentation app itself.

F. WebGL: WebGL app [153] is a generic app that allows developers and users to render 3D content on the screen-space using WebGL [169]. It uses babylon.js [18] for rendering. It loads the scene on both to control interface and nodes interfaces. Similar to maps app, it calculates a camera orientation in the control interface and distributes this to node components of the app as camera view changes. Although, it performs this in 3 dimensions instead. Nodes calculate their local camera orientation through an algorithm and update their view. This app loads meshes and resources to 3D scene selectively on the nodes. Each partial visualisation, loads resources and meshes to produce 3D image accordingly. Thus, overall 3D scene is produced distributedly.

It allows users to perform fly-by in 3D content through navigating on control interface. It provides options through keyboard such as movement speed of camera, toggling mesh bounding boxes, locking horizontal or vertical camera rotation, toggling visibility of meshes. In the interface, it gives the options of resetting posi-

tion, enabling no-clip and gravity, showing stats and finally toggling frame by frame synchronization to reduce tearing in the cost of performance.

As a result, users and developers can use this app directly or as part of another composite app to deploy 3D content on LHRDs for visualisations.

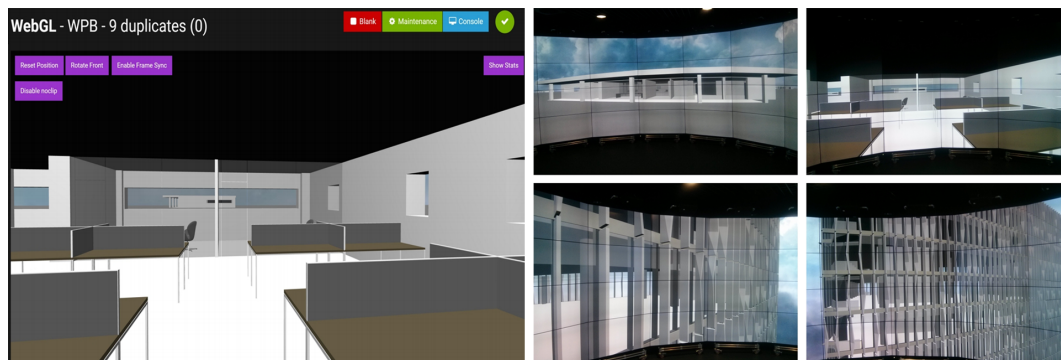


Figure 7.11: WebGL App Control interface (left) and Visualisation (right) Examples [153].

G. Youtube: There are two apps for playing youtube videos on GDO. First one is called YoutubeWall. This app does not scale youtube videos to multiple screens, instead display one video per screen. It allows users to select a channel or playlist and display many videos of these on the screen-space simultaneously. So, if there are 10 videos in the playlist and 5 screens, it will display 1-5 videos on screens. Users can also navigate to next videos. This app uses youtube API and create a youtube canvas inside the node frames. It is mostly used to cycle supplementary videos as a wall in an overall data scene.

The other app is called Youtube. This app again uses youtube API to render youtube videos but enables scaling of them across multiple screens. It allows users to deploy high res (4K, 8K) videos to large screen sections. It uses similar coordination logic to distributed html and calculates partial youtube canvases. It allows users to select videos, play, pause, stop, forward, rewind. It also shows the video on the control interface. It utilizes GDO software frameworks high frequency time sync mechanism. This mechanism syncs javascript times of nodes with server in periodic intervals, thus enabling nodes to be synced in time (less than 30ms). Therefore, apps like youtube can sync partial videos with high accuracy. As a result, this app can

be used both individually or part of a composite app or scenario to deploy videos on multiple-screens of LHRD through GDO.

H. Fractals: Fractals project is a visualisation project to render fractals using GDO software framework [54]. It has two apps under it. First one is the original fractals app which renders fractals. The second one uses Ray Marching method for rendering fractals which intends to improve performance of realistic lightning and shading effects such as soft shadows and reflections in webGL. These apps uses webGL to render fractals. In the control interfaces they offer control of camera through a virtual pad that is used for look and move. Moreover, these apps allow frame level synchronization, which users can turn on and off from control interfaces. These projects gives important insight on distributed 3D content rendering and ray marching, in addition to visualising complex fractals on screen-space.

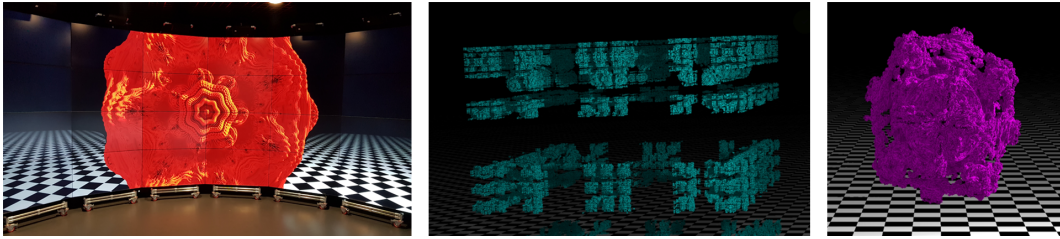


Figure 7.12: Fractal Visualisation Examples from Fractals and RayMarching App in GDO [54].

I. Static HTML: Static HTML is an app that aims to render HTML web pages distributedly on GDO screen-space. It duplicates the existing web page on the nodes and scales and orients them to show a unified web page on screens. It works with web pages that has static or timed content. However, if the web page has interactive or random content, there would visual artifacts produced by the app. For example, A news web page where on each load it shows a random news on the page. Since we do not duplicate the DOM and re-distribute it, each partial visualisation would be showing a different state of the web page and thus fail. If the events of the web pages are timed like periodic or continuous animations, this app can show these perfectly, since these are identical on all duplicates.

This app currently doesn't implement interaction too, but we have designed

how this could be possible. It will use control interface as input canvas which shows the original web page and mouse pointer, and interaction events will be distributed to nodes. Although, interaction like logging in can create serious problems as duplicate logins will be made. Another problem is certain web pages can block browsers from running them inside an iframe through X-Frame-Options [173]. Therefore, some external web pages can not be run inside GDO framework due to this cause.

This app can also be used as part of a composite app. A composite app can use an external visualisations system such as charting a visualisation and use this app to visualise that chart as a web page in a section. This allows, that chart which is visualised externally to be displayed in the GDO. Furthermore, the updates to the chart or animations, as long as they are not random, will be reflected since duplicates of the chart is running on each node of the app. If the external web server broadcast an update to the visualisation, each node will receive it and update their canvas.

This app also provides visualisation projects to be possible with team members that are not skilled in developing apps. Thus they can use external online resources or general web technologies to create web based visualisations externally and display them in the GDO.

7.5 Integration based Apps and Projects

Integration based apps are group of apps that aims to integrate GDO software framework with other systems and devices.

A. SAGE2: This app provides an easy integration with SAGE2 platform [126]. SAGE2 has a similar technology background and layout to GDO, thus gives us a unique opportunity to integrate these two. SAGE2 runs display interfaces and server control interface. In SAGE2 app, we run these display interfaces inside node interfaces of GDO and run server control panel of SAGE2 inside control interface of the app. As a result, GDO can connect to SAGE2 server and render each display interface and server control panel of SAGE2 will be displayed in control interface encapsulated.

B. tranSMART: DSI has a eTRIKS/tranSMART visualisation application [155] for visualising data related to eTRIKS project [64]. This app works through a similar distribution mechanism to GDO as a control interface coordinates section interfaces

of the application using web sockets. Instead of nodes, this runs in columns. Thus have to be deployed horizontal node sections only. Similar to SAGE2 integration, we encapsulate these column interfaces inside node interfaces of the GDO. Again, we encapsulate control interface of this external application inside control interface of this app in GDO.

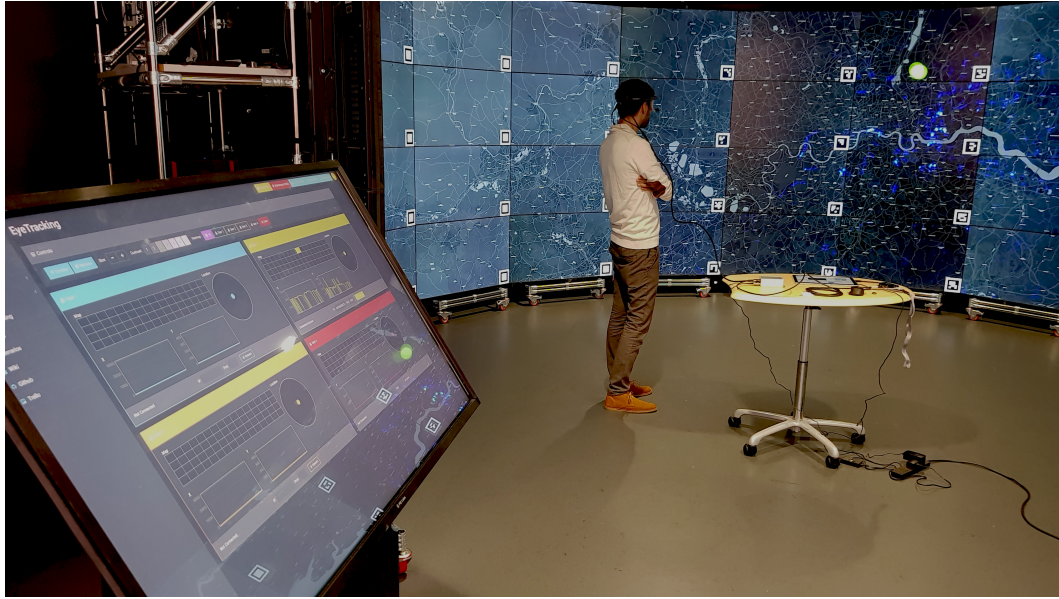


Figure 7.13: Eye-tracking Module using QR codes on screen with real-time pointer that creates a live heatmap and controls running on console interface.

C. Eyetracking: EyeTracking is a module app that runs in the background of the GDO system, which connects eye tracking devices to the system. It currently only works with Ergoneers Dikablis Eye Tracking glasses and D-Lab software [69]. This module supports up to 4 users. Each user connect to their D-Lab software running device which the eye tracking unit is connected through module's control interface. This D-Lab software streams the data of the device to the eye tracking module in the GDO server. This module displays QR codes on screen space as an overlay on lower right corner of each screen. D-Lab software recognizes these tags and calculates their location on the overall area the unit is seeing. It sends to server coordinates of QR codes in current view and coordinates of the users gaze on the view. Thus, it allows us to calculate where the gaze is exactly on the screen-space.

This module can display a virtual pointer on the overall screen-space color coded to the user on where the user is gazing in real time. Furthermore, this pointer can draw an heatmap of gaze as users continue to examine the screen-space. It creates 5 heatmaps, one for each user again color coded in the user and one meta heat-map for overall user gaze data. As users continue to examine the screen-space heatmap generation continues real-time. Users can clear individual and overall heatmaps and restart them. Users can toggle heatmaps and on-screen cursor on and off through the interface. Users can also change size and contrast of QR codes to adjust light setting of the environment for more accurate tag reading.

The interface also displays overall data on each user. As where they are looking in the LHRDs. It also uses a location calculation algorithm to determine the location of the user inside the LHRD. This algorithm uses the sizes of the QR codes and orientation of them to calculate the location of the user. This will work only for LHRDs with circular or wall shapes.

Overall, this module can be used to study where the spectators most examine and focus. Spectators in the environment can wear these head units and be shown various visualisations without the heatmap and cursors showing. Afterwards, these heatmaps can be turned on to see where on the visualisations the spectators focus most.

D. Hololens: Hololens project aims to integrate Hololens device to GDO and enable it to be used in KPMG DO [16]. It uses markers to calibrate the device to the orientation of the screen-space. Afterwards, the system knows where the user is gazing in the screen space. It does not require markers constantly displayed like eyetracking since the device has a gyroscope and can track its orientation after the initial calibration. One project example, as seen in the figure 7.14, is showing additional data on Augmented Reality over the twitter visualisation. These kind of projects enable private enriched data to be shown to users wearing AR devices.

E. EEG: EEG has not been directly integrated to GDO. However, as part of Jawyng collaboration with DSI, data from spectators has been collected for behavioural analysis using EEG devices and GDO framework where used for visualisations for this research [89].



Figure 7.14: Hololens project with Twitter App Integration where on screen AR overlay can be seen on the left image [16].

F. Kinect: KPMG Data Observatory also has 5 Kinect devices deployed in the observatory. This project intends to collect data from these devices and create and overall data from all angles regarding inhabitants of the environment. It allows spectators in the environment to make physical gestures to control the environment. These gestures can be used to control GDO and data can be used by apps of GDO for various purposes. Several developers, including original developer of Kinect project put integration of this distributed Kinect system with GDO framework as the next objective on the roadmap for this project for enabling interaction with data through multiple Kinect devices [3].

G. TimeV: This app aims to connect GDO framework with Spark on the DSI Cloud. Spark is cluster processing framework based on Hadoop [74]. This app allows to process queries on Spark cluster and visualise the results. Users can deploy this app to any section, they can execute queries on the spark cluster and visualise the results in the GDO [152].

H. Temperatures: Temperatures app is part of WikiSensing [171] and Concinity [113] projects. In DSI, there are many MSP430 based light and heat sensors, especially in the LHRDs environment. We have reprogrammed these sensors and their host computer to send data to WikiSensing system. This system is a collaborative sensor data analytics platform built in DSI. This app receives these sensor data, and displays them in the screen-space. The temperature visualisations across screens are aligned with location of sensors. In other words, screens show tempera-

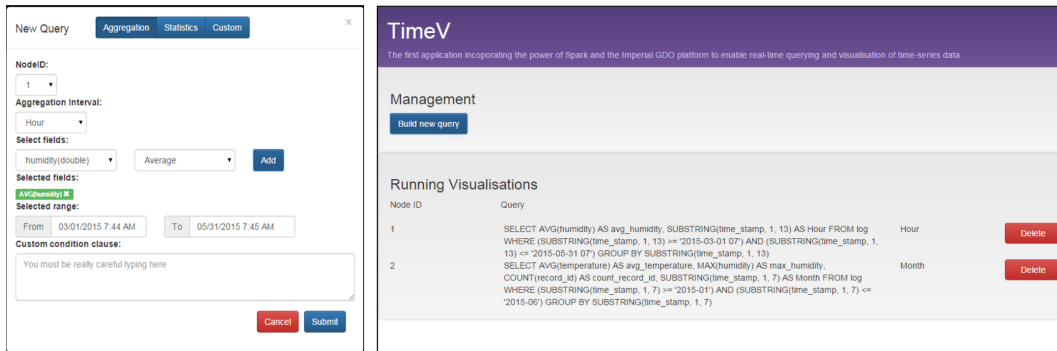


Figure 7.15: TimeV Query Interface and Control interface where users can input Spark queries to be computed and visualised in GDO.

tures calculated from sensors close to them. This is a basic example of how LHRDs can be connected to Internet of Things.

I. Cortical: Cortical app is integration of cortical.io to GDO framework. Cortical.io is a set of AI tools for text analysis. They use retinaSDK to extract keywords, compare text, detect language, filter tweets and many other [38]. This app enables this functionality for GDO. Users are given 3 configurations for modes such keyword extraction, compare text and others. For example, when user loads a cortical app with text configuration. He or she can upload text and app will extract keywords and display them on screen-space.

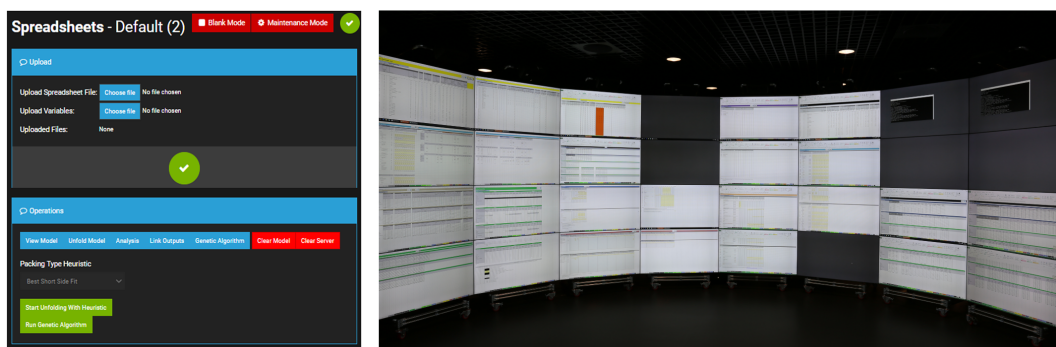


Figure 7.16: Spreadsheets system running example in GDO and control interface [55].

J. Spreadsheets: Spreadsheets project is regarding running an Microsoft Office Excel [107] on each screen in LHRDs and use GDO framework to interact with them. The purpose of distributed Excel spreadsheets are to load all sheets of a workbench at once, enabling users to see the whole solution and distributing large workbenches to multiple excels. User uploads excel file through control interface of Spreadsheets app in GDO and it loads the data across Excel instances running in the environment. It uses another background system called SheetServer to coordinate these Excel programs. Primary purpose is to unfold the spreadsheet model and load it distributedly. It creates a graph of formulas to execute on these sheets distributedly. Thus, the system benefits from distributed computation through excel. One use-case is Genetic Algorithms, which try to mimic the process of evolution. [55]

7.6 Conclusion

In conclusion, various apps and projects have been developed by colleagues, students, interns, other researchers and even companies for GDO as well as by the author of this thesis. These apps use various mechanisms provided to visualise and operate, and still work in harmony with core project. These apps demonstrate success of our architecture and software paradigm. What we have learned from these developments is mainly using CVSP enables developers to implement custom or general applications for scalable and modular data visualisations on LHRD environment thus enabling modern visual data analytics for multi-disciplinary projects which GDO has used many times for.

In this category, GDO is unprecedented as no other LHRD software was able to enable development and use of diverse general and custom data visualisations for versatile research topics and projects on LHRDs. Furthermore, GDOs interaction style greatly enhances use of these data visualisations as many users can easily engage and conduct visual analytics, which is not present in any other LHRD software. Moreover, integration of devices has many times used by developers to integrate wide range of interaction, sensing and AR devices to the system for research modularly. Finally, few developers made use of collaged and synced concepts to create coordinated data scenes using CVSP and proving these mechanisms enhance visual data analytics on LHRDs by enabling multi-view visualisations, which is novel and never been seen in the literature.

Overall, through all these capabilities and applications, we were able to provide the objective we targeted on the thesis, as many data-driven multi-disciplinary research were able to be conducted with aid of immersive and social visual data analytics.

Chapter 8

Conclusion

The following chapter contains summary of the thesis, critical analysis, contributions and future work of it.

8.1 Summary

In this thesis, we have introduced Large High Resolution Displays (LHRDs). We argued that new generation multi-disciplinary data-driven research will benefit from conducting modern web-based and interactive visual data analytics on these environments. Furthermore, unique properties of these environment can be used to extend existing visual data analytics techniques to provide what we call Immersive and Social Visual Data Analytics. We have identified problems on the existing literature regarding this. Therefore, we explored design issues for a new platform with novel software paradigm and features to enable this. Afterwards, we introduced the architecture and implementation of our platform GDO as a candidate for this task, which solves many challenges we identified for this purposes using its novel software paradigm (CVSP).

As a result, we argue that using this novel software approach, which enables modular distributed operation of system and app components on the distributed system of LHRDs and allowing these components to coordinate freely using a communication schema and thus creating many different data acquiring, processing, rendering scenarios or distributed app logics, modular integration of devices or enabling coordinated, collaged or synced apps on screenspace; and overall enables us

to overcome challenges we listed as requirements for conducting immersive and social visual data analytics on LHRDs for multi-disciplinary data-driven research.

A. Related Work: In the related work chapter, we have summarized and introduced general topics of Visual Analytics, Big Data and Data Science; and argued how their emergence have transformed research and projects to become multi-disciplinary in the age of data and require visual analytics for cross-field communication and how they need a new kind of hardware to match their requirements. We believe these hardware is LHRDs. In the literature survey, we have discussed their origins, types and properties; and listed notable examples of them in the literature. Afterwards, we have introduced in our own categorization the three generations of LHRD software in the literature. The first generation started with CaveLib built for the original CAVE. This generation was based on legacy technologies like MPI, and resorted to legacy methods for overcoming that age's problems. The second generation were focused on render distribution. Examples of this generation explored different clustering and parallel rendering methods to power LHRDs with visualisations. Finally, the third generation (current one) focused on web technologies, app orientation and rapid visualisation to improve multi-modal use of these hardware. However, none of these software targets or solves our objective.

B. Design Issues: In the third chapter, we have explored design landscape for using modern visual data analytics techniques on LHRDs. In this chapter first, we have explored collaborative use of LHRD for target objective, where we identified different roles, activities and sessions on LHRDs and potential use-cases for the target platform. Secondly, we have explored immersive display and management of data visualisations on LHRDs where we identified various display, management, interaction and other mechanisms and paradigms for the target platform. Afterwards, we explored data visualisation apps and their required properties, where we found that they have to be versatile, modular, general, distributed, scalable, interactive, customizable, extensible, web-based, standardized, reliable, castable, API-based, collaged, multi-type and inter-operable. Finally, we explored data-driven, API based and collaged data visualisation concepts. In that section, we discussed various data acquiring, distribution and processing issues on LHRDs, and also API based operation, and collaged and synced applications. Overall, we compared these challenges and requirements to existing software on the literature and summarized

them into a list of technical requirements.

C. Architecture: In the forth chapter, we have introduced software architecture of GDO, where we have a ASP.NET web project deployed to IIS web server hosting signalR web-socket libraries on the server-side and on the client-side we have web-based javascript clients that use signalR to communicate with the server and peerJS to communicate with each other. We deploy node web clients to display nodes on LHRD for creation of a distributed screen-space, where we can deploy data visualisations. We have discussed how our server software is structured, where we hold a registry behind communication hubs. Afterwards, we introduced our control interface concept and how it is structured. Later on, we introduced our application paradigm where we have server, node and control components for apps which are distributed to LHRD. Finally, we introduced our Clustered Visualisation Services Paradigm briefly. Overall, we discussed architecture of our system and how individual parts of it is structured.

D. Implementation: In the fifth chapter, we discussed our implementation by going through how each base functionality is created. First of all, we discussed how we created a distributed display area; afterwards, we discussed how we implemented management of the system and how we load and unload apps on the system. Later on, we introduced how we implemented our app paradigm, its architecture, its communication schema, its instancing mechanism, its types and finally how apps are developed in our application framework. Overall, we demonstrated implementation of all base functionalities of the system.

E. Contributions: In the sixth chapter, we introduced our novel software paradigm we created for enabling functionality that is required for conducting modern visual data analytics on LHRDs for multi-disciplianry data-driven research. Clustered Visualisation Services Paradigm (CVSP) is formed by allowing logical components of apps and system to modularly exist on distributed physical locations in LHRD such as server, cluster of display nodes, and user control clients; allowing communication between all components in the system through APIs and thus allow creation of distributed and/or on-demand functionality using meshing functionality of different components together. In this chapter, we went through each novel or critical functionality we enabled through overcoming challenges we stated in the third chapter for

the given objective as our contributions. These are Immersive and Scalable Display of Data Visualisations; Data Analytics Oriented LHRD Operating System; Modular, General, Data-Driven, Interactive, Dynamic, Collaged and Synced Data Visualisations on LHRDs; Integration of Interaction and Sensing Devices; Data Distribution and Load Balancing in Data Intensive Apps; Reliability in Visual Data Analytics on LHRDs; Interoperability, Flexibility and Remote Deployment; and Developer Support for Distributed App Development.

F. Applications: In the final content chapter, we have introduced various apps that are developed in GDO by students, colleagues, interns, companies, and their usage scenarios as a proof to succeeding in our objective.

8.2 Contributions

Our contributions to the literature is the challenges we solved using our novel software paradigm Clustered Visualisation Services Paradigm (CVSP) of GDO platform, that through it we enable conducting immersive and social visual data analytics for multi-disciplinary research projects. To reiterate, this paradigm enables modular and dynamic distributed components of apps and system on the cluster, server and control clients of LHRD and while enabling a communication schema for all to all API based communication between these distributed components, where we can enable many functionalities we deem necessary, with most significant:

Primary Contributions:

- **Immersive and Scalable Display of Data Visualisations:** General, dynamic and interactive data visualisations and other type of apps can be immersively deployed to a LHRD in any layout and in any size which includes scaling these data visualisations to tens even hundreds of display nodes.
- **Visual Data Analytics oriented Operating System for the LHRD:** A novel underlying operating system is provided for data visualisations that enables app deployment and operation, session management and distributed debugging on the cluster, where also the system can be managed from the web interfaces enabling users to operate the system from tablets/PCs.
- **Application Framework for General, Interactive and Modular Data Visualisation Apps:** General, instanced and dynamic data visualisations

that can work with different data and configurations, that are deployed modularly, which have control interfaces to configure/control/steer these visualisations which can be casted to other interfaces, can be implemented by the application framework of CVSP.

- **Data-driven and API based Data Visualisations:** Many data acquiring, processing and distribution scenarios, and also many clustering and distributed logic can be implemented on applications of GDO using CVSP.
- **Collaged and Synced Visual Data Scenes:** Collaged and synced data visualisation scenes can be created using distributed coordination between applications or using composite applications that use other applications to visualise.
- **Integration of Sensing and Interaction Research Devices:** Many different interaction and sensing devices can be integrated modularly to LHRD environment and their data and controls can be made available to other applications on the environment and control clients.

Secondary Contributions:

- **Reliable Operation:** CVSP based operation provides reliability even within a diverse, dynamic, distributed system, where applications are developed by external developers that are distributed, while using many online libraries and while many of these applications work in communication to create combined functionality.
- **Interoperability with other Web based Visualisations and Projects:** Other web-based LHRD software following control-server-client paradigm or many online visualisations can be integrated into GDO.
- **Flexibility on Display Technologies and Setup:** GDO can work with any number of screens, resolutions or setup without prior installation on its clients.
- **Remote Operation Capabilities:** GDO have capability to be deployed to remote locations in different scenarios and offer synchronization capabilities while doing so.

- **Developer friendly Distributed App Development:** Developers with different skill-sets can develop basic to advanced applications using many .NET and C# libraries on backend and web libraries on front-end, where they can implement also distributed functionality thus deciding how the visualisation acquires, processes, distributes and renders data and how the users can interact with this visualisation using CVSP architecture and software framework of GDO.

Overall, GDO through its novel architecture and features enable many functionalities we deemed necessary to enable conducting web-based, interactive, dynamic visual analytics on LHRDs for multi-disciplinary research projects. Most significantly, in order to enable immersive and social visual data analytics on LHRDs for data-driven multi-disciplinary research, many of these contributions should be provided together, hence GDO using its CVSP paradigm is the only platform that provides all these novel contributions and other critical functionalities in a single platform.

Moreover, in alignment with these academic contributions, we also contribute open source (MIT licensed) GDO Software Framework alongside many of its apps on <https://github.com/bkavuncu/GDO>[39] as practical and real world contributions that have been used by many research at Imperial College London (Contribution level of the author on the apps in the repository varies).

8.3 Conclusion and Future Work

In final conclusion, a wide range of work on LHRDs has been explored in this thesis for enabling immersive and social visual data analytics for data-driven multi-disciplinary projects on KPMG DO and other similar data science and research oriented LHRDs. This exploration resulted in many challenges and findings to enable this objective. Overall, the research problem is regarding having data visualisations that are web-based, general, modular, scalable, interactive, dynamic, ... which are deployed and operated by a system designed for this purpose and providing all these reliably, flexibly and developer friendly. Thus, we can have a vast library of visualisation apps that can be meshed together, collaged, synced or coordinated to have dynamic layouts on screen-space, connected to external or internal data sources or devices for almost any research purpose. This problem is overcome by CVSP paradigm. This paradigm in summary enables system and app components

to operate on distinct blocks on server and clients and have structured all-to-all communication schema between each other both locally and remotely. Thus, we can have all these functionality enabled for the given objective, and hence be used by many data-driven multi-disciplinary research in Data Science Institute at Imperial College London. Many apps have been developed by students as part of their masters or bachelor thesis and even used in presentations to media and important visits to the institute and used in many cross-field research projects.

In comparison to other work, even the closest platforms in functionality can not provide or not purposed for visual data analytics in these terms. This is the first platform that achieved this functionality. And the key to achieving this is partly 3rd generation LHRD paradigms we utilized and GDOs unique and novel CVSP paradigm for LHRDs. Overall, it demonstrates its software paradigm is the only or first way to achieve these vast amount of novel or critical functionality, overcoming challenges stated in the thesis. As a result, this thesis demonstrates the success of CVSP paradigm for this objective, which was the thesis argument, while becoming the first platform to enable conducting visual data analytics for data-driven multi-disciplinary research through allowing these research to visualise data by many general and custom data visualisation and supplementary apps on LHRD which are deployed by our management mechanism; enabling interaction with these visualisations during visual data analytics sessions; using session management to conduct these sessions; using collaging capability to create composite and complex and synced data scenes for social media analysis, advanced presentations, transport research and many other; using device integration capability to connect interaction and sensing devices; even conduct behavioural study by using these devices on spectators and users during sessions.

For the future of this work, we see few directions advancements can be made. First direction is integrating data processing and storage mechanisms further into GDO. Data storage, processing mechanisms that are mostly left to developers in the current version of the project can be further integrated into GDO. As a result, it can transform the system into a data science workbench, where data is stored, processed and visualised all within the system, thus it enables this infrastructure to become a data science infrastructure for institutions that deploy LHRDs. The difference to the current version is, we enable these, but not implement structure or mechanisms

to regulate these.

Another direction advancement can be made is in the area of interaction. Many different interaction capabilities can be implemented. For example, a visual programming canvas that data storage, processing and visualising elements can be connected on-demand for instant data visualisations of many versatile cases. This direction is ofcourse aligned with the previous direction.

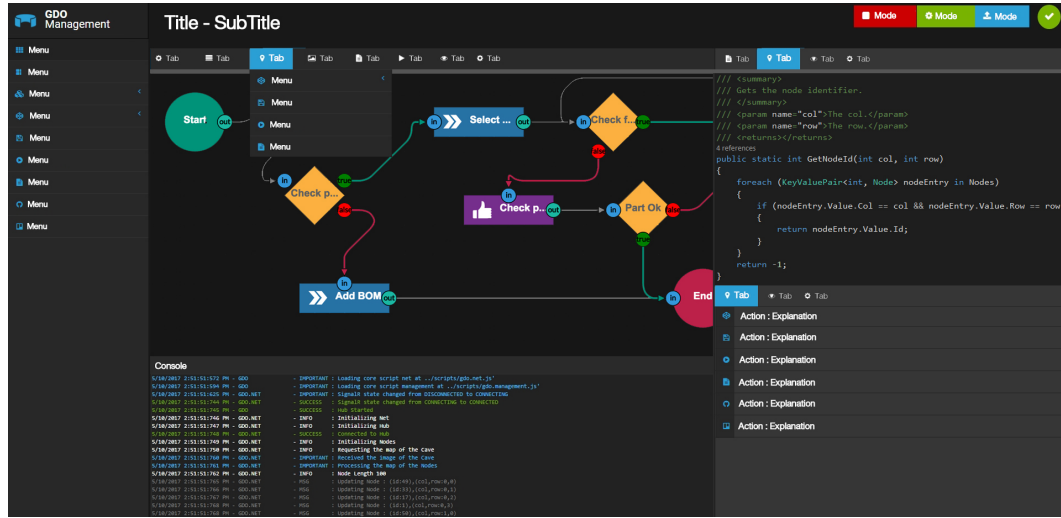


Figure 8.1: Visual Programming Concept for LHRD based Data Science

Finally, a different rendering approach could be taken in the future. For example, cloud based rendering [11]. Many LHRDs currently have a cluster of capable computers that are connected to displays, that take on the rendering role. However, in the future, this may not be feasible or necessary. LHRD can be deployed with only smart displays that have a integrated PC as part of the display. This device can host a light client that cloud can stream rendering directly. In this scenario, all the distributed rendering would be performed on a GPU cluster on the cloud and streamed to corresponding displays directly over the network.

Overall, I believe that as data becomes more important in every aspect of our lives, all research directions will include more data orientation and would need more visual data analytics. Therefore, this area is a future I am looking forward to explore and advance.

Bibliography

- [1] S. Deshpande and S. Daly, “80 . 1 : Technologies and Applications for Large Sized High Resolution Tiled Display System Technologies and Applications for Large Sized High Resolution Tiled Display System,” *Architecture*, pp. 1188–1191, 2010.
- [2] Chromium, “A Beginner’s Guide to Chromium,” 2018. [Online]. Available: <http://chromium.sourceforge.net/doc/chromium2.html> (Date last accessed 2018).
- [3] G. Zhao, “A GDO Gesture Interaction System using multiple Kinects,” MSc Thesis, Imperial College London, 2015.
- [4] D. Pape, “A hardware-independent virtual reality development system,” *IEEE Computer Graphics and Applications*, vol. 16, no. 4, pp. 44–47, jul 1996. [Online]. Available: <http://ieeexplore.ieee.org/document/511852/>
- [5] T.-R. Chen, “A More In-Depth Analysis of Twitter User and Brexit,” Bachelor Thesis, Imperial College London, 2017.
- [6] R. Kooima, A. Prudhomme, J. Schulze, D. Sandin, and T. DeFanti, “A multi-viewer tiled autostereoscopic virtual reality display,” *Proceedings of the 17th ACM Symposium on Virtual Reality Software and Technology - VRST ’10*, vol. 1, no. 212, p. 171, 2010. [Online]. Available: <http://portal.acm.org/citation.cfm?doid=1889863.1889899>
- [7] H. Wei, “A New Method of Twitter Exploration: Time Sequential Hashtag Co-occurrence Networks,” MSc Thesis, Imperial College London, 2017.
- [8] M. S. Brown and W. B. Seales, “A practical and flexible tiled display system,” *Proceedings - Pacific Conference on Computer Graphics and Applications*, vol. 2002-Janua, no. 1, pp. 194–203, 2002.
- [9] P. A. Navrátil, B. Westing, G. P. Johnson, A. Athalye, J. Carreno, and F. Rojas, “A practical guide to large tiled displays,” *Lecture Notes in Computer*

- Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 5876 LNCS, no. PART 2, pp. 970–981, 2009.
- [10] S. Molnar, D. Ellsworth, H. Fuchs, and M. Cox, “A Sorting Classification of Parallel Rendering,” *IEEE Computer Graphics and Applications*, vol. 14, no. 4, pp. 23–32, 1994.
- [11] S. Shi and C. H. Hsu, “A survey of interactive Remote rendering systems,” *ACM Computing Surveys*, vol. 47, no. 4, 2015.
- [12] Tao Ni, G. Schmidt, O. Staadt, M. Livingston, R. Ball, and R. May, “A Survey of Large High-Resolution Display Technologies, Techniques, and Applications,” in *IEEE Virtual Reality Conference (VR 2006)*. IEEE, 2006, pp. 223–236. [Online]. Available: <http://ieeexplore.ieee.org/document/1667648/>
- [13] J. Leigh, L. Renambot, T. A. DeFanti, M. Brown, E. He, N. Krishnaprasad, J. Meerasa, A. Nayak, K. Park, and R. Singh, “An experimental OptIPuter architecture for data-Intensive collaborative visualization,” *3rd Workshop on Advanced Collaborative Environments*, no. May 2014, 2003.
- [14] Microsoft, “ASP.NET,” 2018. [Online]. Available: <https://dotnet.microsoft.com/apps/aspnet> (Date last accessed 2018).
- [15] —, “ASP.NET SignalR,” 2018. [Online]. Available: <https://www.asp.net/signalr> (Date last accessed 2018).
- [16] H. Lim, “Augmenting Immersive Large High Resolution Displays,” Imperial College London, Tech. Rep., 2018.
- [17] Autofac, “Autofac,” 2018. [Online]. Available: <https://autofac.org> (Date last accessed 2018).
- [18] Babylonjs, “babylon.js,” 2018. [Online]. Available: <https://github.com/BabylonJS/Babylon.js> (Date last accessed 2018).
- [19] S. Johns, T. Angus, and A. Scheuber, “Beijing leader hails growing Imperial-China collaborations,” 2018. [Online]. Available: <https://www.imperial.ac.uk/news/186287/beijing-leader-hails-growing-imperial-china-collaborations/> (Date last accessed 2018).
- [20] N. Marz and J. Warren, *Big Data: Principles and best practices of scalable realtime data systems*. Manning, 2015.

- [21] J. Qiu, “Big-Data Detection and Classification of Algorithmic Behaviour in the Bitcoin Blockchain,” MSc Thesis, Imperial College London, 2016.
- [22] D. McGinn, “Bitcoin Observatory in the KPMG GDO,” MSc Thesis, Imperial College London, 2015.
- [23] S. Rastogi, “BitcoinViz: A Set of Visualisation Tools for Bitcoin Transactions,” BEng Final Project, Imperial College London, 2016.
- [24] K. Li, H. Chen, Y. Chen, D. W. Clark, P. Cook, S. Damianakis, G. Essl, A. Finkelstein, T. Funkhouser, T. Housel, A. Klein, Z. Liu, E. Praun, J. P. Singh, B. Shedd, J. Pal, G. Tzanetakis, and J. Zheng, “Building and using a scalable display wall system,” *IEEE Computer Graphics and Applications*, vol. 20, no. 4, pp. 29–37, 2000. [Online]. Available: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=851747>
- [25] CalVR, “CalVR Wiki,” 2018. [Online]. Available: <http://ivl.calit2.net/wiki/index.php/CalVR> (Date last accessed 2018).
- [26] J. P. Schulze, A. Prudhomme, P. Weber, and T. A. DeFanti, “CalVR: an advanced open source virtual reality software framework,” vol. 864902, no. March 2013, p. 864902, 2013. [Online]. Available: <http://proceedings.spiedigitallibrary.org/proceeding.aspx?doi=10.1117/12.2005241>
- [27] EVL, “CAVE User’s Guide,” 1997. [Online]. Available: <https://www.evl.uic.edu/pape/CAVE/prog/CAVEGuide.html> (Date last accessed 2018).
- [28] J. Leigh, A. Johnson, L. Renambot, L. Long, D. Sandin, J. Talandis, A. Febretti, and A. Nishimoto, “CAVE2 documentary,” *Virtual Reality (VR)*, 2014 *iEEE*, p. 153, 2014.
- [29] A. Febretti, A. Nishimoto, T. Thigpen, J. Talandis, L. Long, J. D. Pirtle, T. Peterka, A. Verlo, M. Brown, D. Plepys, D. Sandin, L. Renambot, A. Johnson, and J. Leigh, “CAVE2: a hybrid reality environment for immersive simulation and information analysis,” vol. 864903, no. March 2013, p. 864903, 2013. [Online]. Available: <http://proceedings.spiedigitallibrary.org/proceeding.aspx?doi=10.1117/12.2005484>
- [30] Mechdyne, “CaveLib,” 2018. [Online]. Available: <https://www.mechdyne.com/software.aspx?name=CAVELib> (Date last accessed 2018).
- [31] EVL, “CAVETM Library,” 2001. [Online]. Available: <https://www.evl.uic.edu/entry.php?id=1778> (Date last accessed 2018).

- [32] K. U. Doerr and F. Kuester, “CGLX: A scalable, high-performance visualization framework for networked display environments,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, no. 3, pp. 320–332, 2011.
- [33] Chromium, “Chromium Documentation,” 2006. [Online]. Available: <http://chromium.sourceforge.net/doc/index.html> (Date last accessed 2018).
- [34] R. Frank, S. Ahern, M. Houston, R. Ng, G. Humphreys, J. T. Klosowski, and P. D. Kirchner, “Chromium: a stream-processing framework for interactive rendering on clusters,” *ACM Transactions on Graphics*, vol. 21, no. 3, pp. 693–702, 2002.
- [35] L. Renambot, B. Jeong, R. Jagodic, A. Johnson, J. Leigh, and J. Aguilera, “Collaborative visualization using high-resolution tiled displays,” 2006. [Online]. Available: <https://evl.uic.edu/documents/sage-chi06-final.pdf>
- [36] B. Jones, *Communicating data with Tableau*. O’Reilly Media, 2014.
- [37] C. Wu, D. Birch, D. Silva, C.-h. Lee, O. Tsinalis, and Y. Guo, “Concinnity: A Generic Platform for Big Sensor Data Applications,” *IEEE Cloud Computing*, vol. 1, no. 2, pp. 42–50, jul 2014. [Online]. Available: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6924615https://docs.google.com/file/d/0ByzBYZoteKyfME5kWIInUzFwSms/edit>
- [38] Cortical.io, “Cortical.io,” 2018. [Online]. Available: <https://www.cortical.io/> (Date last accessed 2018).
- [39] A. Kavuncu, “Data Observatory Software Framework Repo,” 2015. [Online]. Available: <https://github.com/bkavuncu/GDO> (Date last accessed 2019).
- [40] J. D. Kelleher and B. Tierney, *Data science*. The MIT Press, 2018.
- [41] J. Grus, *Data Science from Scratch: First Principles with Python*. O’Reilly, 2017.
- [42] ICL, “Data Science Institute Data Observatory,” 2018. [Online]. Available: <https://www.imperial.ac.uk/data-science/data-observatory/> (Date last accessed 2018).
- [43] D. Birch, A. Grover, C. Tatli, G. Chatzaras, Q. Huang, S. Baraba, and X. Yu, “Data Spark: Understanding Cyber Attacks,” Imperial College London, Tech. Rep., 2017.
- [44] A. Kirk, *Data Visualisation: A Handbook for Data Driven Design*. SAGE, 2016.
- [45] Google Data Arts Team, “dataarts Repo,” 2018. [Online]. Available: <https://github.com/dataarts> (Date last accessed 2018).

- [46] D3js, “Data-Driven Documents,” 2018. [Online]. Available: <https://d3js.org> (Date last accessed 2018).
- [47] P. Brezovszky, “Developing a System to Visualise Clinical Scan Data on the Data Observatory for the Study of Multiple Sclerosis,” MSc Thesis, Imperial College London, 2016.
- [48] S. Lorimer, “Digital Strategies in London’s boroughs: state of play 2018,” 2018. [Online]. Available: <https://medium.com/@SmartLondon/digital-strategies-in-londons-boroughs-state-of-play-2018-1368b43aee3e> (Date last accessed 2018).
- [49] Microsoft, “DirectX.” [Online]. Available: <http://msdn.com/directx/sdk/> (Date last accessed 2015).
- [50] TACC, “DisplayCluster,” 2018. [Online]. Available: <https://www.tacc.utexas.edu/research-development/tacc-software/displaycluster> (Date last accessed 2018).
- [51] G. P. Johnson, G. D. Abram, B. Westing, P. Navrátil, and K. Gaither, “DisplayCluster: An interactive visualization environment for tiled displays,” *Proceedings - 2012 IEEE International Conference on Cluster Computing, CLUSTER 2012*, no. Figure 1, pp. 239–247, 2012.
- [52] E. Courdier, “Distributed D3,” 2015. [Online]. Available: <https://github.com/theevann/DD3> (Date last accessed 2018).
- [53] DMX, “Distributed Multihead X Project,” 2018. [Online]. Available: <http://dmx.sourceforge.net/dmx.html> (Date last accessed 2018).
- [54] D. Thomas, “Distributed Ray Marching Library for a 3d Fractals Project,” Undergrad Final Project, Imperial College London, 2016.
- [55] S. Vasu, “Distributed Spreadsheet Visualisation and Analysis on Video Walls,” MEng Thesis, Imperial College London, 2016.
- [56] C. O’Neil and R. Schutt, *Doing Data Science*, 1st ed. O’Reilly Media, 2013.
- [57] ICL, “DSI in Media,” 2018. [Online]. Available: <http://www.imperial.ac.uk/data-science/news-and-events/dsi-in-the-media/> (Date last accessed 2018).
- [58] J. Leigh, A. Johnson, L. Renambot, D. Sandin, T. DeFanti, M. Brown, B. Jeong, R. Jagodic, C. Krumbholz, D. Svistula, and Others, “Emerging from the CAVE: Collaboration in Ultra High Resolution Environments,” *Proceedings of the First International Symposium on Universal Communication, Kyoto, Japan*, vol. 6, no. 14, pp. 2007–06, 2007. [Online]. Available: <http://evlweb.eecs.uic.edu/aej/papers/LeighUniversal.pdf>

- [59] S. Eilemann, C. Marten, S. Amsellem, P. Bouchaud, P. Robert, R. Hauck, and L. P. Dulley, “Equalizer Programming and User Guide,” 2012.
- [60] S. Eilemann, M. Makhinya, and R. Pajarola, “Equalizer: A scalable parallel rendering framework,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 15, no. 3, pp. 436–452, 2009.
- [61] A. Scheuber, “Estonian President explores ‘Industry 4.0’ and next gen tech at Imperial,” 2018. [Online]. Available: <https://www.imperial.ac.uk/news/185548/estonian-president-explores-industry-40-next/> (Date last accessed 2018).
- [62] A. Gleeson, “EthCon Viewer: Visual Analytics for Classification of Ethereum Cryptocurrency Transactions,” MSc Thesis, Imperial College London, 2017.
- [63] A. Safdar, “Ethereum Transactions Visualisation,” Bachelor Thesis, Imperial College London, 2017.
- [64] ETRIKS, “eTRIKS,” 2018. [Online]. Available: <https://www.etriks.org/> (Date last accessed 2018).
- [65] D. Evanson, “EU science head meets Imperial researchers,” 2016. [Online]. Available: <http://www.imperial.ac.uk/news/174569/eu-science-head-meets-imperial-researchers/> (Date last accessed 2018).
- [66] E. Jozuka, “Europe’s Largest Data Observatory Turns Big Data into Big Images,” 2015. [Online]. Available: https://motherboard.vice.com/en_us/article/4xagbb/europes-largest-data-observatory-turns-big-data-into-big-images (Date last accessed 2018).
- [67] T. Peterka, R. L. Kooima, J. I. Girado, J. Ge, D. J. Sandin, and T. A. DeFanti, “Evolution of the Varrier (TM) autostereoscopic VR display: 2001-2007 - art. no. 649004,” *Proceedings of the Society of Photo-Optical Instrumentation Engineers (Spie)*, vol. 6490, p. 49004, 2007.
- [68] K. Reda, C. Offord, A. E. Johnson, and J. Leigh, “Expanding the Porthole: Leveraging Large, High-resolution Displays in Exploratory Visual Analysis,” *CHI '14 Extended Abstracts on Human Factors in Computing Systems*, pp. 2047–2052, 2014. [Online]. Available: <http://doi.acm.org/10.1145/2559206.2581202>
- [69] Ergoneers, “Eye Tracking Solutions,” 2018. [Online]. Available: <https://www.ergoneers.com/en/hardware/eye-tracking/> (Date last accessed 2018).
- [70] FusionCharts, “FusionCharts Suite XT,” 2018. [Online]. Available: <https://www.fusioncharts.com> (Date last accessed 2018).

- [71] Nirnimesh, P. Harish, and P. J. Narayanan, “Garuda: A scalable tiled display wall using commodity PCs,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 13, no. 5, pp. 864–877, 2007.
- [72] J. Grant, “Generative Procedural-Parametric Architectural Design,” Bachelor Thesis, Imperial College London, 2017.
- [73] B. Jeong, L. Renambot, R. Jagodic, R. Singh, J. Aguilera, A. Johnson, and J. Leigh, “Grid applications—High-performance dynamic graphics streaming for scalable adaptive graphics environment,” *Proceedings of the 2006 ACM/IEEE conference on Supercomputing - SC '06*, no. November, p. 108, 2006. [Online]. Available: <http://portal.acm.org/citation.cfm?doid=1188455.1188568>
- [74] Apache, “Hadoop,” 2018. [Online]. Available: <https://hadoop.apache.org> (Date last accessed 2018).
- [75] S. Su, V. Perry, N. Cantner, D. Kobayashi, and J. Leigh, “High-resolution interactive and collaborative data visualization framework for large-scale data analysis,” *Proceedings - 2016 International Conference on Collaboration Technologies and Systems, CTS 2016*, pp. 275–280, 2016.
- [76] C. Rooney and R. A. Ruddle, “HiReD: A High-Resolution Multi-Window Visualisation Environment for Cluster-Driven Displays,” *Proceedings of the 7th ACM SIGCHI Symposium on Engineering Interactive Computing Systems - EICS '15*, pp. 2–11, 2015. [Online]. Available: <http://dl.acm.org/citation.cfm?id=2774225.2774850>
- [77] SpaceCenter.org, “Historic Mission Control,” 2018. [Online]. Available: <https://spacecenter.org/exhibits-and-experiences/nasa-tram-tour/historic-mission-control/> (Date last accessed 2018).
- [78] Sandia National Laboratories, “IceT,” 2012. [Online]. Available: <https://icet.sandia.gov> (Date last accessed 2018).
- [79] A. Chaturvedi, “Identifying users on the Bitcoin network using machine learning and graph visualisations,” MSc Thesis, Imperial College London, 2018.
- [80] Y. Zhang, “Immersive Video Wall Presentation Tool,” MSc Thesis, Imperial College London, 2016.
- [81] G. Timmins, “Imperial 2016 conference: Collaboration remains essential to combating AMR,” 2016. [Online]. Available: <https://www.imperial.ac.uk/news/174833/arcimperial-2016-conference-collaboration-remains-essential/> (Date last accessed 2018).

- [82] R. Moore-Colyer, “Imperial university opens big data observatory to boost analytic insights,” 2015. [Online]. Available: <https://www.v3.co.uk/v3-uk/feature/2433062/-bigdatasummit-imperial-university-opens-big-data-observatory-to-boost-analytic-insights> (Date last accessed 2018).
- [83] T. Angus and A. Scheuber, “In pictures: President Xi Jinping at Imperial,” 2015. [Online]. Available: <https://www.imperial.ac.uk/news/168535/in-pictures-president-xi-jinping-imperial/> (Date last accessed 2018).
- [84] R. Bundulis and G. Arnicans, “Infiniviz — Virtual Machine Based High-Resolution Display Wall System,” *Databases and Information Systems IX: Selected Papers from the Twelfth International Baltic Conference, DB&IS 2016*, vol. 0, pp. 225–238, 2016.
- [85] P. Nilsson, “Inside Europe’s biggest data visualisation laboratory,” 2017. [Online]. Available: <https://www.ft.com/content/a8d9f1ca-154a-11e7-80f4-13e067d5072c> (Date last accessed 2018).
- [86] T. Boonnak, V. Visoottiviseth, J. Haga, D. Kobayashi, and J. Leigh, “Integration of Gesture Control with Large Display Environments Using SAGE2,” *2018 15th International Joint Conference on Computer Science and Software Engineering (JCSSE)*, pp. 1–5, 2018. [Online]. Available: <https://ieeexplore.ieee.org/document/8457323/>
- [87] S. Murray, *Interactive Data Visualization for the Web*, 1st ed. O’Reilly Media, 2013.
- [88] T. Marrinan, A. Nishimoto, J. A. Insley, S. Rizzi, A. Johnson, and M. E. Papka, “Interactive Multi-Modal Display Spaces for Visual Analysis,” *Proceedings of the 2016 ACM on Interactive Surfaces and Spaces - ISS ’16*, pp. 421–426, 2016. [Online]. Available: <http://dl.acm.org/citation.cfm?doid=2992154.2996792>
- [89] ICL, “Jaywing Collaboration,” 2018. [Online]. Available: <https://www.imperial.ac.uk/data-science/partnerships/jaywing-collaboration/> (Date last accessed 2018).
- [90] EVL, “JuxtaView,” 2004. [Online]. Available: <https://www.evl.uic.edu/cavern/optiputer/juxtaview.html> (Date last accessed 2018).
- [91] N. Krishnaprasad, V. Vishwanath, S. Venkataraman, A. Rao, L. Renambot, J. Leigh, A. Johnson, and B. Davis, “JuxtaView - a tool for interactive visualization of large imagery on scalable tiled displays,” in *2004 IEEE International Conference on Cluster Computing (IEEE*

- Cat. No.04EX935*). IEEE, 2004, pp. 411–420. [Online]. Available: <http://ieeexplore.ieee.org/document/1392640/>
- [92] KAUST, “KAUST Visualization Core Lab,” 2018. [Online]. Available: <https://kvl.kaust.edu.sa/pages/Showcase.aspx> (Date last accessed 2018).
- [93] ICL, “KPMG Data Observatory Technical Specifications,” 2018. [Online]. Available: <https://www.imperial.ac.uk/data-science/data-observatory/technical-specifications/> (Date last accessed 2018).
- [94] X. Wang, V. Vishwanath, B. Jeong, R. Jagodic, E. He, L. Renambot, A. Johnson, and J. Leigh, “LambdaBridge: A scalable architecture for future generation terabit applications,” *2006 3rd International Conference on Broadband Communications, Networks and Systems, BROADNETS 2006*, pp. 1–10, 2006.
- [95] D. Wood, M. Zaidman, L. Ruth, and M. Hausenblas, *Linked Data: Structured Data on the Web*. Manning Publications, 2014.
- [96] L. Coleman, “London Institute Uses 64 Monitors to Demonstrate How Bitcoin Works,” 2016. [Online]. Available: <https://www.ccn.com/london-institute-64-monitors-bitcoin/> (Date last accessed 2018).
- [97] A. Kothari, “Machine learning for Bitcoin address classification,” MSc Thesis, Imperial College London, 2018.
- [98] Y. Lou, H. Zhang, W. Wu, and Z. Hu, “Magic view: An optimized ultra-large scientific image viewer for SAGE tiled-display environment,” *Proceedings - IEEE 9th International Conference on e-Science, e-Science 2013*, pp. 262–269, 2013.
- [99] D. Fisher and M. Meyer, *Making data visual : a practical guide to using visualization for insight*, first edit ed. O’Reilly, 2018.
- [100] Microsoft, “Managed Extensibility Framework (MEF),” 2018. [Online]. Available: <https://docs.microsoft.com/en-us/dotnet/framework/mef/> (Date last accessed 2018).
- [101] S. Johns, “Michael Gove launches Government’s Clean Air strategy at Imperial,” 2018. [Online]. Available: <https://www.imperial.ac.uk/news/186390/michael-gove-launches-governments-clean-air/> (Date last accessed 2018).
- [102] Microsoft, “Microsoft PowerBi,” 2018. [Online]. Available: <https://powerbi.microsoft.com/en-us/> (Date last accessed 2018).

- [103] S. Hayden, D. P. Ames, D. Turner, T. Keene, and D. Andrus, “Mobile, Low-Cost, and Large-Scale Immersive Data Visualization Environment for Civil Engineering Applications,” *Journal of Computing in Civil Engineering*, vol. 29, no. 6, p. 05014011, 2015. [Online]. Available: <http://ascelibrary.org/doi/10.1061/{%}28ASCE{%}29CP.1943-5487.0000428>
- [104] NASA, “NASA HyperWall,” 2018. [Online]. Available: <https://www.nasa.gov/hecc/resources/viz{-}systems.html> (Date last accessed 2018).
- [105] Qualcomm, “NEXCAVE,” 2018. [Online]. Available: <http://qi.ucsd.edu/research-services/nexcave.php> (Date last accessed 2018).
- [106] T. Angus and J. Narcross, “Norwegian Minister visits Imperial’s innovative Data Science Institute,” 2017. [Online]. Available: <https://www.imperial.ac.uk/news/179597/norwegian-minister-visits-imperials-innovative-data/> (Date last accessed 2018).
- [107] Microsoft, “Office,” 2018. [Online]. Available: <https://www.office.com> (Date last accessed 2018).
- [108] EVL, “OmegaLib Repo,” 2018. [Online]. Available: <https://github.com/uic-evl/omegalib> (Date last accessed 2018).
- [109] A. Febretti, A. Nishimoto, V. Mateevitsi, L. Renambot, A. Johnson, and J. Leigh, “Omegalib: A multi-view application framework for hybrid reality display environments,” *Proceedings - IEEE Virtual Reality*, pp. 9–14, 2014.
- [110] Khronos, “OpenGL,” 2018. [Online]. Available: <https://www.opengl.org> (Date last accessed 2018).
- [111] OpenLayers, “OpenLayers,” 2018. [Online]. Available: <https://openlayers.org> (Date last accessed 2018).
- [112] OpenSeadragon, “OpenSeadragon,” 2018. [Online]. Available: <https://openseadragon.github.io> (Date last accessed 2018).
- [113] D. Birch, O. Tsinalis, K. H. van Dam, C.-H. Lee, D. Silva, C. Wu, M. Ghanem, and Y. Guo, “Paper: Concinnity: A Digital City Exchange Platform,” 2013. [Online]. Available: <https://docs.google.com/file/d/0ByzBYZoteKyfRkNNUnZFbWptZ2c/edit>
- [114] K. A. Perrine and D. R. Jones, “Parallel graphics and interactivity with the scaleable graphics engine,” in *Proceedings of the 2001 ACM/IEEE conference on Supercomputing (CDROM) - Supercomputing '01*, vol. 999, no. November. New York, New York, USA: ACM Press, 2001, pp. 5–5. [Online]. Available: <http://portal.acm.org/citation.cfm?doid=582034.582039>

- [115] T. van der Schaaf, M. Koutek, and H. E. Bal, “Parallel Particle Rendering: a Performance Comparison between Chromium and Aura,” *6th Eurographics Symposium on Parallel Graphics and Visualization*, no. January 2006, pp. 137–144, 2006.
- [116] ParaView, “ParaView,” 2018. [Online]. Available: <https://www.paraview.org> (Date last accessed 2018).
- [117] Peerjs, “PeerJS,” 2018. [Online]. Available: <https://peerjs.com> (Date last accessed 2018).
- [118] Z. Chunyan and K. Kwok, “Pharmaceutical tycoon’s college challenge,” 2015. [Online]. Available: <http://www.chinadaily.com.cn/a/201511/27/WS5a2b4ab2a310eefe3e99f70a.html> (Date last accessed 2018).
- [119] S. Johns, “Portuguese Science Minister discusses energy and health innovation at Imperial,” 2018. [Online]. Available: <https://www.imperial.ac.uk/news/185886/portuguese-science-minister-discusses-energy-health/> (Date last accessed 2018).
- [120] T. Rutland, “President Gast welcomes Digital Secretary to Imperial’s Data Science Institute,” 2017. [Online]. Available: <https://www.imperial.ac.uk/news/183221/president-gast-welcomes-digital-secretary-imperials/> (Date last accessed 2018).
- [121] J.-P. Jones, M. Sayers, and C. Smith, “Prince William sees impact of mental health campaign at Imperial Institute,” 2017. [Online]. Available: <https://www.imperial.ac.uk/news/182274/prince-william-sees-impact-mental-health/> (Date last accessed 2018).
- [122] Stony Brook University, “Reality Deck - Immersive Gigapixel Display,” 2018. [Online]. Available: <https://labs.cs.sunysb.edu/labs/vislab/reality-deck-facility/> (Date last accessed 2018).
- [123] UC San Diego, “Research Projects: HIPerWall,” 2018. [Online]. Available: <http://vis.ucsd.edu/mediawiki/index.php/Research{ }Projects{ }HIPerWall> (Date last accessed 2018).
- [124] T. V. D. Schaaf, “Retained mode parallel rendering for scalable tiled displays,” *Immersive Projection Technology*, 2002. [Online]. Available: <http://nguyendangbinh.org/Proceedings/ImmersiveProjectionTechnologyWorkshop/2006/papers/VR2002-283.pdf>
- [125] D. Plepys and L. Renambot, “SAGE TM (Scalable Adaptive Graphics Environment),” 2013. [Online]. Available: <http://sage.sagecommons.org> (Date last accessed 2018).

- [126] Scalable Amplified Group Environment, “SAGE2,” 2018. [Online]. Available: <http://sage2.sagecommons.org> (Date last accessed 2018).
- [127] —, “SAGE2 Repo,” 2019. [Online]. Available: <https://bitbucket.org/sage2/sage2/src/master/> (Date last accessed 2019).
- [128] L. Renambot, T. Marrinan, J. Aurisano, A. Nishimoto, V. Mateevitsi, K. Bharadwaj, L. Long, A. Johnson, M. Brown, and J. Leigh, “SAGE2: A collaboration portal for scalable resolution displays,” *Future Generation Computer Systems*, vol. 54, pp. 296–305, 2016. [Online]. Available: <http://dx.doi.org/10.1016/j.future.2015.05.014>
- [129] T. Marrinan, J. Aurisano, A. Nishimoto, K. Bharadwaj, V. Mateevitsi, L. Renambot, L. Long, A. Johnson, and J. Leigh, “SAGE2: A New Approach for Data Intensive Collaboration Using Scalable Resolution Shared Displays,” *Proceedings of the 10th IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing*, 2014. [Online]. Available: <http://eudl.eu/doi/10.4108/icst.collaboratecom.2014.257337>
- [130] Y. Kido, K. Ichikawa, S. Date, Y. Watashiba, H. Abe, H. Yamanaka, E. Kawai, H. Takemura, and S. Shimojo, “SAGE-based Tiled Display Wall enhanced with dynamic routing functionality triggered by user interaction,” *Future Generation Computer Systems*, vol. 56, pp. 303–314, 2016.
- [131] C. Papadopoulos, S. Mirhosseini, I. Gutenko, K. Petkov, A. E. Kaufman, and B. Laha, “Scalability limits of large immersive high-resolution displays,” *2015 IEEE Virtual Reality Conference, VR 2015 - Proceedings*, pp. 11–18, 2015.
- [132] J. Leigh, L. Renambot, A. Johnson, R. Jagodic, H. Hur, E. Hofer, and D. Lee, “Scalable adaptive graphics middleware for visualization streaming and collaboration in ultra resolution display environments,” *2008 Workshop on Ultrascale Visualization, UltraVis 2008*, pp. 47–54, 2008.
- [133] J. Leigh, A. Johnson, L. Renambot, T. Peterka, B. Jeong, D. J. Sandin, J. Talandis, R. Jagodic, S. Nam, H. Hur, and Y. Sun, “Scalable resolution display walls,” *Proceedings of the IEEE*, vol. 101, no. 1, pp. 115–129, 2013.
- [134] G. Paillot, “Scaling Data Visualizations For Data Observatories,” MSc Thesis, Imperial College London, 2016.
- [135] M. A. Bahameish, “Scientific Visualization in an Immersive and Collaborative Environment,” MSc Thesis, Imperial College London, 2017.
- [136] A. B. Kavuncu, “Self-Adaptive Sensor Networks for the Robust Provision of Services: Simulation and Performance Analysis,” Msc Thesis, University of Kent, 2010. [Online]. Available: <https://docs.google.com/file/d/0ByzBYZoteKyfVTgzVVNpSW5IUDA/edit>

- [137] Sigmajs, “SigmaJS,” 2018. [Online]. Available: <http://sigmajs.org> (Date last accessed 2018).
- [138] K. Moreland, B. Wylie, and C. Pavlakos, “Sort-last parallel rendering for viewing extremely large data sets on tile displays,” *Proceedings - IEEE 2001 Symposium on Parallel and Large-Data Visualization and Graphics*, pp. 85–92, 2001.
- [139] TACC, “Stallion, 328 Megapixel Tiled-Display,” 2018. [Online]. Available: <https://www.tacc.utexas.edu/vislab/stallion> (Date last accessed 2018).
- [140] R. Singh, B. Jeong, L. Renambot, A. Johnson, and J. Leigh, “TeraVision: A distributed, scalable, high resolution graphics streaming system,” *Proceedings - IEEE International Conference on Cluster Computing, ICC*, pp. 391–400, 2004.
- [141] VRAC, “The C6,” 2018. [Online]. Available: <http://www.vrac.iastate.edu/facilities/c6/> (Date last accessed 2018).
- [142] C. Cruz-Neira, D. J. Sandin, T. A. DeFanti, R. V. Kenyon, and J. C. Hart, “The CAVE: audio visual experience automatic virtual environment,” *Communications of the ACM*, vol. 35, no. 6, pp. 64–72, jun 1992. [Online]. Available: <http://portal.acm.org/citation.cfm?doid=129888.129892>
- [143] M. Weiser, “The Computer for the 21st Century,” *Scientific American*, vol. 265, no. 3, pp. 94–105, 1991. [Online]. Available: <https://www.jstor.org/stable/24938718?seq=1#page{ }scan{ }tab{ }contents>
- [144] T. A. DeFanti, D. Acevedo, R. A. Ainsworth, M. D. Brown, S. Cutchin, G. Dawe, K. U. Doerr, A. Johnson, C. Knox, R. Kooima, F. Kuester, J. Leigh, L. Long, P. Otto, V. Petrovic, K. Ponto, A. Prudhomme, R. Rao, L. Renambot, D. J. Sandin, J. P. Schulze, L. Smarr, M. Srinivasan, P. Weber, and G. Wickham, “The future of the CAVE,” *Central European Journal of Engineering*, vol. 1, no. 1, pp. 16–37, 2011.
- [145] OpenSceneGraph, “The OpenSceneGraph Project Website,” 2018. [Online]. Available: <http://www.openscenegraph.org> (Date last accessed 2018).
- [146] T. A. DeFanti, J. Leigh, L. Renambot, B. Jeong, A. Verlo, L. Long, M. Brown, D. J. Sandin, V. Vishwanath, Q. Liu, M. J. Katz, P. Papadopoulos, J. P. Keefe, G. R. Hidley, G. L. Dawe, I. Kaufman, B. Glogowski, K. U. Doerr, R. Singh, J. Girado, J. P. Schulze, F. Kuester, and L. Smarr, “The OptiPortal, a scalable visualization, storage, and computing interface device for the OptiPuter,” *Future Generation Computer Systems*, vol. 25, no. 2, pp. 114–123, 2009.

- [147] L. L. Smarr, A. A. Chien, T. DeFanti, J. Leigh, and P. M. Papadopoulos, “the Optiputer.” *Communications of the ACM*, vol. 46, no. 11, pp. 58–67, 2003. [Online]. Available: <http://10.0.4.121/948383.948410{%}5Cnhttp://flagship.luc.edu/login?url=http://search.ebscohost.com/login.aspx?direct=true{%}&db=a9h{%}&AN=11407037{%}&site=ehost-live>
- [148] C. Papadopoulos, K. Petkov, A. E. Kaufman, and K. Mueller, “The Reality Deck—an Immersive Gigapixel Display,” *IEEE Computer Graphics and Applications*, vol. 35, no. 1, pp. 33–45, jan 2015. [Online]. Available: <https://labs.cs.sunysb.edu/labs/vislab/reality-deck-home/http://ieeexplore.ieee.org/document/6866849/>
- [149] BBC, “The Secret Science Of Pop,” 2017. [Online]. Available: <https://www.bbc.co.uk/mediacentre/proginfo/2017/09/secret-science-pop> (Date last accessed 2018).
- [150] T. A. DeFanti, G. Dawe, D. J. Sandin, J. P. Schulze, P. Otto, J. Girado, F. Kuester, L. Smarr, and R. Rao, “The StarCAVE, a third-generation CAVE and virtual reality OptIPortal,” *Future Generation Computer Systems*, vol. 25, no. 2, pp. 169–178, 2009.
- [151] Brigham Young University, “The VuePod: Powerful enough for a gamer, made for an engineer,” 2018. [Online]. Available: <https://news.byu.edu/news/vuepod-powerful-enough-gamer-made-engineer> (Date last accessed 2018).
- [152] W. Zhang, “TimeV: Real- Time Big Data Interactive Visualisation on 100-Megapixel Screen with Spark,” MSc Thesis, Imperial College London, 2015.
- [153] C. Finlay, “Towards a 100 Megapixel Game Engine,” BEng Final Project, Imperial College London, 2016.
- [154] S. Ubik and Z. Trávníček, “Towards scalable streaming to SAGE2 video walls,” *International Conference on Systems, Signals, and Image Processing*, pp. 2–5, 2017.
- [155] F. Guitton, “Transmart UI,” 2018. [Online]. Available: <https://github.com/fguitton?tab=repositories> (Date last accessed 2018).
- [156] C. Snowden, “Twitter Observatory: A Set of Twitter Visualisation Tools,” MSc Thesis, Imperial College London, 2016.
- [157] P. Zikopoulos and C. Eaton, *Understanding Big Data: Analytics for Enterprise Class Hadoop and Streaming Data*, 1st ed. McGraw-Hill Osborne Media, 2011.

- [158] S. Yamaoka, “Visual analytics in scalable visualization environments,” *UC San Diego Electronic Theses and Dissertations*, 2011. [Online]. Available: <https://escholarship.org/uc/item/93h825tf>
- [159] M. Lima, *Visual Complexity: Mapping Patterns of Information*. Princeton Architectural Press, 2013.
- [160] K. Börner and D. E. Polley, *Visual Insights: A Practical Guide to Making Sense of Data*. The MIT Press, 2014.
- [161] R. S. King, *Visual Storytelling with D3: An Introduction to Data Visualization in JavaScript*. Pearson Education, 2015.
- [162] M. H. Chia, “Visualisation Framework for Large-scale Graph using Browser-based Distributed Rendering,” MSc Thesis, Imperial College London, 2015.
- [163] N. Yau, *Visualize This: The FlowingData Guide to Design, Visualization, and Statistics*. NIELSEN BOOKDATA, 2014.
- [164] B. Fry, *Visualizing data*, first edit ed. O’Reilly, 2008.
- [165] Q. Huang, “Visualizing Netflow Patterns in Large-Scale Enterprise Network,” MSc Thesis, Imperial College London, 2017.
- [166] N. Schwarz, J. Leigh, and G. Kent, “Vol-a-Tile - a Tool for Interactive Exploration of Large Volumetric Data on Scalable Tiled Displays,” *Proceedings of the conference on Visualization*, pp. 598–600, 1994.
- [167] A. Bierbaum, C. Just, P. Hartling, K. Meinert, A. Baker, and C. Cruz-Neira, “VR Juggler: a virtual platform for virtual reality application development,” *Proceedings IEEE Virtual Reality 2001*, pp. 89–96, 2001. [Online]. Available: <http://ieeexplore.ieee.org/document/913774/>
- [168] VRJuggler, “VRJuggler Repo,” 2018. [Online]. Available: <https://github.com/vrjuggler/vrjuggler> (Date last accessed 2018).
- [169] Khronos, “WebGL,” 2018. [Online]. Available: <https://www.khronos.org/webgl/> (Date last accessed 2018).
- [170] A. Hern, “Why data is the new coal,” 2016. [Online]. Available: https://www.theguardian.com/technology/2016/sep/27/data-efficiency-deep-learning?CMP=share_{_}btn_{_}link (Date last accessed 2018).
- [171] D. Silva, M. Ghanem, and Y. Guo, “WikiSensing: an on-line collaborative approach for sensor data management.” *Sensors (Basel, Switzerland)*, vol. 12, no. 10, pp. 13 295–332, jan 2012.

- [Online]. Available: <http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=3545568&tool=pmcentrez&rendertype=abstract><https://docs.google.com/file/d/0ByzBYZoteKyfTi1kZmxPbEVIMUU/edit>
- [172] G. Humphreys, M. Eldridge, I. Buck, G. Stoll, M. Everett, and P. Hanrahan, “WireGL,” in *Proceedings of the 28th annual conference on Computer graphics and interactive techniques - SIGGRAPH '01*, no. August. New York, New York, USA: ACM Press, 2001, pp. 129–140. [Online]. Available: <http://portal.acm.org/citation.cfm?doid=383259.383272>
- [173] MDN, “X-Frame-Options,” 2018. [Online]. Available: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Frame-Options> (Date last accessed 2018).