Research and application of graphic software architecture for power grid dispatching control system

CaiShen Fang1,2,a*, Zhimeng Lv1,2,b*, Yuan He1,2,c, Hui Peng1,2, Kun Huang1,2, Xiao Han1,2,Zhen Yan1,2,Hua Xu1,2

1NARI Technology Co.,Ltd., Nanjing 211100, China
2NARI Group Corporation (State Grid Electric Power Research Institute), Nanjing 211100, China
a*fangcaishen@sgepri.sgcc.com.cn, b*lvzhimeng@sgepri.sgcc.com.cn, cheyuan@sgepri.sgcc.com.cn

Abstract—Currently, the independent design and development of power grid automation system cannot meet the urgent requirements of integrated system for electric dispatching, monitoring and synthetic analysis and decision-making. In accordance with this situation, a general power grid dispatching control graphics software architecture based on the modular hierarchical theory is proposed. The architecture extracts and abstracts the common modules of different system graphics software architectures to build an integrated basic development platform. Based on this platform, an integrated graphical interactive software for power grid dispatching, power distribution network or substation supervision can be quickly developed. In this paper, the overall architecture, modular hierarchical theory, graphics general API and the key technologies for this architecture are described in detail. This architecture has been applied to the development of multiple power grid control graphics software, and the engineering application is performing well in reality. In a word, the architecture proposed in this paper satisfies the needs of power grid control integration, and effectively reducing system

1. Introduction
In recent years, with the booming development of power industry, the scale of the power grid has been expanding quickly, and the automation level of the power system has also been rapidly developed. With the development of computer technology, automatic control technology and network technology, various automatic systems such as automatic substation supervision system, automatic power grid dispatching system, and automatic power distribution system are upgrading frequently. However, at present, each automatic system is independently designed and developed, and different systems interact through files or data interfaces. This design and development mode cannot meet the urgent requirements of integrated system for electric dispatching, monitoring and synthetic analysis and decision-masking. In addition, this design and development contrary to the development direction for integration, standardization and intelligence of the power system[1,2,3].

There are many commonalities in automatic systems such as automatic substation supervision system, automatic power grid dispatching system, and automatic power distribution system. Each system is basically composed of modules such as front-end systems, SCADA systems, data services, message management, graphics systems, database systems, and advanced applications. These common modules are abstracted and reconstructed to build an integrated power automation platform. And based on this platform, automatic subsystems in various fields can be developed. This design philosophy is
according with the direction of future power automatic system development. As the key technology of this power automatic development platform, the importance of graphics software is self-evident.

At present, as a module of each automatic system, graphics software is driven by application requirements to meet functional demands. It is tightly bound to service logic, with high coupling factor and flattening architecture which cannot satisfy software quality requirements such as reliability, extensibility, customization, and maintainability. At the same time, the common functions of each system are repeated development and cause poor software reusability.

Therefore, it is necessary to research a kind of general power graphics software architecture. In this paper, by abstracting the commonalities of the graphics modules of various automatic systems, a power graphics platform software is designed based on a component-based loosely coupled architecture. Based on this architecture, automation graphics software such as EMS, DMS, and automatic substation supervision system can be quickly developed.

2. The overall architecture of the graphics software of the power grid dispatching control system

2.1. Overview
Up to now the graphics modules of each power automatic system have perfective functions. Generally, a three-layer architecture design of service layer, logic layer and interface layer is adopted for graphics module. The service layer includes functions such as drawing tools, integrated graphics elements, and graphics reading and writing. The logic layer is responsible for data interaction analysis and other data processing work. The interface layer is responsible for graphics browsing, editing and dialog management. This hierarchical structure seems reasonable and basically consistent with the MVC development model. However, the graphics module is the entrance for the power automatic system, the dispatchers, maintenance personnel, and monitoring personnel need interact with system by graphics module all the time, reliability and scalability directly affect the use and maintenance of the system. In consideration of the fact that the graphics module is the entrance of the automation system, various of new application requirements performed as interface requirements are proposed for different subsystem, so high requirements on the scalability of the graphics system is needed by with the demand of reliability. However, in the original three-layer architecture, the new application requirements would transfer from the interface layer to the logic layer and the service layer, which cause all graphic module changed with new application requirements. The addition of extension functions leads to changes in the underlying code indicate that the architecture is relatively flat and does not leave enough level space for extensions and result at reducing the stability and maintainability of the system.

The abstraction of graphic platforms for automatic substation supervision system, automatic power grid dispatching system, and automatic power distribution system requires that the graphic architecture has a multi-layer architecture, abstracting cross-functional requirements into specific modules, and providing development interfaces to the upper level. The underlying framework need to no longer design for power system business needs, but is designed in a general graphics platform, so that the underlying framework has the flexibility to develop different application graphics systems.

2.2. Modular layered architecture
The modular layered architecture adopts a hierarchical organization pattern. Each layer provides services for the upper layer and uses the functions provided by the next layer. Each layer is composed of several different sub-layers and libraries. This pattern allows a complex problem to be gradually disassemble in layers and libraries. Each layer in the hierarchical pattern only affects two layers at most. As long as the same interface is provided to adjacent layers, each layer is allowed to be implemented in different ways, and can fully support software reuse.

Each automatic system only needs to care about those functional layers related to specific functions. Each layer provides services to the higher layers and obtains services from the lower layers. When
constructing various application graphics software, it is only necessary to select the required resources from the underlying platform according to the related functional requirements of the graphics software, and then develop secondary development based on the general resources. The complex application needs are decomposed by function to make the overall design clearer, support the step-by-step abstraction of the power graphic module design, and result in better scalability and reusability.

The power graphics platform software architecture proposed in this paper divides the entire graphics platform into six specific levels, each level completes its work independently, the high level depends on the low level, and the low level does not rely on the high level. The overall architecture is shown in Figure 1. A detailed analysis of the architecture is followed.

2.2.1. System abstraction layer
System abstraction layer provides encapsulation of system-related APIs, and a common object-oriented API to access system resources in a platform-independent manner. Standard Template Library (STL) is used as a generic container class with providing containers, iterators, and algorithms And so on.

The operating system library (OSL) encapsulates the operating system functions used to access and utilize system resources (such as files, memory, sockets, pipes, etc.). OSL is a very thin layer with an object-oriented API. The difference from the upper layer is that the object-oriented API here implemented in C language. The advantage of C language API is to allow this layer to be transplanted to different platforms with different implementation languages.

Qt is used as the abstract realization of the interface library of each operating system in the system abstraction layer. Qt is a cross-platform programming development package with open-source code and has unique advantages in interface development. First of all, it can solve the differences between different platforms and unifies the function and style; secondly, it provides a set of interfaces drawing tools, making the interface very convenient; third, Qt is open source.

2.2.2. Infrastructure layer
This layer builds a platform-independent environment for applications and provide components and services upward. There are many aspects of the API required by the object-oriented platform in this layer, including script analysis, tools, message management, window systems, and data services.
The script parsing library provides the function of accessing the graphics module API through the scripting language Python. Allowing scripting languages to access graphics APIs brings many advantages, such as cross-platform, rapid development, script-based applications, scalability, etc.

The window system provides related functional modules to meet the needs of different automation systems for window generation, layout, interaction and management.

Build a message management module to meet flexible message management between different layers and different libraries, provide message object binding function, realize one-to-one, one-to-many, many-to-many and other different message distribution and response mechanisms. Besides, this layer provides message filtering function and builds an event-based graphical operating environment.

The data service uses the push data method to obtain data, provides a standard and unified data interface, refreshes the screen when there is data update; provides a partial refresh function in rendering process. This mechanism improves the screen response speed, and ensures the screen operate smoothly.

2.2.3. Architecture layer
In order to achieve reuse between different applications and provide the architecture and environment required for each implementation and all shared functions (such as public dialogs, file access, configuration management, etc.), the architecture layer is the core module of the system's scalability and necessary to provide multi-dimensional openness.

The application framework library provides an environment for all implementations. The common functions are implemented in this layer. In terms of program structure, each application must implement a widget and several views.

The shared function library provides sharable basic modules, encapsulates the functions of the system abstraction layer and the infrastructure layer, providing flexible module management and interaction mechanisms to meet the needs of different rendering and interaction forms of the realization layer.

2.2.4. Implementation layer
The realization layer realizes the basic functions of two-dimensional graphics through Qt basic rendering, builds basic tools for graphics drawing, encapsulates basic rendering tools such as points, lines, and circles, abstracts basic metafile of the power system, supports local coordinate systems, and has automatic redrawing and collision detection functions to realize the object management data structure based on directed acyclic graph. Basic visualization is realized through OpenGL rendering, virtual reality is realized through 3D rendering engine, and integrates advanced technologies such as Html5, Javascript, WebGL to improve better way of presentation. The GIS module includes Qt-based GIS client and 3D GIS.

2.2.5. Application layer
Based on the basic functions provided by the system abstraction layer, infrastructure layer, architecture layer, and implementation layer, the application layer can select the underlying modules required for application development according to the needs of each automation system for the graphics system, freely combine them, and build an exclusive development framework to quickly meet the needs of application functions.

2.2.6. Scene layer
The system provides display functions for different scenes, including production system workstations, large screens and handheld devices and other suitable display of different equipment.

2.3. Graphics system general API
The general API interface of the graphics system adopts widely openness, from infrastructure layer to the upper layer all open completely, and each layer implements the inheritance system based on the
core reference class. The API opening based on meta-object programming realizes open solutions at different levels and different granularities. This interface structure is very important for determining the extent to which applications need to be re-implemented during development.

The API interface has the following characteristics. Firstly, A fully defined component and environment interface can be easily combined to meet the needs of a specific object; Secondly, general functions and functions required by a specific version are distinguished to achieve version independence; Thirdly, developers can start from A minimal set of interfaces began to gradually increase applications to achieve scalability; Finally, adopt common interface definitions to achieve reusability.

The API is designed from the perspective of application and component developers, provides almost all graphical platform framework component programming interfaces, and can integrate new components.

3. Architecture key technology

3.1. Build the infrastructure

3.1.1. Window system
As power graphics need to display various of information and the total amount of information is ever-increasing, how to display information more comprehensively and efficiently becomes particularly important. The multi-window and multi-view graphical display mode is a more advanced window system display mode which can simultaneously display multi-category information through different information display modules and display themes in an overall interface.

Our system adopts a multi-window and multi-view display mode. There could be multiple windows in the screen, and each window can have multiple views. The graphics system is built on the window with the window provides a viewport and a container for rendering. Each window independently completes drawing and interactive operations. At the same time, the window management system provides a complete interactive mechanism. This system provides free interaction between different windows and different views, allows global information and local information to be linked, and supports the traceability of historical information in the graphics system and the drilling of multi-dimensional data.

3.1.2. Scripted
Scripting is a technology that allows secondary developers of power graphics to use a scripting language to access the underlying API of graphics. Through Qt's signal and slot (SIGNAL/SLOT) mechanism, different graphic objects can send and receive messages to each other; and by writing code to intercept Qt events, scripts can receive events and complete information transfer. Generally speaking, scripting is developed based on the adapter design pattern, and provides a mapping relationship from one language to another. Schematic diagram of scripting technology is shown in Figure 2.

![Fig. 2. Schematic diagram of scripting](image)

The graphics API and Python packaging module with the open graphics architecture are compiled to generate the associated dynamic library, and the associated dynamic library is imported into the Python program, then the packaged graphics API can be called in the script.

Using scripting language to access API native code brings many advantages.
A. Cross-platform, the scripting language is interpreted and executed, and the execution is pure text source code, independent of the platform, so it can run on various operating system platforms. And the modified script program does not need to be compiled, and can be run after editing directly.

B. Rapid development. When development of C++ language program, the newly developed code needs to be recompiled and linked to test and run. This is a huge time cost for development work, especially for large-scale projects. However, script language development naturally avoids this problem.

C. High performance. Modules with higher performance requirements in the graphics platform are written in C++ language, but logic programs based on core modules are developed in script language. In the MVC (model, view, control) design mode, the view is written in C++, the model is written in C++, and the control module is written in script language. Because the model and view parts of the program change little with the needs of users, and the control part changes more frequently, it is suited to use a script language in control module. The script calls the efficient APIs of the model and view modules without affecting the performance of the original system. In the face of the changing requirements of the business logic of the power automation system, the script language can be more flexible and there is no need to compile and link during the development process, which greatly improves the development efficiency.

D. Increase the scalability of the system. Support scripting from the overall framework of the design, open the core underlying functions of the system, scripts plugins can be added to develop new function in the graphics system, and support different application layers to realize the development of business modules.

3.2. Implement layer hybrid rendering

The realization layer supports different rendering methods in the rendering toolkit through multiple scenes and multiple views. The scene manager provides the context environment required for rendering and the model module provides a plugin management mechanism. It will support a variety of different formats of model files through plugin development, including G files, SVG, OSG, XML and other formats. The scene manager submits the rendered result to the view for display by traverser. Realize the layered hybrid rendering architecture is shown in Figure 3.

![Fig. 3. Schematic diagram of rendering architecture](image)

3.2.1. Qt basic rendering

There are many expression requirements for two-dimensional graphics in power graphics, and two-dimensional graphics are the main operation interface in the system. Qt has an excellent cross-platform two-dimensional graphics engine, which is based on QPainter and can draw any geometric shapes, images and text. It can basically meet the rendering work of simple pictures with a small amount of data, but it is weak in rendering graphic under large data in the scenes.

Our graphics framework uses a graphics/view (GraphicsView) architecture to display a mass of data, user-interactive, and arbitrary-shaped items on the two-dimensional drawing board. The model is managed through the scene manager named GraphicsScene, and different GraphicsItems are combined in the model to form a rendering scene, providing an item-based graphical operation interface, which
can perfectly meet the needs of item-based high-performance drawing and interaction. Large and complex topological power flow diagrams, geographic power flow diagrams, GIS maps, etc. in power graphics can all be implemented using this rendering framework.

3.2.2. 3D rendering engine
Three-dimensional rendering has effects that are not available in two-dimensional rendering. Our framework provides basic OpenGL rendering to meet the needs of graphics for basic three-dimensional functions. The focus is on the development of visualization applications based on the three-dimensional rendering engine. The 3D rendering engine not only has the cross-platform features of OpenGL and high rendering performance, but also provides 3D data file format analysis, text and science font support, level of detail control, multi-threaded data paging processing and other functions, which can make full use of visualization technology to quickly and effectively process large-scale power graphic data. Base on this engine, it can develop geographic flow diagrams of virtual reality technology and realize electrical power flow diagrams of the entire network, etc.

The basic functions of GIS are drawn by GraphicsView. However, the integration of 2D and 3D is the technological development trend of GIS. The 3D rendering engine module can provide a complete 3D interface for the 3D GIS development module in this system.

3.2.3. Multi-platform display
The power production system's requirements for real-time, operational fluency, resource acquisition channels, and custom widgets, etc., which make the power graphics based on the C/S architecture meet business needs well. However, with the commissioning of various power software systems, more and more systems adopt different architectures. The B/S architecture highlights its advantages with the popularization of cloud computing and various intelligent terminals has accelerated the development of network-driven server-heavy-client architecture.

Multi-platform hybrid display has all the advantages of desktop applications (C/S architecture), and can integrate and display B/S architecture information. This architecture is based on Webkit to develop a hybrid rendering module, which supports the display of web pages such as HTML5, JavaScript, WebGL, Flex, etc. Besides, it is embedded in desktop applications in the form of graphic elements, so that desktop applications can be easily embedded in web pages. Multi-window, multi-screen, and multi-application integrated display of power graphics are the better solutions with the development of technology.

3.3. Multiple application scenarios support
3.3.1. Power graphics products
Based on the graphics system API provided by the modular hierarchical graphics software architecture, each application system dynamically obtains the resources needed for development from the underlying framework, and quickly builds basic graphics editing and browsing tools through application programming, script programming and configuration. Each application system can customize different model files, develop analysis plugins based on the model module plugin mechanism, and realize flexible application of different application system models.

According to the basic functions provided by the platform, each graphics application system is developed to realize the business requirements of the system. The core module calls the underlying implementation provided by the architecture, and the business logic part can be scripted to achieve fast, high iteration, and low coupling business integration.

3.3.2. Multi-display mode support
With the popularization of various terminals, the computer desktop is no longer the only channel for obtaining information. Terminals such as large screens, mobile phones, and tablet computers can easily and quickly obtain information. The framework forms a unified graphics system API through
the abstraction of the window system, low-level rendering, message management and other modules. The API can adapt to the operating environment of different platforms, supports the large screen’s requirements for high rendering pixels for large scenes, and supports various hand-held devices have requirements for performance and small screens, meanwhile supporting interactive linkage between different terminals. The basic graphics function, visualization function and GIS of the realization layer all support cross-platform, and the corresponding packaging program can be developed on different terminals with perfect support in this framework.

4. Application of graphics software architecture
At present, a number of graphics software projects have been developed based on the graphics software architecture, and the projects have been successfully operated in different application.

For example (e.g.):
A. Large-screen display project for advanced applications of North China Power Grid
The project combines the advanced application functions of the power grid and the development of a large-screen display module based on the scene layer of the graphics software architecture, which meets the requirements for dispatching large-screen large scenes, high rendering pixels, and flexible operation.
B. Comprehensive information display project on large screens in Beijing
The project is mainly based on the development of mixed programming modules at the realization layer of the graphics software architecture. It uses HTML5, JavaScript and WebGL technology to display comprehensive information about the operation of the power grid. The project structure is clear, and the modules are loosely coupled and dependent, which satisfies the customized functions of the comprehensive information display project with large number of demands.
C. Modulation integration project
Modulation integration project is developed based on graphics software architecture, using the same set of graphics code, and at the same time meeting the needs of dispatching and substation related business, through the window system and scripting, the code reusability and scalability are well realized.

5. Conclusion
This paper designs a general power grid dispatching control graphics software architecture based on modularization and layering theory, which solves the problems of self-contained systems and system isolation caused by the independent design and development of different automation systems, and at the same time satisfies the integration, standardization and intelligence of power system development demand. Based on this architecture, different power system graphics applications can be quickly developed. And the infrastructure layer and the architecture layer will precipitate stable interfaces and core module dynamic libraries. The core rendering framework of the implementation layer will gradually become stable, and the application layer and scene layer will be flexible. The performance can well meet the needs of different applications and different scenarios. The entire framework will have better stability and scalability with the continuous development of the application system.

6. Acknowledgments
This work was supported by the Science and Technology Project of the State Grid Corporation of China: Research and Application of Key Technologies of Lightweight Human-computer Interaction for Microservice Architecture(5108-202040022A-0-0-00).

References
[1] Wang Jifeng. A preliminary investigation on development of operation smart system for china southern power grid[J].Southern Power System Technology, 2012,6(2):1-5.
[2] Zhang Wenliang, Liu Zhuangzhi, Wang Mingjun, et al. Research status and development trend of smart grid)[J]. Power System Technology, 2009, 33(13): 1-11.
[3] LUO Jie, CHEN Chipei, LIU Jingfang. Research and design of the integrated PAS / DMS / DMIS / TMR system[J]. Relay, 2005, 33(1): 57-59, 72.

[4] REN Jianfeng, DING Yawei, FU Lei, et al. An improved strategy for out-of-step separation based on phase angle principle for 1000 kV ultra-high voltage AC power grids[J]. Automation of Electric Power Systems, 2011, 35(10): 104-107.

[5] NGUYEN T T, LI X J. A fast and accurate method for estimating power systems phasors using DFT with interpolation[C]// Proceedings of 2006 IEEE Power Engineering Society General Meeting, June 18-22, 2006, Montreal, Canada: 8p.

[6] IEEE Std C37.118-2005 IEEE standard for synchrophasors for power systems[S]. New York, NY, USA: The Institute of Electrical and Electronics Engineers, 2006.