ITC 2/52
Information Technology and Control
Vol. 52 / No. 2 / 2023
pp. 471-486
DOI 10.5755/j01.itc.52.2.32029

Mod2Panel: A Design Framework for Model-Based Automated Generation of Interactive Panels

Received 2022/08/08 | Accepted after revision 2023/05/08

HOW TO CITE: Chen, G., Wen, C. (2023). Mod2Panel: A Design Framework for Model-Based Auto-mated Generation of Interactive Panels. *Information Technology and Control*, 52(2), 471-486. https://doi.org/10.5755/j01.itc.52.2.32029

Mod2Panel: A Design Framework for Model-Based Automated Generation of Interactive Panels

Gang Chen
Institute of Computer Application, CAEP, Mianyang 621900, China; e-mails: zjucg@zju.edu.cn

Chunmei Wen
Department of Mathematical and Natural Science, Tübingen University, Tübingen72076, Germany; e-mails: boawenmay@gmail.com

Corresponding author: zjucg@zju.edu.cn

A panel is an event-centric starting point for implementing a model-based interactive system. The design and construction of an interactive panel involve deciding what information to display, how to display it, and ways to implement the design intent to produce an interactive panel. Traditionally, the design of panels has been implicit in the deployed applications, rather than explicitly considered as digital artifacts. In addition, users must realize this implicit design manually by coding or configuring it on programming platforms, resulting in hampered and time-consuming control and analysis. Besides, current tools do not have a unified generation mechanism, which makes it difficult for cooperation. In this paper, we propose a unified framework Mod2Panel, which enables users to draw their interactive panel designs as models and can automatically generate interactive panels from these models. The models are described in a modeling language that involves structures, behaviors, layout, and parameters. Mod2Panel also provides a GUI-assisted editor for customization to fine-tune the generated panels and update their associated models. With the capabilities of Mod2Panel, users can unify prototyping, generation and deployment in this framework for purposes of operation and control. We evaluate its effectiveness and efficiency in applied case studies on complex control systems and system modeling, in which Mod2Panel successfully generates interactive panels to support control monitoring and system-level analysis. The operations in the generated panel systems demonstrate the effectiveness of Mod2Panel for real-world scenarios.

KEYWORDS: Mod2Panel, Model, Design Intent, Auto-Generation, Control.

1. Introduction

Over the past decades, interactive software has grown in importance as an ingredient of the real world. Comprehensive information visualization interfaces for models and data can aid users in creating reliable and responsible decision-making systems [3, 12]. Regardless of where they are or what they are doing, people are accustomed to using a range of interactive software components that facilitate the communication between humans and applications, such as communication applications, data analysis dashboards, vehicle navigation systems, or other devices with interfaces [10].

Human Computer Interaction (HCI) is widely used in real-world environments and typically combines several different academic fields [26], such as design, computing, psychology, and others. For modern systems, continuous and rapid adaption to users' preferences is critical for a system's success in the highly competitive market environment [20, 23]. For example, in current systems for engineering practice, to assure the proper design of complex physical systems, it is vital to develop a model-based system to eliminate implicit or informal specified inter-dependencies in documents. The ultimate goal is to offer tools for various types of system-level analysis and requirement validation. By displaying some features, such as traceability, incidents, and trends, the system instantly offers a useful signal for decision-making when people engage with it through interfaces.

An interface encapsulates a set of task-specific design intent and visual elements for particular users. The design intent behind the visualization techniques i.e., what information to convey or how to visualize it, is typically implicitly inside the mind of users as knowledge or habits. Although we find that current interfaces can work well for particular tasks and platforms, the design intent cannot be fully extracted as digital artifacts [7, 30]. That indicates that most interfaces are not intended for reusing or sharing across functional platforms. Meanwhile, because interfaces implicitly suggest different design intent, it is difficult to control their versions that are typically necessary for task-specific scenarios.

Traditionally, developing the interface of software can be roughly separated into two linked stages: design and implementation [8]. Designing an interface allows designers to define their design intent and personal knowledge such as appropriate visual layouts and effects, information architecture, and interaction style, while implementing the interface focuses on coding and testing inside a programming framework. Therefore, this developing pattern can be a laborious, time-consuming, and error-prone task in particular when users lack developer expertise.

Since they have access to their information systems, users frequently utilize interactive programming platforms like Configuration Software [33] for exploring and modeling data. Users may implement their design intent and ideas in those contexts in fact without worrying about the code specifics. However, they only offer a small collection of widgets and insufficient means of converting users' designs to meaningful interfaces. For example, for system modeling, these environments can assist with tabular data analysis but lack representation techniques for system-level analysis and traceability. Moreover, another separate challenge is sharing the design intent of users and exchanging the output of the platforms, and then version-control it.

To address these issues, it is necessary to represent users' design intent explicitly. Theoretically, the design intent may be turned into a model that has all the details on what to present and how to display it. Users can edit and share the model according to their preferences and do version control on the models. As a result, designers typically engage in an iterative process of summarizing their design intent into a model and subsequently translating it into particular interfaces.

A model can be regarded as the skeleton of design intent that only lacks visual effects. The grammar and syntactic architecture of the skeleton serve as the foundation for an automated generation mechanism. Compared to manually creating them, automatically creating interfaces can result in a substantial reduction in effort. Typically, we use text to implement models. During the early prototype stage, collaboration and communication are made easier by a text-represented model. Designers may concretize, illustrate, and store their designs in a document by using suitable syntax. Some already-in-use tools [1, 14, 22, 34] demonstrate their capacity to transform models into illustrations of interface design. However, they have not been widely applied to more complex applications, such as large-scale control systems with frequent interactions. This is due to the constraints of their design. This makes it highly challenging to accurately describe some key aspects of building interactive interfaces, such as geometric layout, interactions, parameters, and some other customization attributes, and extremely difficult to accommodate more generic circumstances.

In this paper, we introduce Mod2Panel: a design framework that makes it simple for users to condense their design needs into models, which is central to the framework. The reason why we refer to the term 'Panel' here rather than 'Interface' is to emphasize that the auto-generated interfaces follow a low coupling and high cohesion design practice. The proposed models are comprised of users' design intent. By merging a gallery-based widget pool, an automated generating mechanism is established to generate interactive panels. In order to verify the correctness and matching of the models, Mod2Panel also offers a graphical editor so that users can edit the models by configuration and customization according to their needs. Every change will be persisted, which forms a continuous feedback loop. Mod2Panel is highly configurable, enabling users to customize what information to present, how to display it when to trigger interactions, where to obtain feedback signals from users, and who will receive data. This framework enables rapid development of prototypes and deployment of task-specific, model-driven interactive panels to validate and evaluate users' design intent. Additionally, because of a high-level, model-based design of interaction concepts, Mod2Panel is also advantageous to users for focusing more on representation and visualization, as it supports individualizations concerning distinct scenarios and groups of users by customizing models using specialization and configuration.

We implement a Java Swing based version of Mod2Panel and use it to automatically generate thorough panels for different applications. We adopt this tool to train our system modeling group and control group that consists of more than 10 practitioners and conduct a series of user studies to experimentally evaluate Mod2Panel's efficiency and effectiveness. We use Mod2Panel for system modeling because of their requirements for multi-level system analysis, which are subject to change quickly. For the control group, which has a high number of devices to monitor and control, auto-generated tools like Mod2Panel are more expected. Throughout case studies, we systematically select, gather, and human-aided compare more than 10 integrated panels. Our evaluation shows that Mod2Panel is capable of offering different types of visual elements and spatially layout them in a precise and reliable manner. After interviewing the participants, we can state that Mod2Panel can capture their distinct design intent and satisfy their requirements for system-level analysis and control.

The main contribution of this paper is as follows:

- Model-driven and task-specific visualizations derived from and updated with the grammar architecture of a proposed model language.
- An automatic generation mechanism is proposed based on the model syntax where various visual elements are combined and integrated to generate relevant panels.
- A graphical editor is provided to enable a continuous feedback loop to enhance the user's design intent and persist it into models.
- We demonstrate our tool's robust visual understanding across large-scale experiments and provide initial evidence of its usefulness by case studies.

2. Related Work

In this section, we detail some key literatures related to structured modeling languages, document data transfer and automatic generation mechanism, from which we are inspired to build our tool Mod2Panel for interactive interface construction.

2.1. Structured Modeling Language

Patterns and structured modeling languages provide hierarchy components and grammar for abstracting and modeling the visualization behaviors. Modeling languages have an increasing impact on HCI. It specifies how visualizations are constructed and how they are related to data. Initially, these languages are adopted by software engineers and system architects for the reuse of generic visual solutions to facilitate the transition from individual coding to models [2, 11]. Around the beginning of the 21st century, modeling languages have gradually entered the fields of HCI, user experience [31], and organizational workflows [14]. Modeling languages adopt a series of formal representation methods to define important elements of a visual solution to provide a common template for implementing the solution.

To improve the quality and usability of visual design and interactions, many pattern catalogs and modeling languages have been developed [35]. Some UI pattern catalogs are discussed and compared by Deng [9]. The famous visualization library D3 [6] enables users to arrange their design intent and bind data into arbitrary document elements. With D3, designers can use a dynamic transform to both create and edit content, which improves the expressiveness and facilitates integration with other development tools. A structured approach is suggested by Märtin [22] both for structuring the hierarchical HCI pattern language and optimizing the selection of the needed patterns during development. Another approach from the perspective of software engineering using abstract models and atomic visual building blocks is designed by Seiger et al. [28]. This framework follows a basic class structure consisting of a variety of components which are usually software elements with predefined built-in interfaces and behaviors and all of which are interconnected with each other. Unfortunately, most of the aforementioned languages are not well organized to support pattern selection and other domain-specific design preferences.

Compared to the document-based and layer-based visualization schema of D3, Vega is a higher-level modeling language [27]. With the help of this tool, users can concentrate more on the visual effects and interaction design for data visualization. It provides a chart pool and each entity within it implements reusable and sharable interfaces that can automatically produce customizable visual effects. For modeling language, Vega has a JSON-based (JavaScript Object Notation) language as well and it generates low-level visual specifications that abstract more than data models, graphical marks, visual encodings, and other detailed specifications [20]. However, Vega places more emphasis on chart diagrams or data visualization. Although it can abstract users' design intent and interact with each other in some protocols, it lacks capabilities like richer visual elements or additional types of interactions, making it unsuitable for comprehensive scenarios.

2.2. Document Data and Transfer

For data visualization, a good solution for data documentation and transmission must deal with the construction and deletion of document entities, not merely the style of existing nodes [6]. Most existing document manipulators, such as JavaScript libraries, CSS, or JQuery, only have the ability to identify a set of elements using simple predicates rather than adding or removing target elements according to the user's design intent. That implies that JavaScript or JQuery cannot move design intent from modeling documents to visualizations or from visualizations to documents. Therefore, they are not suitable for complex or dynamic visualization tasks involving various transitions.

The eXtensible Markup Language (XML) is a remarkable standard for document data representation and exchange through WWW [5]. Extensible Style sheet Language Transformation (XSLT) [4] is another powerful tool for document transformation. Users can choose, remove, rearrange, or add more information to any XML document using XSLT to transform it into another document. Multiple DTDs in XML provide approaches for accessing, representing, or editing the document that is encoded according to the CES data architecture [16]. However, XSLT is only useful for simple transformation. It lacks mechanisms to capture design intent as well as tools for high-level visual abstractions.

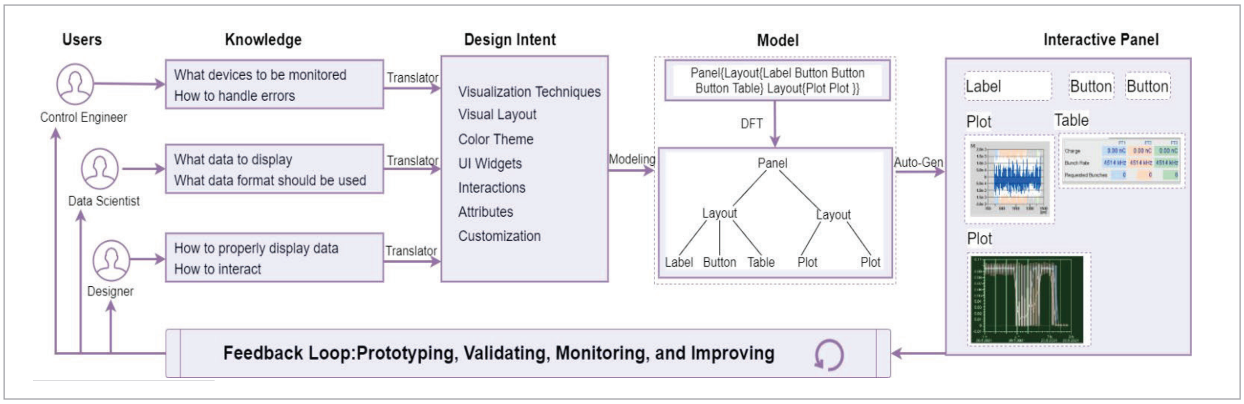
2.3. Automated Interface Generation

Building high-quality applications in a respectable amount of time are the ultimate goal of software development. Achieving this goal has gotten increasingly difficult as project scale and complexity have grown significantly. Because of this, there is a surging demand for techniques for producing more high-quality products within less delivery time. During the initial design phase, aforementioned visual modeling languages can help shorten development time in given scenarios by providing modeling and model-checking mechanisms. In the following implementation phase, automatic code generation can be applied to produce executable software from high-level design models, which will further reduce delivery time and enhance the benefits of modeling and validating [15].

Most existing code generation approaches typically already address design intent modeling and code

Figure 1

An unrolled view of prototyping, validating and feedback loop



generation for object-oriented analysis. For example, IBM Rational Software Architect [19], MagicDraw UML [24] as well as open sourced tools ArgoUML [21], in which models are represented in UML diagrams.

Some other existing tools also offer mechanisms that can generate interfaces based on modeling languages. Generally, the approaches proposed by these tools fall into two categories, namely data automation and visualization automation. Topalian Rivas et al. [32] proposed a module concept in the manufacturing field to process data before it is entered into a visualization template to fulfill interface automation. Palpanas et al. [25] defined an XML-based meta-model to define the user's design intent in the domain of business performance. Information on user roles, data metrics, interface templates, and element organization can all be summarized using this paradigm. Meanwhile, a comparable tool for generating code is also implemented in a given programming environment and can generate automatically executable interfaces. On JavaScript platform, Kintz [18] provided an automatic mechanism for generating interfaces from predefined model documents to regulate and track marketing behaviors. The generic visual information present in these models consists of data, logic, visual entities, and interaction behaviors. Vázquez-Ingelmo et al. [34] also employed an XML-based standard to model user interaction patterns to improve the capabilities of model languages in interaction.

The above tools generally suffer from two clear drawbacks: their visualization techniques mainly focus on chart-level interfaces and the interior interaction is also associated with various charts. The arrangement of various charts cannot meet the demand of complex design intent summarization. Meanwhile, interactive graphical editing tools necessary to improve user customization through visual configuration are absent from these solutions, which instead only offer model-based customization.

3. Problem Formulation and Usage Scenarios

We formulate the design-to-model-to-panel generation as a machine translation task. The input i to the machine translator at the design-to-model node is the design intent or ideas of designers, such as color themes, visual requirements, geometric layouts, etc. As shown in Fig. 1, for the input i, the machine translator should be able to "translate" i into a document described by a modeling language. Generally, the modeling document typically has a tree-hierarchical or semi-structured format and is composited of some container entity (non-leaf nodes) and atomic entity (leaf nodes). By this translator, the modeling documents can accurately reflect the design intent of a designer.

The input j to the machine translator at the node of model-to-panel can be regards the language translation by automatic generation mechanism combined a GUI framework. In the GUI framework language, its vocabulary is comprised of visual component names, such as Layout, Label, Button, Table or Plot. All the vocabulary should adhere to the syntax both of the modeling language and GUI framework language. The automatic generation mechanism should be able to interpret the grammar of the structured modeling language by name inferring, parse them into an equivalent token sequence of components, and then replace them by the visual elements in the GUI framework. This is what the machine translation produced. In this work, we use depth-first traversal (DFT) to enclose all the contained components in the model.

3.1. Conceptual Framework

As depicted in Fig. 1, we present an unrolled view of modeling and panel creation workflow, which explains the entire 'pipeline' from converting user's design to generated interactive panels. In this view, control engineers, data scientists, and UI designers are three types of users that are used as examples. All of them are very concerned about interface generation and user experience feedback. The control engineer is aware of which equipment need to be continuously monitored and which monitoring-related data is useful. They also want to deal with invalid data issues. The data scientist concerns about what types of data should be displayed and their relationships with each other. Sometimes they are not the end users of the generated panels. Similarly, the designer typically collects requirements from final users and determines visual techniques and interactive effects. They are all involved into an iterative model explanation and refinement. However, due to individualization, they have different demands on the final interfaces. Thus, users sometimes change their design rapidly in the feedback loop even for the same requirement and they also wish to version-control these changes.

In general, with these iterative processes, users may be able to describe their design intent in a unified modeling language and translate that model into interactive panels. Users are able to avoid hard-coded implementation during design phase and receive immediate feedback to enhance their design. With aided by the framework, all practitioners can focus more on design and rapidly prototype their designs, and version control all changes.

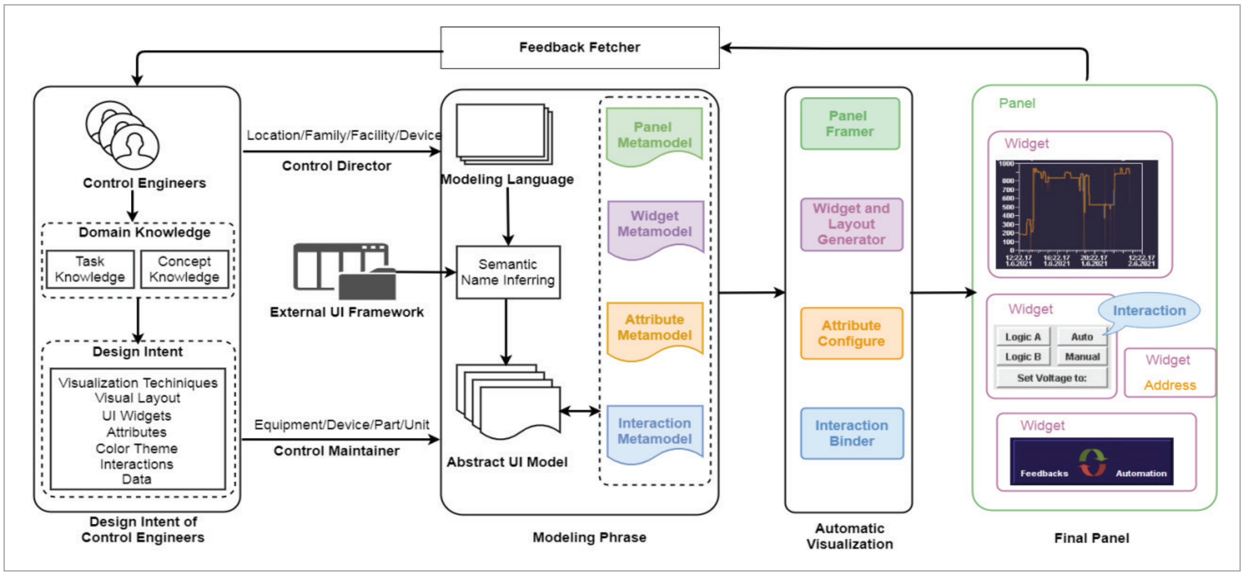
3.2. Usage Scenario in Control Systems

To understand how Mod2Panel is used to support preliminary design, refinement from feedback, and automatically generate final panels in term of distinct roles in development groups, consider the following usage scenario: Eva is a director in a control group of a large scientific facility and she seeks to create a variety of visual panels to display key indicators that can reflect the health level of the facility. Alex is an ordinary control engineer in the maintenance team. His primary responsibility is to monitor all of the physical properties of an equipment in the facility. Both of them are aware that Mod2Panel is designed to handle the challenges met by Eva and Alex and can help them convert their design knowledge and intent into operational visual panels.

Preliminary Design. The first thing Eva and Alex need to do in the preliminary phase is to inspect their control requirements by themselves. They can also consider individualization in this phase. Due to job responsibilities, Eva pays more attention to the macro indicators of the facility in a horizontal direction. She checks up on crucial devices in a Location/Family/Facility/Device manner. She is not very interested in specific devices of a facility, while Alex inspects his equipment in a detailed vertical way i.e., Equipment/Device/Part/ Unit. The preliminary design aims to convey Eva's and Alex's design intent based on their domain knowledge, such as visualization techniques, visual widgets, layout, color theme, and also interactions. During this phase, they are also allowed to communicate with other engineers for consulting their suggestions. This is the embryonic form of their initial model of design intent. Overall Eva and Alex's work on Mod2Panel is depicted in Fig. 2. The design intent reflecting the task knowledge and concept knowledge of control engineers is captured in modeling phase. Then, the models is input into the automatic visualization mechanism to generate final panels. The feedback from the GUI-driven editor is helpful for refinement and all changes will be version controlled.

Modeling and Refinement. After reviewing the design intent and deciding what and how to display information, Eva and Alex develop a variety of versions of their initial designs and then they can use manual, semi-automatical, or automatical ways to model their ideas by the proposed modeling language. In this phase, as they need to bridge the modeling language and the relevant widgets in the given UI framework, Eva and Alex should be completely conversant in the language's grammar as well as, to a lesser extent, the

Figure 2
Overall workflow of control on Mod2Panel



external UI framework. Moreover, Eva and Alex can consult the authors and look at ideas with features similar to their existing design. If they find a decent predefined sample, they may simply import it and use it as a template to quickly prototype their idea. However, if they find a sample that is not as good as the others, Eva and Alex could update their design intent. This refinement process enables intent-driven exploration of Eva and Alex's modeling.

Automatic Visualization. Towards the end of the design, the visualization model is directly input to the automated generation mechanism. This mechanism firstly scans the model document and then parses it in DFT and generates the final panels using the specified UI framework, i.e., JQuery or JavaSwing. This process of development is much faster than starting from scratch. Eva and Alex might be interested in certain functions within the panels, or just want to fine-tune the geometric layout of the widgets, a GUI-based graphical editor is provided to manually adjust the hard-coded or semi-generated model documents to make them fully compliant with the modeling language. In addition, the end users of these panels can interact with each other via the feedback fetcher and improve their designs in continuous iteration. Of course, each iteration can be version controlled. By the way, the feedback fetcher may just be an external communication platform.

4. Mod2Panel Framework

4.1. Mod2Panel Modeling Language

The modeling language in Mod2Panel provides users with a visualization-oriented abstract specification that defines what information can be visualized and how it can be presented. Using this language, Mod2Panel can fully capture the user's design intent, abstract it into a model, and persist it in a certain format. Upon the models, Mod2Panel provides an automatic mechanism to generate the final panels. Moreover, the model created by the language fully recovers the user's design requirement and can perform distributed collaboration, backup, and version control. From the perspective of data abstraction, the language can facilitate the user's design process, enabling them to save effort and focus more on design. On the other hand, the model-driven approach helps developers get out of the hard-coding development mindset, reduces the complexity of UI development, and increases productivity and creativity.

Although UI and interface are frequently used in real-world practices, they involve more than just the arrangement of various visual entities, i.e., buttons, tables, and charts. They also involve multidisciplinary aspects, such as design and psychology. As our end users are usually control engineers and modeling analyzers, we firstly try to integrate Mod2Panel into their design sessions and conceptualize the implications of prototypes. After that, the meta-models and models are designed for Mod2Panel.

4.2. Meta-Model

Meta-model defines the language for specifying models [13]. After inspecting further visualization requirements of end users, the modeling language we designed should consist of three levels: the layout, the widgets, and the interactions. The widgets here are the center of the language which should also adhere to some UI framework for rendering. Widgets are visual entities such as buttons, tables, graphs, and other self-developed controls. Their properties and attributes also can be configured during the entire life circle. Layout decides what goes inside and how to organize them. The interactions usually provide interfaces for initiating actions and data coupling.

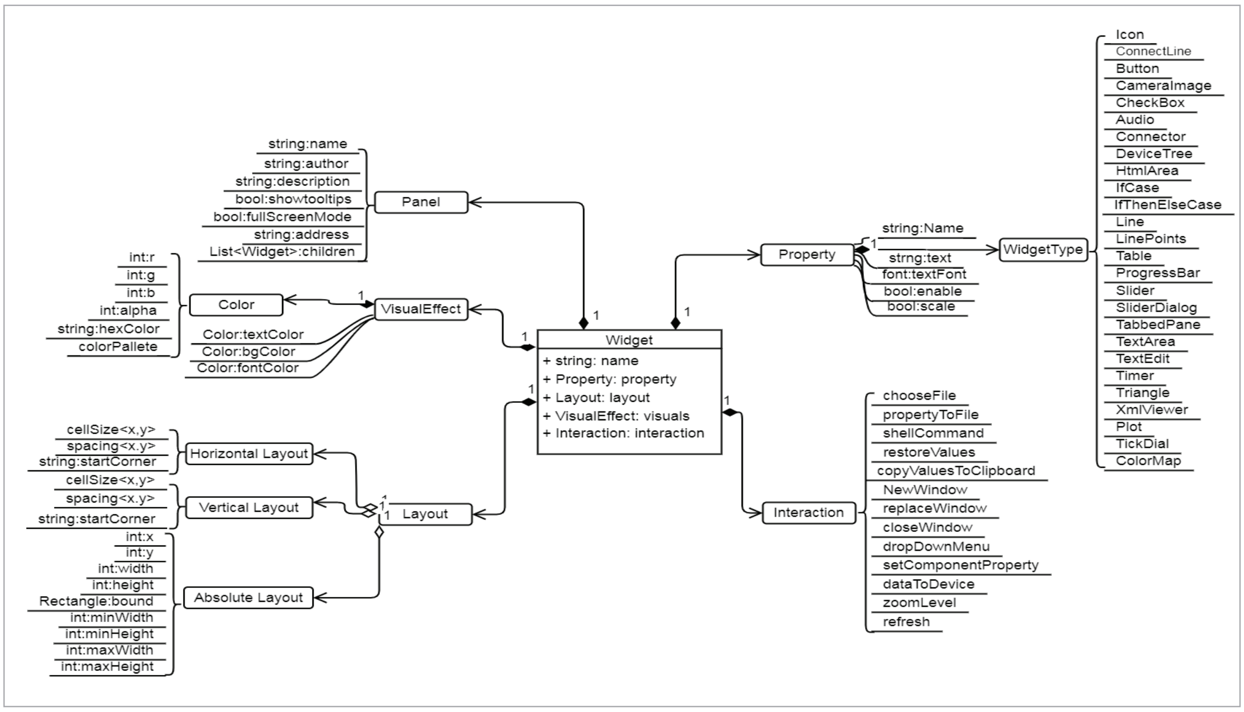
Like the Meta-Model Mechanism of UML, which is based on a four-level meta-modeling architecture [29], we design a three-level meta-model mechanism for Mod2Panel. Fig. 3 depicts the meta-model architecture proposed for Mod2Panel. They are:

- **Panel** is the top container that contains a variety of Widgets. The Panel class contains the following attributes: name, author, description, showTooltips, fullScreenMode, address, and children list. Author is actually the names of the creators, and name is used as a unique indicator of the panel in the software.

  ShowToolTips is a boolean value for deciding whether to show tooltips when the mouse is over widgets and fullScreenMode determines whether to be in a full-screen mode. Address is reserved for distributed control applications. Children is a list of widgets in the panel that are its children.
- **Widget** is the atomic building block of visualization rendering. It functions as the core part of the meta-model mechanism. As aforementioned, the widget here should have a close relationship with the widgets of a specific UI framework. It is typically a picture, button, table, or text. We associate a layout with widgets. It determines the visual level hierarchy for them, describing where the widgets

Figure 3

Meta-model and model structure for Mod2Panel



are located and how to render them. A widget also contains interactions.

- **Interactions.** We only offer interactivity at the widget level. The Interaction class describes what types of interactions are employed by this widget and some detailed configurable properties, i.e., the sender widget and the receiver widget.

4.3. Model

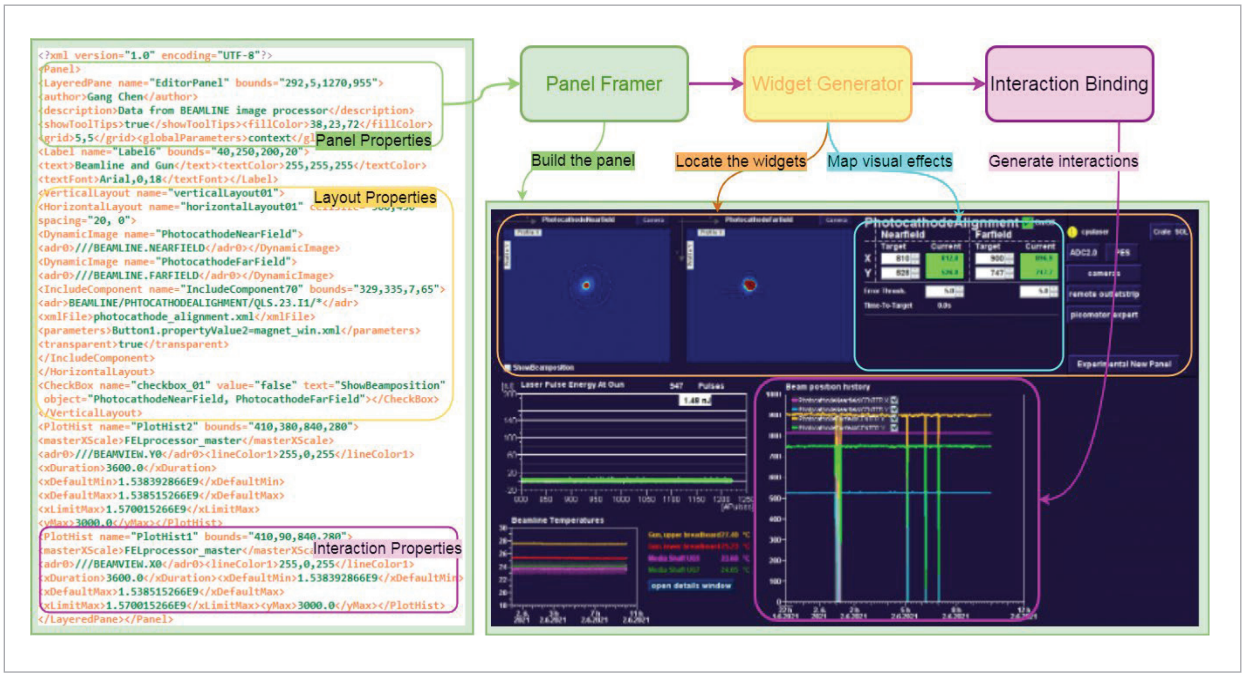
Within our proposed development approach, modeling is used as means to alleviate the complexity of model-driven interface generation. This approach should provide pre-assembled building blocks that can be used for domain and UI model construction. Therefore, it is essential to specify the modeling process in a uniform and machine-readable manner and feed their attributes and properties with desired data and interactions. Another important aspect is that it must be possible to compose the building blocks manually or automatically. In this paper, we adopt XML (the eXtensible Markup Language) [26] to represent the modeling specification. As a language, XML is commonly used to structure data for storage, interchange, and transportation. By using XML syntax and the aforementioned meta-model, we can formulate the modeling representation. Users can accurately assemble models based on their requirements and convey their design intent into them using XML.

For more clarity, we show a comprehensive example in Fig. 4 from a control scenario of a large-scale facility that elaborately details the design and auto-generation of the three-level hierarchy meta-model structure. , the left is a sample model structure for all widgets and properties, the panel framer is the generator for the panel container, the widget generator is for location arrangement and visual rendering, and interaction binding is the specifications for interactions. The layout generator and widget configuration are combined in the widget generator. Interaction generator should also work in widgets.

4.4. Automated Panels Rendering and Graphical Editor

When the modeling documents have been established, we propose to render the UI panels in a top-down way due to the restrictions of data coupling between different widgets. This automated generation mechanism enables users to render the specified pan-

Figure 4
An example in a control scenario



```
<?xml version="1.0" encoding="UTF-8"?>
<Panel>
<LayeredPane name="EditorPanel" bounds="292,5,1270,955">
<author>Gang Chen</author>
<description>Data from BEAMLINE image processor</description>
<showToolTips>true</showToolTips><fillColor>38,23,72</fillColor>
<grid>5,5</grid><globalParameters>context</gl
<Label name="Label6" bounds="40,250,200,20">
<text>Beamline and Gun</text><textColor>255,255,255</textColor>
<textFont>Arial,0,18</textFont></Label>
<VerticalLayout name="verticalLayout01">
<HorizontalLayout name="horizontalLayout01"
spacing="20, 0">
<DynamicImage name="PhotocathodeNearField">
<adr0>///BEAMLINE.NEARFIELD</adr0></DynamicImage>
<DynamicImage name="PhotocathodeFarField">
<adr0>///BEAMLINE.FARFIELD</adr0></DynamicImage>
<IncludeComponent name="IncludeComponent70" bounds="329,335,7,65">
<adr>BEAMLINE/PHTOCATHODEALIGHMENT/QLS.23.I1/*</adr>
<xmlFile>photocathode_alignment.xml</xmlFile>
<parameters>Button1.propertyValue2=magnet_win.xml</parameters>
<transparent>true</transparent>
</IncludeComponent>
</HorizontalLayout>
<CheckBox name="checkbox_01" value="false" text="ShowBeamposition"
 object="PhotocathodeNearField, PhotocathodeFarField"></CheckBox>
</VerticalLayout>
<PlotHist name="PlotHist2" bounds="410,380,840,280">
<masterXScale>FELprocessor_master</masterXScale>
<adr0>///BEAMVIEW.Y0</adr0><lineColor1>255,0,255</lineColor1>
<xDuration>3600.0</xDuration>
<xDefaultMin>1.538392866E9</xDefaultMin>
<xDefaultMax>1.538515266E9</xDefaultMax>
<xLimitMax>1.570015266E9</xLimitMax>
<yMax>3000.0</yMax></PlotHist>
<PlotHist name="PlotHist1" bounds="410,90,840,280">
<masterXScale>FELprocessor_master</masterXSca
<adr0>///BEAMVIEW.X0</adr0><lineColor1>255,0,255</lineColor1>
<xDuration>3600.0</xDuration><xDefaultMin>1.538392866E9</xDefaultMin>
<xDefaultMax>1.538515266E9</xDefaultMax>
<xLimitMax>1.570015266E9</xLimitMax><yMax>3000.0</yMax></PlotHist>
</LayeredPane></Panel>
```

els by the models rapidly to validate, refine and improve their designs. At first, Panel Framer extracts the definition of panels to render the current panel which is the top container of all widgets in the model. The Layout Generator reads out the model elements at the widget level, especially location and bound properties, to locate each widget. The Widget Generator also reads out the widget level information, mainly the attributes and properties, to render the widgets within a certain UI framework. Similarly, the Interaction Generator reads the configurations about interactions for event triggers and data coupling.

Fig. 5 describes the overview pipeline of the panel rendering composition mechanism in detail. It takes the established models as input and adapts the external UI framework to create the component hierarchy structure.

As mentioned above, Fig. 5 shows four visual interpreters of Mod2Panel. Here we describe them in detail.

Panel Framer reads the predefined information of panel-level model. Since panel is the top container in a model, it does not require any layout generator.

Layout Generator mainly concerns about the locations and bounds of each widget defined in the model. According to the widget-level information, it determines the locations and size of a widget. If there is overlap, the incoming widgets should be placed below the outgoing ones.

Widget Generator is the center of the automated rendering mechanism. It transforms the widget-level models into renderable UI widgets together with detailed configurations, such as the color definitions, the functions, or the icons.

Interaction Generator primarily manages the interactions for event triggers based on the interaction- related configurations. For example, if the interaction is about opening a new window when clicking a button. When the user clicks the specified button, another new panel will be opened which is defined by its corresponding XML file.

We are all aware that the tremendous complexity, heterogeneity, and dynamic of the environment lead to constant changes in designs. It leads to the fact that the generated panels from the original model sometimes fail to reflect the original designer's intentions. In this subsection, we provide a graphical editor so that users may visually improve their designs. With the help of this tool, users can create an iterative improvement feedback loop that closely matches their design intent. This graphical tool help users create task-specific and individual panels in particular use cases. Fig. 6 describes this graphical editor which has four sub-windows. The first sub-window in the center position is Editor Area. It contains a design-time view of the panel generated by the current model. The widgets in this panel can be visualized during design time. On the left is Component Inspector which hierarchically displays all the widgets in the current model. Users can navigate all widgets with this tool. On the upper right is the component Palette. From this widget pool, users can add a new element to the mod-

Figure 5

Automatic composition pipeline of Mod2Panel

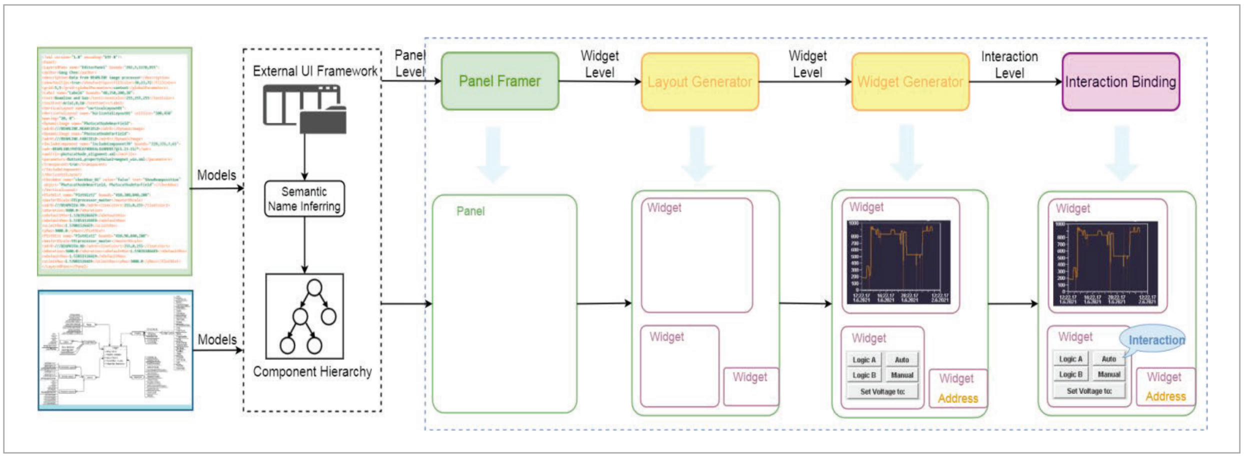
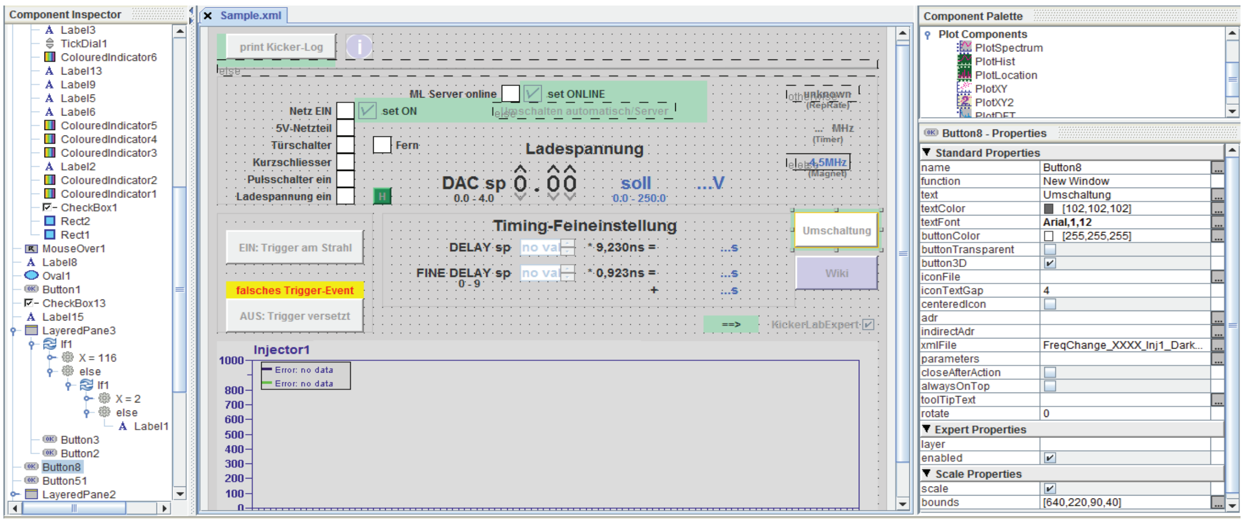


Figure 6

Graphical editing tool for control model refinement of a device



▼ Standard Properties	
name	Button8
function	New Window
text	Umschaltung
textColor	■ [102,102,102]
textFont	**Arial,1,12**
buttonColor	□ [255,255,255]
buttonTransparent	☐
button3D	✓
iconFile	
iconTextGap	4
centeredIcon	☐
adr	
indirectAdr	
xmlFile	FreqChange_XXXX_Inj1_Dark...
parameters	
closeAfterAction	☐
alwaysOnTop	☐
toolTipText	
rotate	0
▼ Expert Properties	
layer	
enabled	✓
▼ Scale Properties	
scale	✓
bounds	[640,220,90,40]

el, and drag and drop the widget to relocate and resize. On the bottom right is the Properties configuration table. It shows all the editable setting parameters for the currently active widget. In this table, users can select configuration interaction to start to customize and choose visualization techniques to set up visualization and interaction parameters.

By this tool, users can roughly add elements, resize, relocate and configure them. For detailed visual effects, manual editing is inevitable. Due to the three-level hierarchy language architecture, Mod2Panel provides configuration tools through this basic graphical editor for widgets at different levels, even for the ones in other containers.

5. Evaluation

In this section, we describe the methodology of our study, the feedback from the participants in the case and the results we draw from the feedback. We use the results to quantify the efficiency and effectiveness of Mod2Panel in real-world situations.

5.1. User Study

In order to verify the intuitiveness of our workflow and the usability of Mod2Panel, we conducted a variety of user studies with two separate user groups. The scenarios we used in the use cases were complex control-oriented systems mainly for monitoring (in Germany) and modeling level analysis for highly interactions (in China), simulating a real- world environment. The study aims to understand where the system can be improved and whether the necessary task-specific features for real-world scenarios have been covered by the framework.

Methodology and Study Design. We know that each user has different technical and aesthetic priorities. In order to avoid this type of discrepancy, we decide to adopt similar approaches to pair analytics studies [17], allowing each participant to inject their individual design intent into both the control system and modeling system. We trained nine pairs of participants and ensure that they fully learned how to use Mod2Panel. The target participants were requirement analysis users (RA,●), model developer (MD,●) and maintenance staff (MS,●), among which model developers are mainly responsible modeling and generate the final panels. Each study consists of three phases: a quick introduction to the real-world requirements, construction of panels and then a semi-structured review for their user experience. All participants are asked to properly transfer their design intent into models by communication and thinking, taking as much time as they want. The second phase is to use Mod2Panel to realize their design intent. The final phase is to collect their comments, complains, or anything else that reflects the discrepancy between their initial expectation and the final panels. After three phases are finished, all participants swap their roles, i.e., the control group will evaluate the usability of the modeling system under the guidance of all members of modeling group, and vice versa. All user studies are recorded by technical staff.

Participants. We chose the 18 participants from two different user groups, who were working in the control systems and modeling level analysis respectively. For RA, we asked 3 pairs of engineers to take charge of three distinct subsystems. For MD, we chose 3 pair of engineers with academic background of computer science, who have worked long time in the field of control and modeling analysis. And for MS, we invited 3 pairs of maintenance engineering from the two groups. All participants were currently working in the two groups and one female participant must be involved in RA, MD and MS.

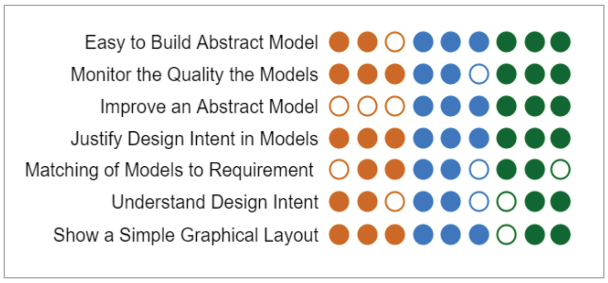
Tasks. Two out of three participants were working with control systems, while the rest of participants must join the tasks of modeling analysis, because control system was more complex and required more manpower. All the participants were guided through the interactions along the task requirement understanding, modeling and refinement from the feedback of other co-participant.

5.2. User Feedback

In this subsection, we describe and analyze the feedback received from the participants during the involved three phases (requirement analysis and abstract modeling, design intent extraction and development, refinement and panel re-generation). We noticed that all participants must join all these phases, not just finishing their own job, and gave the comments (positive and negative, need to make a human-assist statistics and analysis) based on their user experience.

Requirement Analysis and Abstract Modeling. Requirement analysis is the collection process of control- and modeling-related development. During this period, the participants tried to make the outcome of the requirement analysis practical, thorough, and reflect the real-world scenarios. Meanwhile, they were also allowed to refer to existing models and then finished their abstract models in the modeling language which only contained the skeleton of visual widgets. In this phase, when the participants were asked about their expected user experience for Mod2Panel, the most frequent answers were modeling simplicity, model quality monitoring, improvement of abstraction, and design intent transferring, as depicted in Fig. 7.

Figure 7
Subjective results of user feedback in requirement analysis and abstract modeling



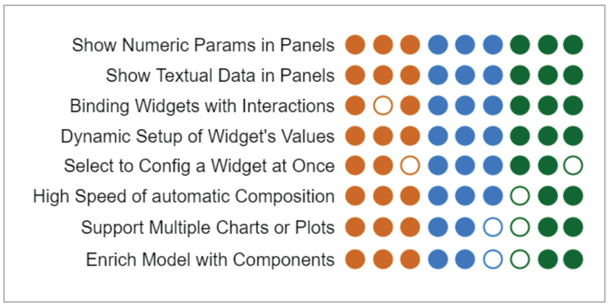
Within our framework, this can be seen as an example of the expressiveness and scalability of the modeling language. Besides that, due to dynamic requirements, participants were still concerned about whether it was easy or not to edit the abstract models and the close matching between design intent and the models. During this phase, we have noticed that the requirement analysis staff was mainly concerned about the layout and fundamental components of panels rather than the details of widgets. From the comments, we found that the control group was not satisfied with the skeleton visualization of the graphical editor. They complained that the skeleton only was represented by a variety of rectangles, which were meaningless for design. MD and MS also suggested adding more widgets for data visualization.

Design Intent Extraction and Development. Each panel has captured a set of design intent, reflecting the participants' knowledge about control and modeling analysis. The abstract models from the phase of Requirement Analysis and Abstract Modeling are mainly about visual skeleton without detailed configuration parameters. In order to make sure that the process of creating panels could cover the design intent of all practitioners, we followed a predefined procedure to precisely extract the design intent. First, all participants in the first phase who were involved with the requirement analysis should also participate in this phase. They explained all the panels one by one. If the developers in this phase faced challenges, they ought to explain, especially the potential design parameters, such as color theme, the content of labels, and tables. Then, the developers enriched the models in the modeling language and generated the final panels. Finally, all the participants should check and confirm that all data and parameters were correctly distinguished.

During this phase, RA was satisfied with added visual effects that made their deigns more colorful and meaningful. MD and MS staff were mainly concerned about the richness of the widget pool and the expressiveness of details of the design intent. Most of the feedback was positive, as shown in Fig. 8. However, due to the constraints of Mod2Panel, part of the plots or charts were not supported. Mod2Panel provides various plots for data trends, not for data classification, such as PieChart and BarChart.

Figure 8

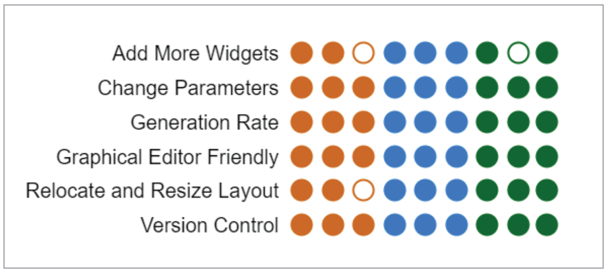
Subjective results of user feedback in design intent extraction and development



- **Refinement and Panels Reconstruction.** After the generation of panels in the second phase, the MS staff should use them in real control systems and modeling level analysis systems, and confirmed with RA to make sure that the design intent of RA was captured as much as possible and visualized the models as closely as possible in the panels.
- The MD staff examined the original abstract models and result of requirement analysis, and then enriched the XML described models.
- The MD staff version controlled these models in a certain repository, such as database or Git.
- The MD staff generated panels according to the models.

Figure 9

Subjective results of user feedback in refinement and panels reconstruction



- The RA, MD and MS staff examined and checked the panels and compared them with the original design intent.
- The MD iteratively kept improving the models by GUI-aided editor to make the models better.

During the refinement and reconstruction phase, the feedback and design intent was somewhat mixed. Participants' personalities and aesthetic preferences varied. Some of them may have quite high hopes, suggestions, and aspirations. They wanted to refine the models in some ways, such as adding more widgets, modifying parameters, or changing plots, as depicted in Fig. 9. Moreover, some of the participants were doubtful. Although they did not question the usability of this tool, but rather the proficiency, because Mod2Panel sometimes is a little redundant to achieve the design objective.

Demo-panels of Exemplary Scenarios. A set of panels have been developed by the proposed Mod2Panel framework in the context of control systems and modeling level analysis. Here we noticed that when the case studies were conducted, some CAE applications and numerical simulations were still under construction. So the panels of modeling analysis were not so perfect and colorful. However, it had no impact on the effectiveness of Mod2Panel. The construction result of control systems can be found in Fig. 10. These representative panels are mainly related to macro-controls which are more colorful and visually attractive. The panels from the modeling analysis group are described in Fig. 11. The demo panels show that the generated panels by Mod2Panel framework represent the design intent of practitioners and have an appearance with aesthetic functions combined with layout, colors, and visual effects.

Figure 10

Selected panels for macro-control

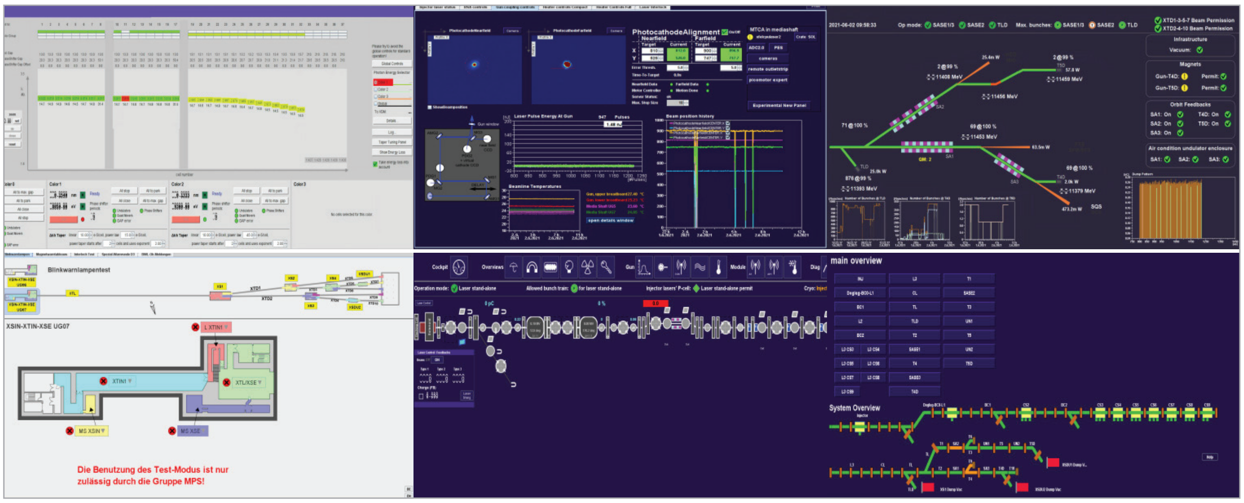
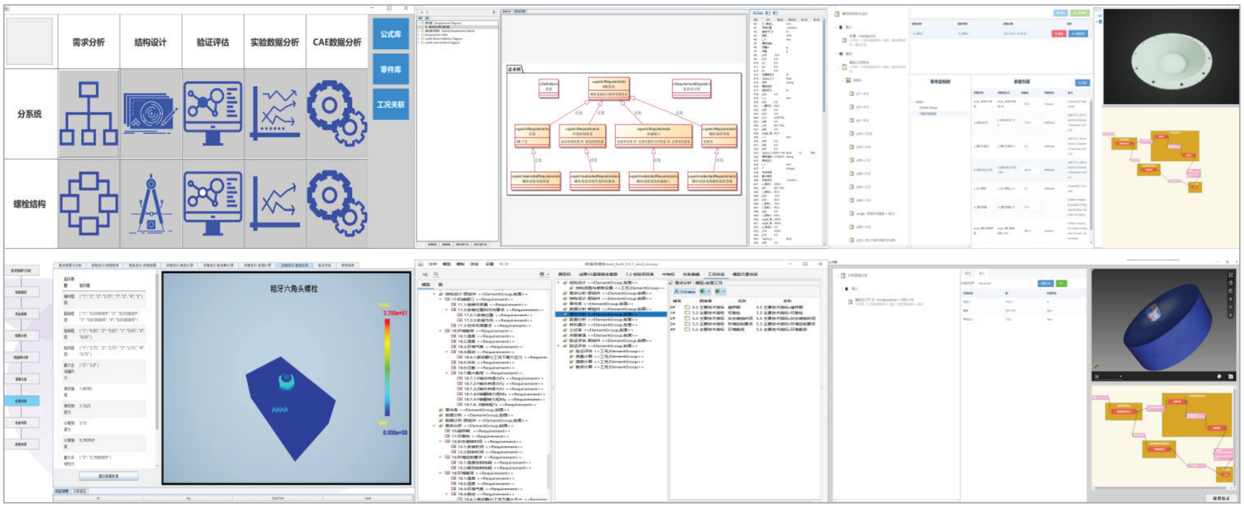


Figure 11

Panels from modeling level analysis model, which are used to navigate between different design perspectives of the system



5.3. Lessons Learned and Future Work

Generally, the feedback from all participants was positive. Nevertheless, the participants also suggested some complementary ideas for Mod2Panel during the use studies. One of the main complaints about Mod2Panel was its complex representation in XML of the model structure. They suggested using a more abstract and highly expressive approach to represent models. Another significant point was interaction binding. Only some complicated widgets supported distributed protocol or HTTP based data coupling. This meat if we input data into these types of widgets, we should implement the corresponding interfaces on the server side, otherwise, they could not receive data. Some requirement analyzers suggested that an additional meta-modeling GUI editor should be extended for them because the current GUI tool was too complicated. This meta-modeling tool should only aim at skeleton modeling that could be used to create better design interfaces. Finally, some more visual widgets should be included in the framework. This could be achieved by a way to open sourcing the widget pool to other developers. A further extension could make this tool suitable for more advanced analysis and control tasks.

Next, we are going to conduct more studies in more scenarios to identify different requirements of users with respect to their special tasks. By doing this, we want to improve the functionality and usability of Mod2Panel. When we get further findings, we will enhance our modeling language, for example, converting from XML to JSON for better abstraction, or supporting more open source widget libraries. In addition, in order to use the saved models as templates to reduce the need for time-consuming and manual collection, in the future we will create a rule-based or machine learning-based mechanism to capture users' behaviors and patterns to recommend models for particular users. Users can quickly edit these recommended models to generate their own versions.

6. Conclusion

This paper proposed Mod2Panel, a framework for model-driven automated generation of interactive panels, which can capture the theoretical and practical knowledge of users as well as their design intent. Using this framework, users can rapidly model, prototype and produce interactive panels for the purposes of control or analysis. Mod2Panel contains three fundamental stages. First, the modeling language is designed to help users in summarizing their design intent and converting it into an understandable format that can be backed up and version controlled. Second, an automatic visualization composition mechanism is provided to generate operational panels. Users can benefit from prototyping and validating their designs. Finally, a GUI-assisted tool aids users iteratively enhancing and improving their designs. Mod2Panel is evaluated in a variety of user studies. The result of the evaluations shows that Mod2Panel can abstract and visually show the designs in real-world scenarios.

Acknowledgement

This research has been supported by National Nature Science Foundation under Grant 61903348. We especially thank all the participants of the user studies both in Germany and China for their generosity and creative work.

References

1. Aksu Ü., del-Río-Ortega A., Resinas M., Reijers H.An Approach for the Automated Generation of Engaging Dashboards. Lecture Notes in Computer Science on the Move to Meaningful Internet Systems, 2019, 363-384. https://doi.org/10.1007/978-3-030-33246-4_24
2. Alexander C., Ishikawa S., Silverstein M.A Pattern Language. Oxford University Press, 1977.
3. Lawson B. How Designers Think: The Design Process Demystified. University Press, Cambridge, 2006. https://doi.org/10.4324/9780080454979
4. Onder R., Bayram Z. Xsl Transformations: A Delivery Medium for Executable Content over the Internet. Doctor Dobbs Journal, 2007, 32, 48-53.
5. Biron P., Malhotra A. Consortium WWW. XML Schema Part 2: Datatypes. W3C Recommendation, 2001.
6. Bostock M., Ogievetsky V., Heer J. D3: Data-Driven Documents. IEEE Transactions on Visualization and Computer Graphics, 2011, 17(12), 2301-2309. https://doi.org/10.1109/TVCG.2011.185
7. Bäuerle A., Cabrera A., Hohman F., Maher M. Symphony: Composing Interactive Interfaces for Machine Learning. In the 2022 CHI Conference on Human Factors in Computing Systems, 2022, 1-14. https://doi.org/10.1145/3491102.3502102
8. Chen C., Su T., Meng G., Meng G. Xing Z. From UI Design Image to GUI Skeleton: A Neural Machine Translator to Bootstrap Mobile GUI implementation. In the 2018 International Conference on Software Engineering, 2018, 665-676. https://doi.org/10.1145/3180155.3180240
9. Deng J., Kemp E., Todd E. Managing UI Pattern Collections. In the 2005 International Conference on Computer-human Interaction: Making CHI Natural, 2005, 31-38. https://doi.org/10.1145/1073943.1073951
10. Engel J., Märtin C. Pamgis: A Framework for Pattern-Based Modeling and Generation of Interactive Systems. In the 2019 Human-Computer Interaction Conference, 2019, 826-835. https://doi.org/10.1007/978-3-642-02574-7_92
11. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. MA: Addison-Wesley, 1995.
12. Goel V., Pirolli P. The Structure of Design Problem Spaces. Cognitive Science, 1992, 16(3), 395-429. https://doi.org/10.1016/0364-0213(92)90038-V
13. Group OMG. Model Driven Architecture, MDA Guide, 2013.

14. Guerrero Garcia J., Vanderdonckt J., Calleros J., Winckler M. Towards A Library of Workflow User Interface Patterns. In the 2008 International Conference on Interactive Systems Design, Specification, and Verification, 2008, 96-101. https://doi.org/10.1007/978-3-540-70569-7_9
15. Hovsepyan A., Baelen S., Vanhooff B., Joosen W. Key Research Challenges for Successfully Applying MDD Within Real-Time Embedded Software Development. In the 2006 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation, 2006, 49-58. https://doi.org/10.1007/11796435_7
16. Ide N., Bonhomme P., Romary L. XCES: An XML based Encoding Standard for Linguistic Corpora. In the 2000 Annual Conference on Language Resources and Evaluation, 2000.
17. Kaastra L., Fisher B. Field Experiment Methodology for Pair Analytics. In the 2014 International Conference on BELIV Beyond Time and Errors: Novel Evaluation Methods for Visualization. https://doi.org/10.1145/2669557.2669572
18. Kintz M. A Semantic Dashboard Description Language for a Process-Oriented Dashboard Design Methodology. In the 2012 International Workshop on Model-based Interactive Ubiquitous Systems, 2012.
19. Leroux D., Nally M., Hussey K. Rational Software Architect: A Tool for Domain-Specific Modeling. IBM Systems Journal, 2006, 45(3), 555-568. https://doi.org/10.1147/sj.453.0555
20. Li D., Mei H., Shen Y., Su S. ECharts: A Declarative Framework for Rapid Construction of Web-Based Visualization. Visual Informatics, 2018, 2(2). https://doi.org/10.1016/j.visinf.2018.04.011
21. Lopes S., Silva C., Tavares A., Monteiro J. Extending ArgoUML for Real-Time UML. In the 2004 International Conference on Advances in Computer Science and Technology, 2004, 431(57), 191-197.
22. Märtin C., Roski A. Structurally Supported Design of HCI Pattern Languages. In the 2007 International Conference on Human-Computer Interaction, 2007, 1159-1167. https://doi.org/10.1007/978-3-540-73105-4_126
23. McInerney J., Lacker B., Hansen S., Higley K. Explore, Exploit, and Explain: Personalizing Explainable Recommendations with Bandits. In the 2018 ACM Conference on Recommender Systems, 2018, 31-39. https://doi.org/10.1145/3240323.3240354
24. Neuendorf D. Review of Magicdraw UML, 11.5 Professional Edition. Journal of Object Technology, 2006, 5(7), 115-118. https://doi.org/10.5381/jot.2006.5.7.r1
25. Palpanas T., Chowdhary P., Mihaila G., Pinel F. Integrated Model-Driven Dashboard Development. Information Systems Frontiers, 2007, 9(2-3), 195-208. https://doi.org/10.1007/s10796-007-9032-9
26. Rehem Neto A., Saibel Santos C., Carvalho L. Touch the Air: An Event-Driven Framework for Interactive Environments. In the 2013 Brazilian Symposium on Multimedia and the Web, 2013, 73-80. https://doi.org/10.1145/2526188.2526216
27. Satyanarayan A., Russell R., Hoffswell J., Heer J. Reactive Vega: A Streaming Dataflow Architecture for Declarative Interactive Visualization. IEEE Transactions on Visualization and Computer Graphics, 2015, 22(1), 659-668. https://doi.org/10.1109/TVCG.2015.2467091
28. Seiger R., Niebling F., Korzetz M., Nicolai T. A Framework for Rapid Prototyping of Multimodal Interaction Concepts. CEUR Workshop Proceedings, 2015, 1380, 21-28.
29. Selic B. OMG Unified Modeling Language 2.0. OMG, 2005.
30. Shneiderman B. Bridging the Gap Between Ethics and Practice: Guidelines for Reliable, Safe, and Trustworthy Human-Centered AI Systems. ACM Transactions on Interactive Intelligent Systems, 2-020, 10(4), 1-31. https://doi.org/10.1145/3419764
31. Tiedtke T., Krach T., Martin C. Multi-Level Patterns for the Planes of User Experience. Theories Models and Processes in HCI, 2005, 4.
32. Topalian Rivas G., Wassermann J., Severengiz M., Kruger J. Automated Dashboard Generation for Machine Tools with OPC UA Compatible Sensors. In the 2020 International Conference on Emerging Technologies and Factory Automation, 2020, 1009-1012. https://doi.org/10.1109/ETFA46521.2020.9212136
33. Vlaeminck H., Vennekens J., Denecker M. A Logical Framework for Configuration Software. In the 2009 International ACM Conference on Principles and Practice of Declarative Programming, 2009, p.141-148. https://doi.org/10.1145/1599410.1599428
34. Vázquez-Ingelmo A., García-Peñalvo F., Therón R.Connecting Domain-Specific Features to Source Code: Towards the Automatization of Dashboard Generation. Cluster Computing, 23(3), 1803-1816. https://doi.org/10.1007/s10586-019-03012-1
35. Welie M., Trætteberg H. Interaction Patterns in User Interfaces. In the 2000 Pattern Languages of Programs Conference, 2000, 13-16.

This article is an Open Access article distributed under the terms and conditions of the Creative Commons Attribution 4.0 (CC BY 4.0) License (http://creativecommons.org/licenses/by/4.0/).