Fields In Stateful Model

FISM & TomFism

FISM is a software architectural pattern which improve efficiency of GUI development by separating the physical implementation GUI and logical UI information layer and standardizing the logical UI model.

FISM is a variation / combination of MVVM and MVP patterns.

	MVP	MVVM	FISM
View / UI	View	View	Physical UI Layer
Mediator	Presenter	View Model , Virtual Component	Logical UI Layer
Data Object	Model	Model	Data Model

FISM Logical UI Layer is analogous to View Model of MVVM. However MVVM defines Model View as

The view model of MVVM is a value converter, meaning the view model is responsible for exposing (converting) the data objects from the model in such a way that objects are easily managed and presented

It does not define how the data objects should be presented.

This leaves room for developers' individual creativity. Yet not every developer enjoys this freedom, and that's why patterns are invented allowing application developers to follow a standardized and efficient architecture (best to be at detail level) and can focus on the business domain

For this reason FISM defines the Logical UI Layer as following:

- should present full information to Physical UI layer needs (but can be more)
- should present information in structure close to the logical structure of the Physical UI layer
- The Logical UI Layer should be organized in View Model (don't confuse with the View Model of MVVM) and View Field.
- FISM View Model and View Field are OOP classes which form tree structure analogous to that of Folder-File structure

- View Field
    - The most basic element for the logical UI Layer.
    - Represents elementary UI widgets e.g.
        - Input box
        - Drop down
        - Button
    - Covert data from Data Model and present to Physical UI
    - Should NOT store any persistent state variables
    - View Field can be extended to specialized View Fields. E.g.
        - Date View Field
            - With date validator
        - Select View Field
            - Present list of key-value pairs
        - Button View Field
            - Provide command logic
    - The most basic View Field provides
        - Getter / setter for target value in Data Model
        - Optionally visible / editable flags / rules
        - Error message display mechanism
    - View Field can be extended at application level to create customized View Field.
    -
- View Model
    - Container for View Fields
    - View models can contain children view Models
    - Covert data from Data Model and present to Physical UI
    - Should NOT store any persistent state variables
    - The base View Model provides
        - Pointer to the node in Data Model that the View Model is binding to
        - Optionally visible / editable flags / rules that children View Fields / View Models use as default value
        - Basic initialization for constructing / restoration of the View Fields / children View Models from the Data Model
        - Utilities e.g.
            - Traversing up and down the full hierarchy View Model tree
            - Refreshing from / to Data Model
            - Enforce the visible / editable rules

    - View Model should be extended and nested at application level to form customized Logical UI layer according to biz need.

- Stateful Data Model
    - Presentation of the biz model analogous to MVVM Models
    - Provide data source to FISM View Field and View Model
    - And any persistent state used by view model or view field
    - The full hierarchy of Data Model must be perfectly serializable and deserializable
        - Which means the full state can be captured
        - The captured state can be restored to form the exact programming model of FISM Data Model
        - Since View Field and View Model DONOT store any state, all state is from the Data Model.
        - With the same Data Model, reconstruction of the same View Model (with its children View Models and View Fields) should be exactly the same as the moment of Data Model Serialization.

-
    - View Fields and View Models presents the data model as a tree like structure with all the information needed by the physical UI
- Model Serialization / Restoration
    - Data model must be fully de/serializable
    - View Model can be fully restored by deserialzed Data Model
    - Thus the physical UI can be exactly restored to any state

- Physical UI Layer
    - Should be built with FISM component
    - Basic widget such as input box and drop down list components are View Field level components and must be created by extendingSimpleComponent
    - The Physical UI Layer View Field level Component always binds to a View Field
    - View Model levelComponents encapsulate View Field level components should be created by extending the ComplexComponent
    - The Physical UI Layer View Model Component always bind to a View Model.
    - At application level, the Physical UI Layer SHOULD contain only the binding declaration to View Field or View Model but NOT any other logic .
    - The Physical UI layer is free to have its only layout and style
    - However the behavior and state presentation (field value, visible , editable , validation etc) must follow the View Field / View Model STRICTLY.
    - Therefore when View Model is restored by deserialization. The Physical UI must look and behave exactly the same as the moment when the serialization was done, even the popup opened / closed status.

TomFism

TomFism is an implementation of FISM using typescript targeting HTML GUI.

The following features are implemented

- FISM View Model
    - Base class of View Model
    - App View Model
        - App Level root View Model
        - Contains developer friendly children View Models for simulation / recording and View Model for Log Viwer
        - These developer friendly View Models are transparent to users but are useful for developers and testers
        - Container for application level View Models which are called Page View Models
        - Mechanism for configuring attributes down to View Field level
            - Each View Field or View Model can be configured to in/editable in/visible
            - JSON format config file can be applied at run time.
    - Base class of View Field
    - Typed View Fields
        - Date View Field
        - Selection View Field
        - Check box View Field
        - Command Button View Field
    - Perfect Serialization / Deserialization
    - FISM Routable Command / FISM Service
    - Provides a path definition (nodepath) for referencing any View Field / View Model in the Logical UI Layer.
    - Utilities
        - Logging facilities
            - Logs are saved to the LocalStorage of browser
            - Different types of loggers etc
                - Data Model logger
                - Check point logger
                - Command Logger
                - Information Logger
            - Logs can be downloaded as JSON files
            - Saved logs can be uploaded and reused

- Simulation / recording
    - The full series of user actions can be recorded and saved to log files.
    - The log files can be replayed using deserialization
    - Auto Integration test is achieved by capturing user actions and replaying the actions
    - Check Points can be added in code that can be optionally enabled to allow unit testing within Auto Integration Testing replay.

Tomgular

An Angular 7 implementation of Physical UI Layer providing the basic components. Tomgular uses two way binding of Angular. Display values are pulled from the View Field getters. User change / input is automatically pushed to Data Model via setter of the corresponding View Field.

The following components are implemented

Component	Description
TomText	Read Only Tex
TomButton	Command Button
TomDateBox	Input with date pick
TomCheckBox	Check Box
TomSelect	Dropdown
TomRadio	Radio button
TomDebugger	Composite Component for debug / list View Model / Data Model structure
TomMsgList	Display List of Message e.g. error msg
TomInput	Input box
Sim Dirver	Composite Component with the floating top button expandable to show tools for user action recording / replay / Auto Integration Test etc
Simple Link	Link Command
TomButtonInput	A read only input box, can execute a command e.g. popup when clicked
TomClickable	A directive for executing a command for any component
TomCheckPointResult	A composite component listing Check Point Results
TomFile	Composite component for upload file

TomLog	Composite component for listing / display download log
TomPopup	For opening another composite component as a popup
TomTextArea	Text Area
TomTextBox	Read Only input box
TomUploader	Composite component for upload file
	Composite component for paginator
TomViewTable	Composite component for table
TomSessionMgr	Composite component for listing all log sessions
TomViewSort	Directive for sorting

Benefits of Building Apps with TomFism and Tomgular

- The logical UI layer can better present the full information to developers compared to the physical UI which usually hides most of the information. E.g.
    - An invisible field
    - Editibility of a field
    - Possible error message
- Much easier to build a Logical UI Layer than physical UI layer
- Developer can have an overview of the complete Logical UI Layer easily for
    - Testing the logical flow
    - Examine the model
- UI and logic are totally decoupled.
- Developer only need to control the Data Model or Logical UI Layer which is much easier than direct manipulation the mixture of Physical UI and Data Model.
- Standardize view field and view model as building blocks / units for logical UI Layer.
- Standardize the physical UI Layer building blockencourages clearer UI structure and UI reusability.

- Standardization allows easier framework level control of the model down to the most elementary level : View Field
- Providesstandard field / model referencing mechanism.
- The permission / behavior of each View Field can be controlled at framework level (e.g. configuration) instead of only at customization level (hard coding)
- Other framework level utilities are also possible e.g.
    - Perfect Serialization / Deserialization
    - User Action Recording / replay / Auto integration test
        - Integration can be automated is a big convinience for both testers and developers
        - Testers can record test
            - Replay fully
            - Replay step by step
            - Modify the input data before replay the test
            - Check points can identify any descrephency compared with the control sample.
        - Developers can jump to any stage without going thru many manual input or changing code.