BTP600 Visual Studio Code Open-Source Final Presentation

**Script**

**Intro**

Soodeh: Intro to visual studio code open-source Project

Visual Studio Code (VS Code for short) is an open-source editor for Linux, Mac, and Windows that supports intelligent code completion, syntax highlighting, and code refactoring, debugging, and snippets. Visual Studio Code has written with C ++, Node.js, Electron framework (software framework), and other web development technologies based on the Monaco Web Code Editor. The smaller size and having a large repository of plugins, makes it a serious competitor to other editors. The software has developed by Microsoft and is now available for free and open source.

Visual Studio Code was unveiled by Microsoft on April 29, 2015, at the Build Conference, and a preview was released shortly after. On November 18 of that year, Visual Studio Code has released under a security license on Github.com.

Alvia: Intro to research question

Taking a look at Visual Studio, one of the regularly used aspects when working on any project is the explorer section on the left of the window. That can be used to track and manage all the files within a project. There is a portion of the explorer tab that has your main folder name, and there are tools beside the folder name that allow a user to add a file, add a folder, refresh the explorer, and collapse folders in the explorer. These tools are our focus area for this presentation.

While analyzing this area of the explorer we focused on which design patterns were used in the implementation of the explorer tools, and how would one go about adding a tool using the respective design patterns?

Alvia: overview of the patterns

While analyzing the code de for the explorer tools, tools, in the open-source code, there were a few different design patterns such as Builder, Decorator, Strategy, Adapter, and prototype being used, however the ones we focused on were the Composite and Command Design Patterns.

Soodeh: Composite pattern …

Composite is a structural design pattern that allows composing objects into a tree-like structure and work with it as if it was a singular object.

Composite’s great feature is the ability to run methods recursively over the whole tree structure and sum up the results.

The Composite pattern is pretty common in TypeScript code. It’s often used to represent hierarchies of user interface components or the code that works with graphs. Visual Stodio code is writen in TypeScript.

In other words, this pattern allows you to handle a set of similar components as a group, being able to execute a particular operation on them and collecting the result from them all.

The interesting part about this pattern though, is that it’s not a simple group of objects, it can contain entities or groups of entities, and each group can at the same time, contain more groups. This is what we would call a Tree.

In vs code, we can find this pattern everywhere. In explorer, by examining the explorerView.ts, I recognize these participants:

ExplorerView class as a composite, ViewPane abstract class as it's component, All the existing Action2 class callings as leaves and Explorer itself, as a client in our case.

as you can see, ExplorerView is extends from ViewPain which is an abstract class extends from Pane abstract class and implements IView. Pane abstract class is extends from Disposable abstract class and implements IView as well.

If you have an object tree, and each object of a tree is a part of the same class hierarchy, this is most likely a composite. If methods of these classes delegate the work to child objects of the tree and do it via the base class/interface of the hierarchy, this is definitely a composite.

inside the explorerView class, we can see multiple private members, like tree! which could be another evidence for extistance of composite pattern in VS codes explorer. these private memebres inherted from other classes and components. for example, tree! is inherited from WorkbenchCompressibleAsyncDataTree class. and so on...

- Composite: (ExplorerView)

https://github.com/microsoft/vscode/blob/3aca77c01618839353d4360e72c1cac971276621/src/vs/workbench/contrib/files/browser/views/explorerView.ts#L133

defines behavior for components having children.

stores child components.

implements child-related operations in the Component interface.

- Component: (ViewPane) https://github.com/microsoft/vscode/blob/3aca77c01618839353d4360e72c1cac971276621/src/vs/workbench/browser/parts/views/viewPane.ts#L159

declares the interface for objects in the composition.

implements default behavior for the interface common to all classes, as appropriate.

declares an interface for accessing and managing its child components.

(optional) defines an interface for accessing a component’s parent in the recursive structure and implements it if that’s appropriate.

- Leaf: (Action2)

https://github.com/microsoft/vscode/blob/3aca77c01618839353d4360e72c1cac971276621/src/vs/platform/actions/common/actions.ts#L514

represents leaf objects in the composition. A leaf has no children.

defines behavior for primitive objects in the composition.

- Client: (Explorer)

manipulates objects in the composition through the Component interface.

Related patterns:

Chain of Responsibility is often used in conjunction with Composite. In this case, when a leaf component gets a request, it may pass it through the chain of all of the parent components down to the root of the object tree. Often the component-parent link is used for a Chain of Responsibility.

Decorator is often used with Composite. When decorators and composites are used together, they will usually have a common parent class. So, decorators will have to support the Component interface with operations like Add, Remove, and GetChild.

Alvia: Command pattern

A command pattern encapsulates a request as an object allowing you to parameterize clients with different requests, queues or log requests and support undoable operations. Useful for test after test/ task after task operations.

The command pattern is good for toolkit objects, allowing them to make requests by turning itself into an object. Command patterns consist of an abstract command class that declare an interface for executing operations, concrete command subclasses that specify a receiver-action pair by storing the receiver as an instance variable and implementing execute to invoke the request.

Command patterns parameterize objects by an action to perform in this case each tool in the explorer. In visual studio the command patterns are used to trigger actions. The use of command patterns in the explorer is also to specify, queue and execute the different tool requests at different times. Commands also support changes, so when user exits, or the program crashes, the changes made will be reapplied. If you added a file and the program crashes, the file will be there when you reopen the program.

Starting from explorerView.ts when the code is implementing each tool it encapsulates each tool in its own object in a class registerAction2 that is defined in action.ts and extends Action2() class in action.ts. In explorerView.ts each tool object is an instance of registerAction2 in the action.ts where the definition of the function is implemented, then creates new definitions for the Action2 member functions. In the registerAction2() function it uses command patterns when adding a command/tool to the explorer. In like 525 in action.ts file there is a function that calls registerCommand function in commandManager.ts with a command.id, a handler, and a description, then binds the command ID to a handler function in the extension(command). The handler function will be invoked whenever the extension command(tool) is executed.

There is also a menu area in registerAction2() that adds the menu item (tool) to the menu, it appends the command to the menuRegistry and then lastly bind the command. If f1 is false, t

Continuing on to the next part of creating a tool, is implementing the Action2() member functions. A Action2 is the abstract command class that declare an interface for executing operations, defined in action.ts and is used by registerAction2() and in this function. Action2 class has 2 member functions definitions, a constructor and a run function. In the constructor it sets up the name and icon of the new tool, and assigns an id and creates the tool. The run function executes the command after the command has already been registered as previously explained in the registerAction2 base implementation in action.ts. The run function just makes a constant commandService object that gets the current command by calling IcommandService interface and then exectutes the command with a new command id in the form of a string using the VS code API to programmatically execute a command.

**Participants:**

* **Command:** declares an interface for executing an operation: **registerAction2()**
* **Concrete Command:** defines a binding between a recieer object and an action: **registerCommand because this binds the command ID to a handler function in the extension(command).**
* **Client:** creates a concrete command object and sets its receiver: **Explorer**
* **Invoker:** asks the command to carry out the request: **Action2: the run function calls executeCommand**
* **Receiver:** knows how to perform operations associatewd with carrying out a request: **commandService in the run function**

Soodeh and Alvia: Answer research question; show how you can use your patterns to go about adding a feature.

To answer the question we had before analyzing the code, the different design patterns we chose to focus on are composite and command pattern

To go about adding another tool feature using command pattern is to follow the same implementation as the other tools by creating a new object using registerAction2 that extends Action2. Similar to the other tools, the constructor would create an id, a title, set f1 to true if it is an event, false otherwise, set icon, and menu In the run function, if the f1 is set to an event and set to true, then I would follow the implemmentation of the run function in the tools like collapse and refresh and handle it like an event. But if it is false, similar to adding a file and adding a folder, that is when you, create an object that inherits from the interface ICommandService and call executeCommand to execute the command.

If we want to add another tool feature to the explorer using the composite pattern, my suggestion is adding a leaf to the existing structure. For now, we have the registerAction2 callings as leaves of our composite pattern. If we are about to add another feature, the best way is adding another calling of registerAction2 function with appropriate arguments and signatures. If we are adding an event feature like refresh or collapse which are present there now, we must follow their leads and if adding a feature like creating the file and folder, follow their samples. This could be the easiest and fastest way to add a tool or feature to the existing explorer in vs code using the composite pattern. The structure of the tree will stay the same and just leaves would be added.

Soodeh: Conclusion

The Explorer is used to browse, open, and manage all the files and folders in your project. VS Code is file and folder-based - you can get started immediately by opening a file or folder in VS Code.

After opening a folder, the contents of the folder are shown in the Explorer. You can do many things from here:

\* Create, delete, and rename files and folders.

\* Move files and folders with drag and drop.

\* Use the context menu to explore all options.

To be able to do so, design patterns have the significant roles in vs code source codes. When dealing with collections of objects, there are often operations that are appropriate for both a single object and the entire collection. Think about Explorer in the vs code. When each item in the collection might itself contain collections of other objects, the use of the Composite pattern is appropriate. Composite is an easy way to represent tree-like collections without having to treat parent and leaf nodes differently. On the other hand, Command pattern is recognizable by behavioral methods in a sender which invokes a method in an implementation of a different receiver which has been encapsulated by the command implementation during its creation. Command classes are usually limited to specific actions.

There are several design patterns used in vs code source code. Just in explorer, we were able to identify few of them. Design patterns are solutions to recurring problems in software application development.