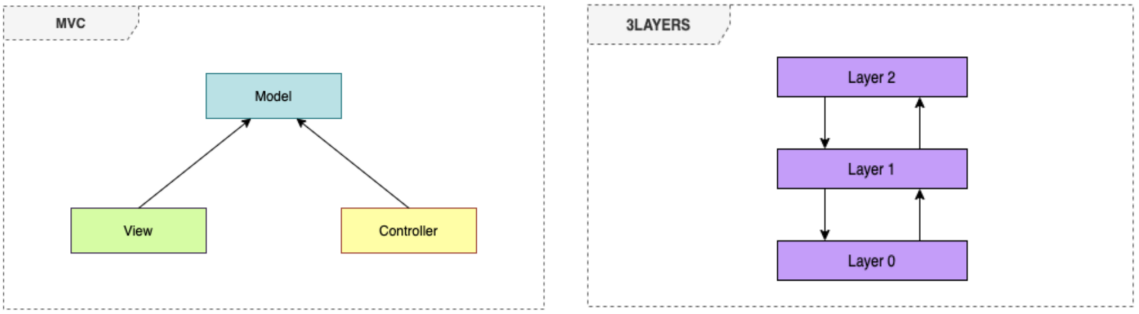
1)

a. MVC no es considerado un patrón de arquitectura 3TIER ya qué no especifica la distribución física de las capas (donde se ejecutan cada una de las 3), puede ocurrir que el modelo, la vista y el controlador estén en 3 capas distintas como no.

   MVC no es considerado un patrón de arquitectura 3LAYER ya que no cuenta con una separación lógica en capas y no cumple con la jerarquía de acceso secuencial (se aprecia mejor en los diagramas)



b. El componente más reusable es el modelo ya que encapsula la lógica de negocio, por lo tanto el mismo modelo puede usarse para distintos controladores o vistas. Por ej una aplicación web y su versión mobile tienen distintas vistas pero por detrás el mismo modelo.

c. Verdadero. Facade brinda una interfaz común para evitar acoplamientos innecesarios y ordenar los accesos al sistema.

d. Verdadero. Facade en sí es una nueva interfaz para el usuario. Inclusive para respetar el principio de segregación de interfaces se pueden llegar a implementar varios Facade.

e. Falso. Justamente se usa Facade para desacoplar al cliente del subsistema del cual controla el acceso. El cliente/usuario no sabe que hay detrás del Facade.

f. Generalmente con Facade se usa mucho el patrón Singleton debido a que se necesita solo un Facade y dicho patrón se asegura que una clase tenga solo una instancia y nos da un punto de acceso a ella.

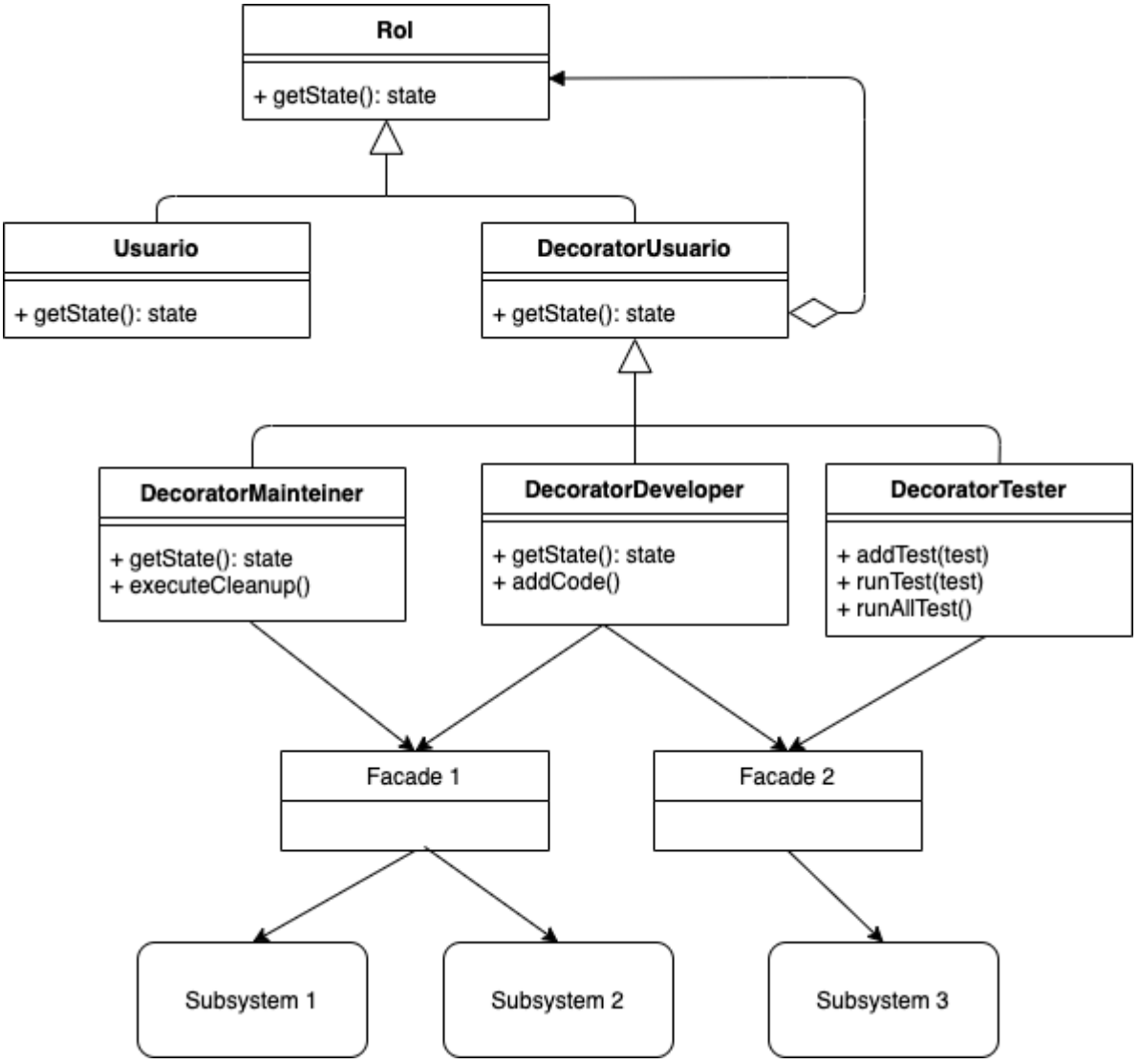
g. FACADE **X** ADAPTER MEDIADOR PROXY **X**

h. El método de actualización de la interfaz de Observadores incluye una referencia al objeto observado para que cuando a este le cambie el estado se ejecute la actualización de los observadores.

El observador tiene que saber que objeto está observando ya que cuando le llegue la actualización va a querer saber cual es el nuevo estado del objeto observado.

2)

Para empezar propondría una arquitectura de **MicroKernel** debido a que hay que interactuar con sistemas externos ya implementados y además tenemos que estar abiertos a cambios en el futuro, debido a que sabemos que está es la primera versión del sistema y que el próximo año va a salir una 2da con nuevas funcionalidades y roles.
Además para aislar el nuevo sistema de los ya existentes y para no estar acoplados a los mismos recomendaría usar el patrón **Facade**.
Por otro lado como menciona que se agregaron funcionalidades para nuevos roles me inclinaría por el patrón **Decorator** ya que este permite agregar nuevas comportamientos a los objetos.
Por ej si un usuario tiene el rol TESTING y el rol DEVELOPER y se agrega un nuevo rol MAINTAINER sería fácil de sumarlo a los ya existentes.



3)

Como se menciona al menos 2 integraciones con sistemas externos (SAT Resolver y Parser) y que el sistema evolucionará con frecuencia elegiría una arquitectura de **Microkernel** que es la que mejor se adapta a cambios (tanto en frecuencia como en dirección)

Debido a la necesidad de una interfaz de edición de texto y vistas gráficas para cada uno de los resultados de salida se propone **MVC.**

También como necesitamos distintos niveles de abstracción y reutilización de interfaces para los distintos pasos del proceso (construcción del modelo y relaciones, asignación y validación de reglas, validaciones consistencia lógica) se propone el patrón **Layer**