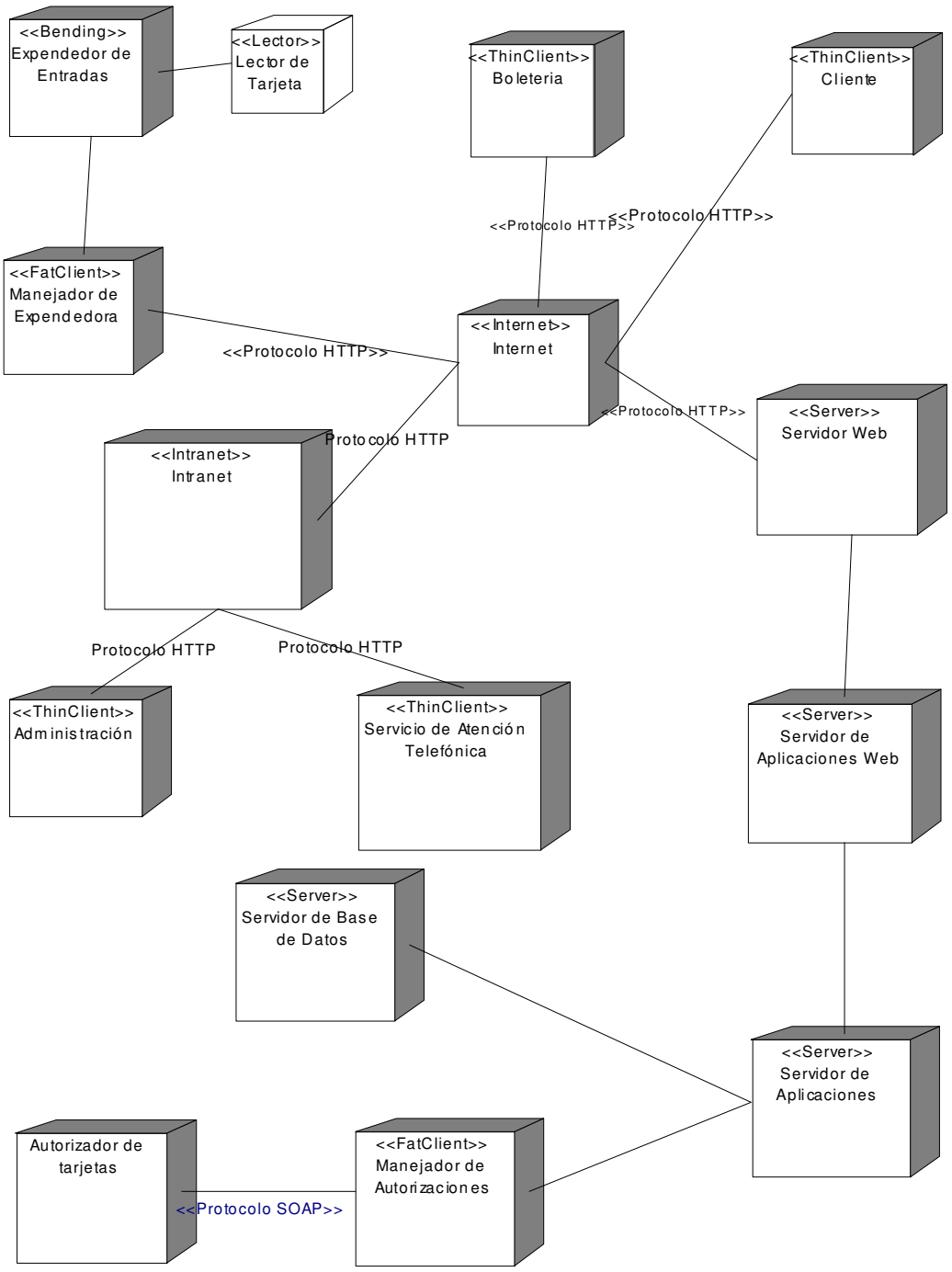
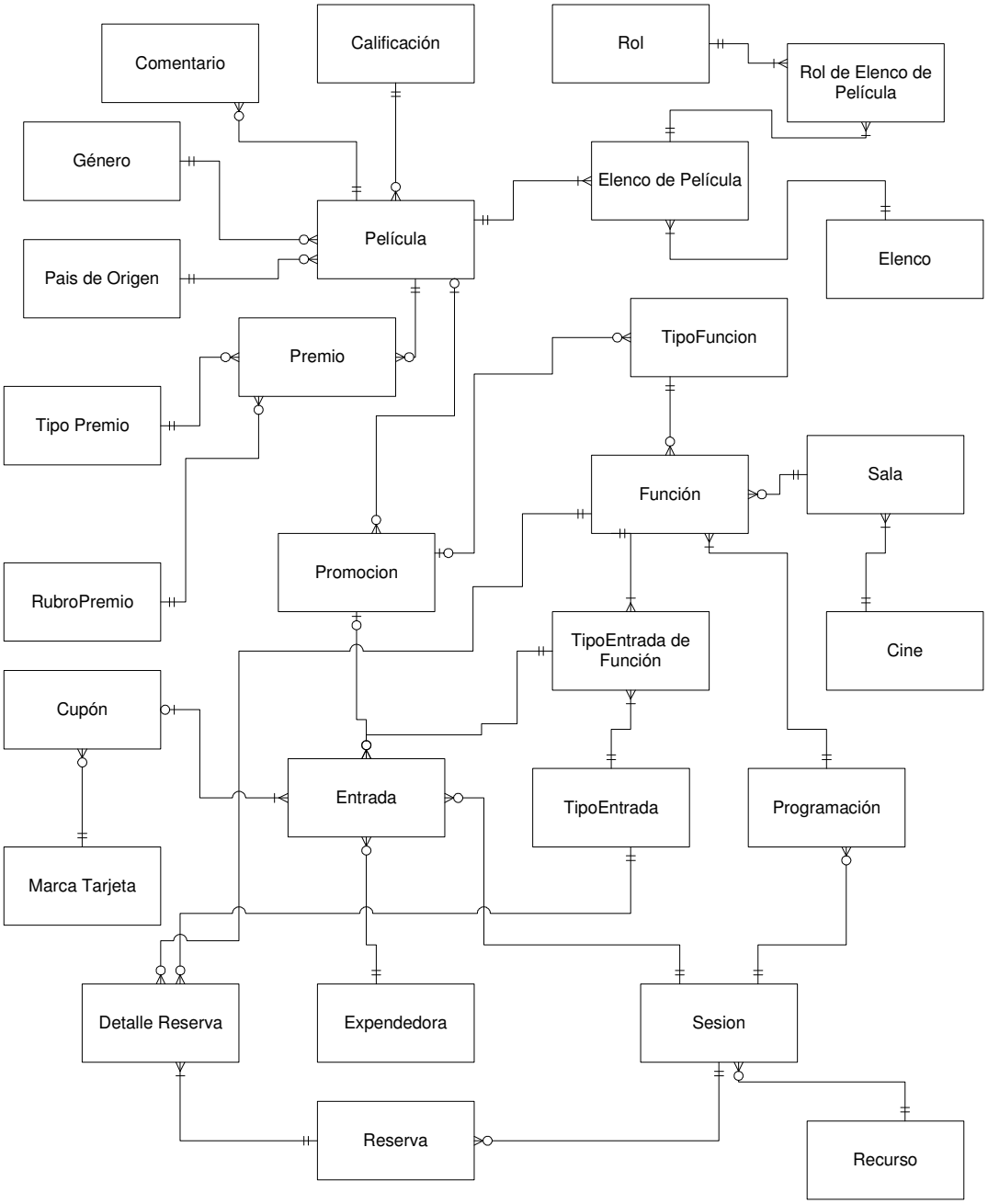
**Diagramas de Despliegue**



## Mapeo de Clases a Modelo Relacional



### Especificación de Atributos que se agregan al DER

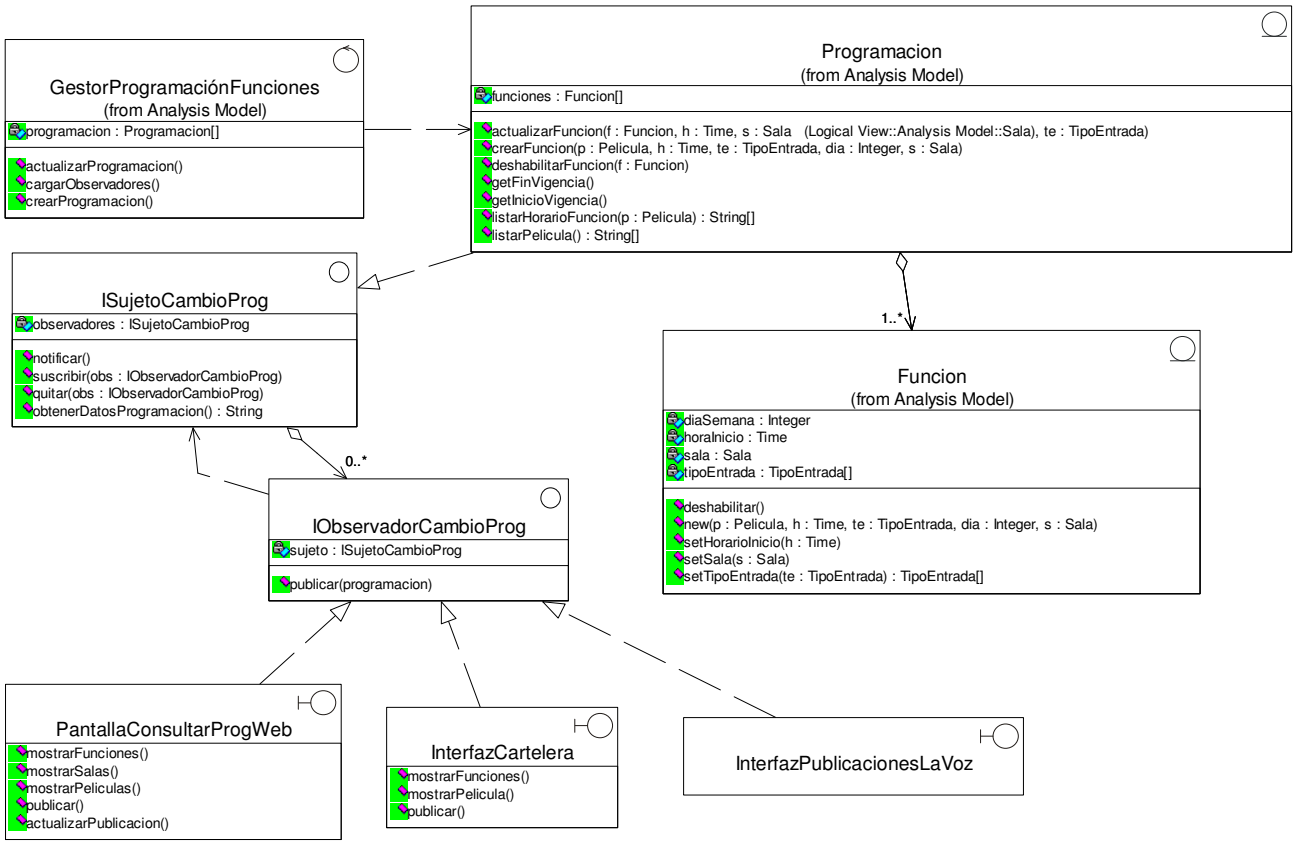
| Entidad | Atributos |
|---|---|
| **Genero** | @ID_genero |
| **Pais de Origen** | @ID_pais_de_origen |
| **Comentario** | @ID_comentario |
| **Calificacion** | @ID_calificación |
| **Rol** | @ID_rol |
| **Tipo Premio** | @ID_tipo_premio |
| **Rol** | @ID_rol |
| **Elenco** | @ID_elenco |
| **Cine** | @ID_cine |
| **Expendedora** | @ID_expendedora |
| **Detalle_reserva** | @ID_detalle_reserva<br>#ID_reserva<br>#ID_tipo_entrada<br>#ID_funcion |
| **Reserva** | @ numeroReserva<br>#ID_recurso |
| **Marca Tarjeta** | @ID_marca_tarjeta |
| **Rubro Premio** | @ID_rubro_premio |
| **Sala** | @numeroSala<br>#ID_cine |
| **Pelicula** | @ID_película<br>#ID_genero<br>#ID_pais_de_origen<br>#ID_calificación |
| **Rol de Elenco de Pelicula** | @ID_rol |
| **Elenco de Pelicula** | @ID_película<br>@ID_elenco |
| **Cupon** | @númeroCupon<br>#ID_marca_tarjeta<br>estado |
| **Premio** | @ID_premio<br>#ID_película<br>#ID_tipo_premio<br>#ID_rubro_premio |
| **Recurso** | @ID_recurso |
| **Tipo Entrada** | @ID_tipo_entrada |
| **Entrada** | @numeroTicket<br>#ID_tipo_entrada<br>#ID_cupon<br>#ID_expendedora<br>#ID_funcion<br>#ID_recurso<br>#ID_promocion |
| **Funcion** | @ID_funcion<br># ID-Peliula<br>#ID_programación<br>#ID_sala |
| **Tipo Entrada de Funcion** | @ID_funcion<br>@ID_tipo_entrada<br>#ID_funcion |
| **Programacion** | @ID_programación<br>#ID_recurso |
| **Sesion** | @ ID_Sesion<br>#ID_recurso |
| **Promocion** | @ ID_promocion<br># ID-Pelicula<br># ID-Tipo_Funcion |
| **TipoFuncion** | @ ID-Tipo_Funcion |

**Aspectos de Diseño a resolver con los patrones de GAMMA**

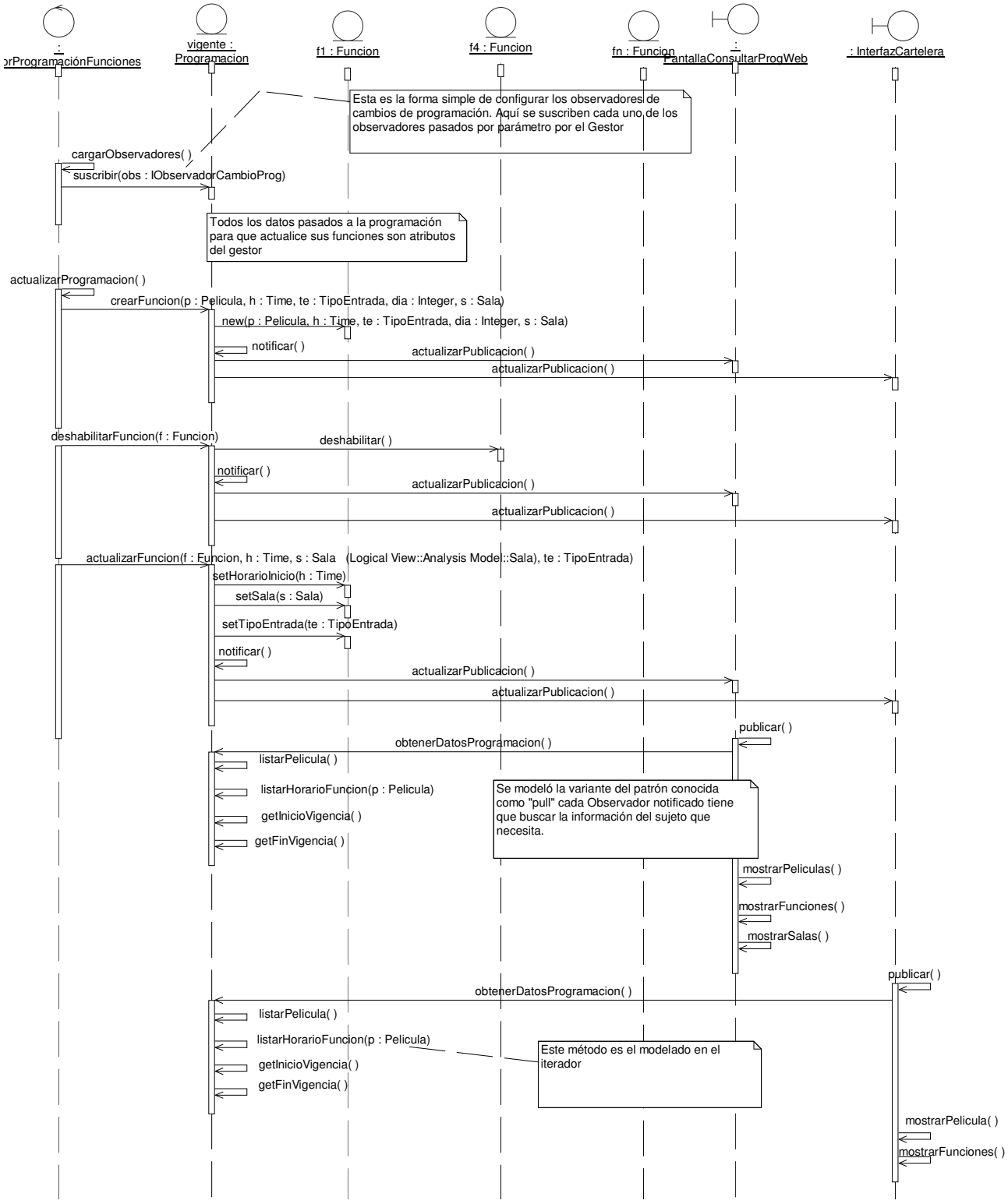
Resolver cómo se realizará la actualización de los cambios en la programación vigente del complejo cuando se esté consultando la misma en tiempo real. Considerar las distintas páginas Web que las publican (incluidas las que no pertenecen al complejo) y la cartelera electrónica de cada cine.

Patrón a aplicar **OBSERVER**

**Estructura**



## Dinámica


: orProgramaciónFunciones
vigente : Programacion
f1 : Funcion
f4 : Funcion
fn : Funcion
: PantallaConsultarProgWeb
: InterfazCartelera
Esta es la forma simple de configurar los observadores de cambios de programación. Aquí se suscriben cada uno de los observadores pasados por parámetro por el Gestor
cargarObservadores( )
suscribir(obs : IObservadorCambioProg)
Todos los datos pasados a la programación para que actualice sus funciones son atributos del gestor
actualizarProgramacion( )
crearFuncion(p : Pelicula, h : Time, te : TipoEntrada, dia : Integer, s : Sala)
new(p : Pelicula, h : Time, te : TipoEntrada, dia : Integer, s : Sala)
notificar( )
actualizarPublicacion( )
actualizarPublicacion( )
deshabilitarFuncion(f : Funcion)
deshabilitar( )
notificar( )
actualizarPublicacion( )
actualizarPublicacion( )
actualizarFuncion(f : Funcion, h : Time, s : Sala (Logical View::Analysis Model::Sala), te : TipoEntrada)
setHorarioInicio(h : Time)
setSala(s : Sala)
setTipoEntrada(te : TipoEntrada)
notificar( )
actualizarPublicacion( )
actualizarPublicacion( )
publicar( )
obtenerDatosProgramacion( )
listarPelicula( )
listarHorarioFuncion(p : Pelicula)
getInicioVigencia( )
getFinVigencia( )
Se modeló la variante del patrón conocida como "pull" cada Observador notificado tiene que buscar la información del sujeto que necesita.
mostrarPeliculas( )
mostrarFunciones( )
mostrarSalas( )
publicar( )
obtenerDatosProgramacion( )
listarPelicula( )
listarHorarioFuncion(p : Pelicula)
getInicioVigencia( )
getFinVigencia( )
Este método es el modelado en el iterador
mostrarPelicula( )
mostrarFunciones( )

**Implementación:**

NOTA: Como en JAVA no se puede especificar en la Interfaces atributos instanciables, se deben implementar las interfaces como clases con métodos abstractos para que deban ser implementados en cada subclase como si fuera realmente una interface.

```
//Participante: "Sujeto"
class ISujetoCambioProg{
//atributos
        private ArrayList observadores; //gestiona un Vector de IObservadorCambioProg


//métodos
        public abstract void suscribir(IObservadorCambioProg obs);

        public abstract  void quitar(IObservadorCambioProg obs);

        public abstract  String()() obtenerDatosProgramacion();

        public abstract  void notificar();

}




//Participante: "Observador"
class IObservadorCambioProg{
//atributos
        private ISujetoCambioProg sujeto;


//métodos
        public abstract void  actualizarPublicacion();
}




//Participante: "Sujeto"
public class Programacion() extends ISujetoCambioProg{
//atributos
        private Date fechaInicioProgramacion;
        private Date fechaFinProgramacion;
        private ArrayList funciones;


//métodos
        ...

        public void crearFuncion(Pelicula p, Date h, TipoEntrada te, Integer diaSemana, Sala s){
                Funcion f=new Funcion(p, h, te, diaSemana, s);
                funciones.add(f);

                //notifica los cambios realizados en la Programacion
```

```
        notificar();
}


public void deshabilitarFuncion(Funcion f){
        f.deshabilitar();

        //notifica los cambios realizados en la Programacion
        notificar();
}


public void actualizarFuncion(Funcion f, Date h, Sala s, TipoEntrada ()te){
        f.setHorarioInicio(h);
        f.setSala(s);
        f.setTipoEntrada(te);

        //notifica los cambios realizados en la Programacion
        notificar();
}


public void suscribir(IObservadorCambioProg obs){
        observadores.add(obs);
}


public void quitar(IObservadorCambioProg obs){
        observadores.remove(obs);
}

public String()() obtenerDatosProgramacion(){

        String peliculas=listarPelicula();

        String ()()datos=new String(peliculas.lenght)(5); //aqui terminar

        for (int i=0; i < peliculas.lenght; i++){
                datos(i)=peliculas(i);

                listarHorarioFuncion( peliculas );
        }


}


public notificar(){
        ListIterator iter=observadores.listIterator();
        IObservadorCambioProg obs;

        //Invoca el actualizarPublicacion() de cada observador
        while (iter.hasNext()){
                obs=(IObservadorCambioProg) iter.next();
                obs.actualizarPublicacion();
        }
}
```

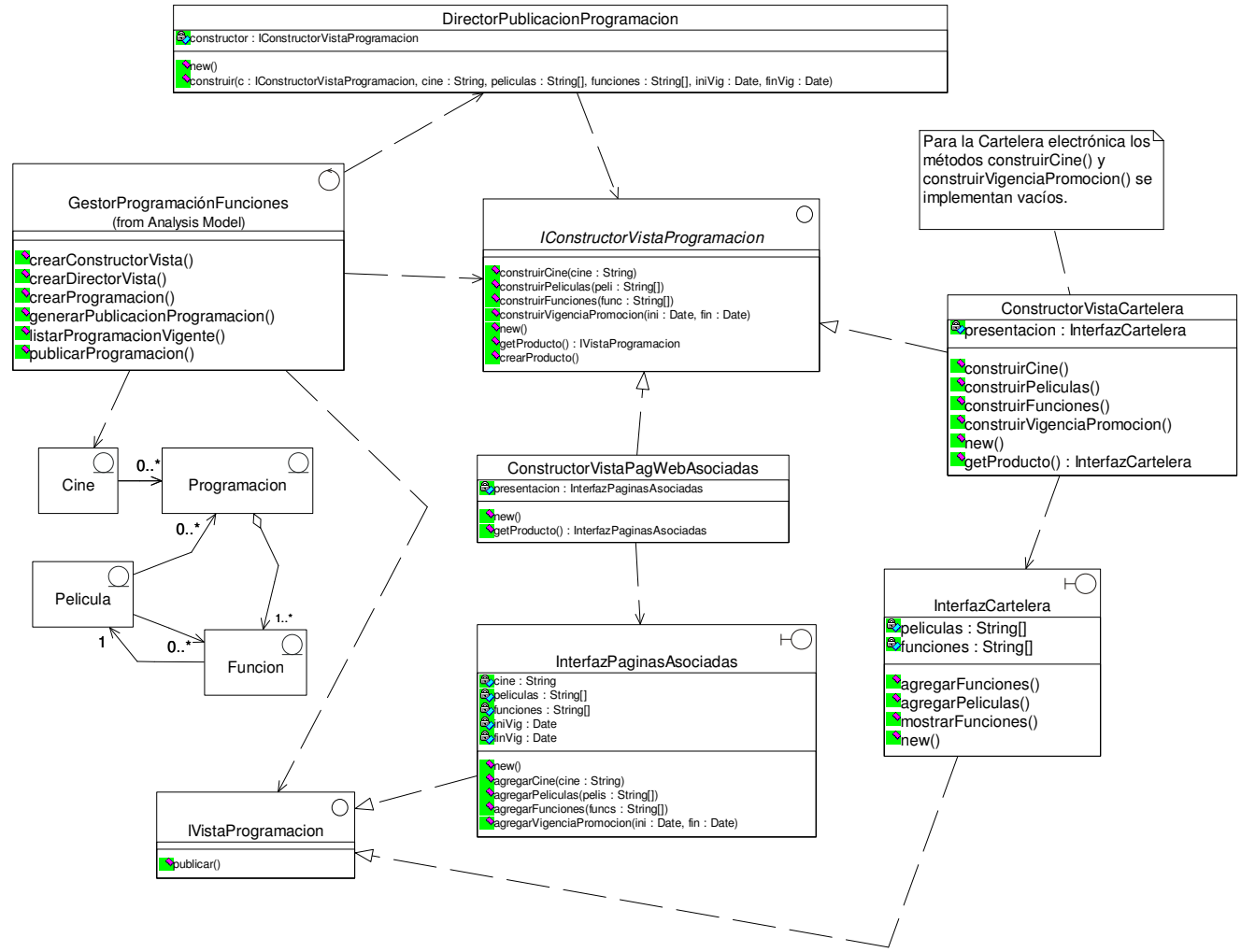
```
	public String ()listarPelicula(){
		ListIterator iter=funciones.listIterator();
		Funcion f;
		String ()peliculas=new String(funciones.size());
		int i=0;

		while (iter.hasNext()){
			f=(Funcion) iter.next();
			peliculas=f.getPelicula().getNombre();
			i++;
		}
		return peliculas;
	}



	public String ()listarHorarioFuncion(Pelicula p){
		ListIterator iter=funciones.listIterator();
		Funcion f;
		String ()horarios=new String(funciones.size());
		int i=0;
		while (iter.hasNext()){
			f=(Funcion) iter.next();
			if ( f.getPelicula()==p ) {
				horarios(i)=f.getFuncion().toString();
				i++;
			}
		}
		return horarios;
	}



	public Date getInicioVigencia(){
		return fechaInicioProgramacion;
	}


	public Date getFinVigencia(){
		return fechaFinProgramacion;
	}

	...
}



//Participante: "Observador Concreto"
public class InterfazCartelera extends IObservadorCambioProg{
//atributos
	private boolean refrescar;

//métodos
	...
```

```
	public void  actualizarPublicacion(){
		refrescar=true;
	}


	public void publicar(){
		if ( refrescar==true) {
			String ()()datos=sujeto.obtenerDatosProgramacion();

			//código para refrescar la cartelera con la programacion actualizada
		}

	}

	...
}




public class GestorProgramaciónFunciones{
//atributos
	private Programacion programacion;


//métodos
	...
	private void cargarObservadores(){
		//cargar los posibles observadores
		programacion.suscribir(observadores);
	}


	public void actualizarProgramacion(){
		//se ejecuta la funcionalidad para agregar nuevas Funciones
		programacion.crearFuncion(pelicula, hora, tiposEntrada, diaSemana, sala);


		//se ejecuta la funcionalidad para quitar Funciones
		programacion.deshabilitarFuncion(funcion);


		//se ejecuta la funcionalidad para actualizar las características de las Funciones
		programacion.actualizarFuncion(funcion, hora, tiposEntrada, sala);
	}

	...

}
```

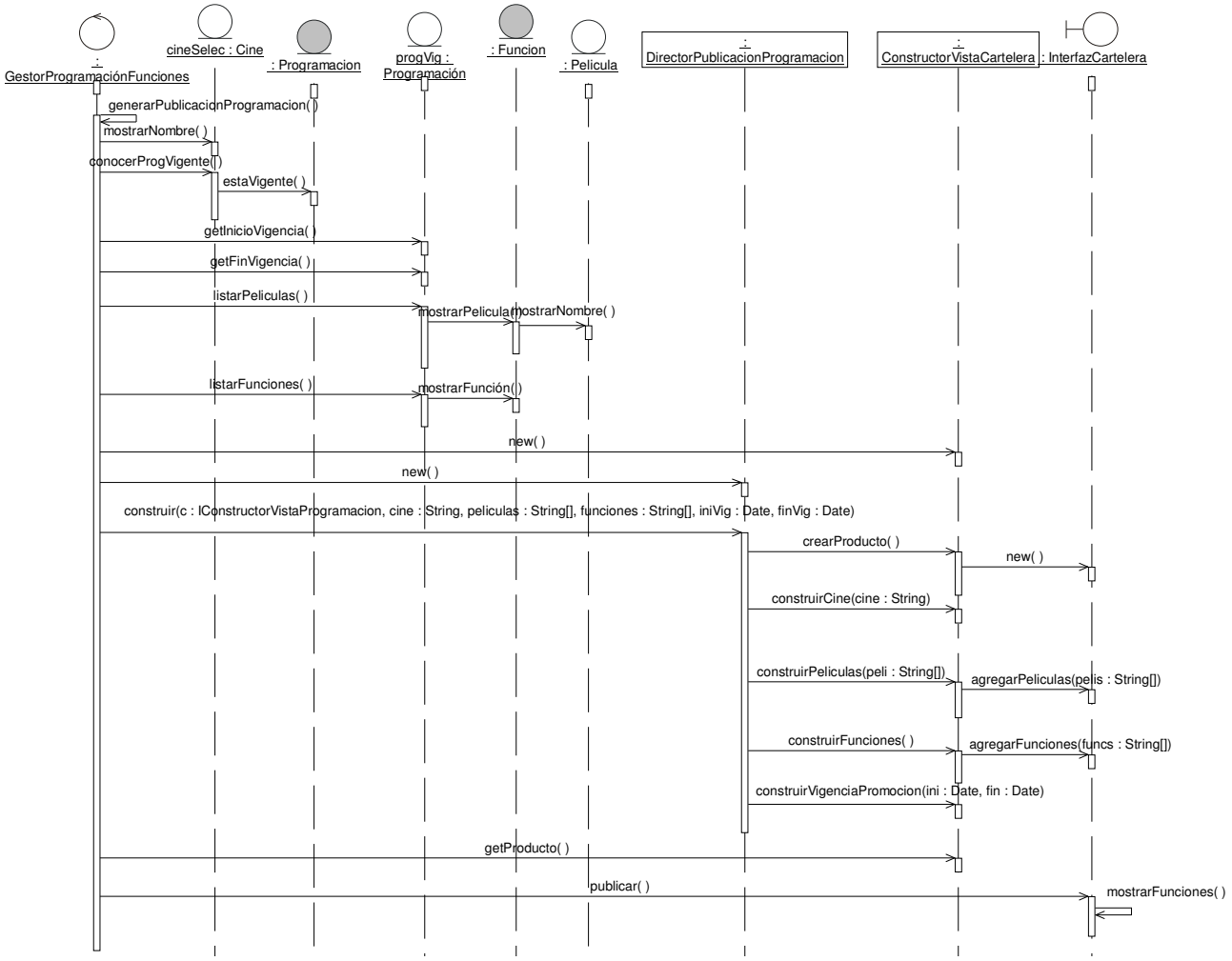
Establecer un proceso de creación de los diferentes medios para la publicación de la programación vigente de cada cine y de sus funciones. Los medios de publicación a considerar son la cartelera electrónica y las páginas Web asociadas.
Nota: modelar sólo la dinámica para la publicación de la programación en la cartelera.

Patrón a aplicar **BUILDER**

**Estructura**



### Dinámica


: GestorProgramaciónFunciones
cineSelec : Cine
: Programacion
progVig : Programación
: Funcion
: Pelicula
: DirectorPublicacionProgramacion
: ConstructorVistaCartelera
: InterfazCartelera
generarPublicacionProgramacion( )
mostrarNombre( )
conocerProgVigente( )
estaVigente( )
getInicioVigencia( )
getFinVigencia( )
listarPeliculas( )
mostrarPelicula( )
mostrarNombre( )
listarFunciones( )
mostrarFunción( )
new( )
new( )
construir(c : IConstructorVistaProgramacion, cine : String, peliculas : String[], funciones : String[], iniVig : Date, finVig : Date)
crearProducto( )
new( )
construirCine(cine : String)
construirPeliculas(peli : String[])
agregarPeliculas(pelis : String[])
construirFunciones( )
agregarFunciones(funcs : String[])
construirVigenciaPromocion(ini : Date, fin : Date)
getProducto( )
publicar( )
mostrarFunciones( )

Implementación

```
//Participante: "Cliente"
publicar class GestorProgramaciónFunciones {


        public void generarPublicacionProgramacion(){

        //NOTA: se omite la parte de consulta de la programación vigente para facilitar el entendimiento
del patrón, ya que la misma no forma parte de éste.
        //Aquí asignaría el valor a los atributos cine, peliculas, funciones, iniVig y finVig


        //se define el Constructor a Utilizar
        IConstructorVistaProgramacion constructor=new ConstructorVistaCartelera();

        //se crea el Director
        DirectorPublicacionProgramacion director=new DirectorPublicacionProgramacion();

        //se le pide al Director que inicie el proceso de creación
        director.construir(constructor, cine, peliculas, funciones, iniVig, finVig);


        //se recupera el producto
        IConstructorVistaProgramacion producto=constructor.getProducto();

        //el Gestor usar el Producto para el fin que lo construyó
        producto.publicar();

        }

}



//Participante: "Director"
public class DirectorPublicacionProgramacion{


        public DirectorPublicacionProgramacion(){

        }


        public void construir(c : IConstructorVistaProgramacion, cine : String, peliculas : String(), funciones :
String(), iniVig : Date, finVig : Date){

                //se construye el Producto paso a paso
                c.crearProducto();

                c.construirCine(cine);

                c.construirPeliculas(peliculas);

                c.construirFunciones(funciones);

                c.construirVigenciaPromocion(iniVig, finVig);
```

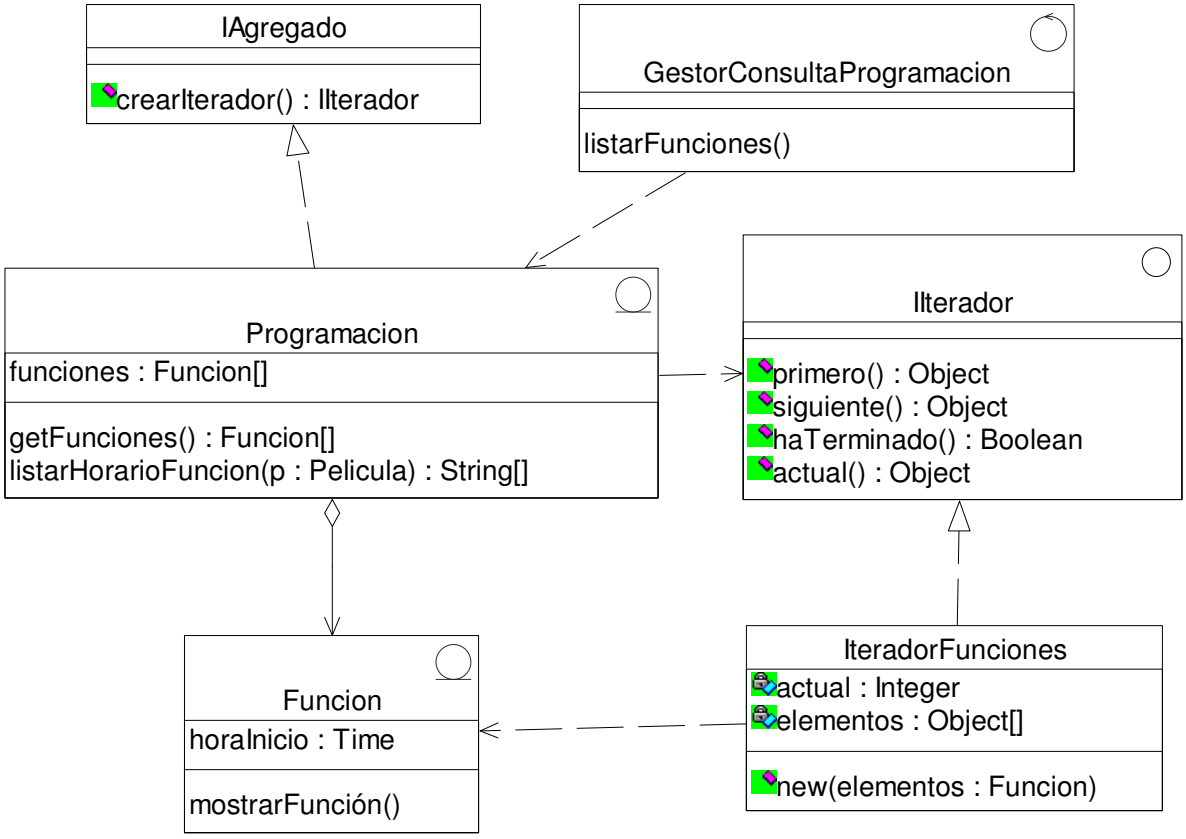
```
        }

}



//Participante: "Constructor Abstracto"
interface IConstructorVistaProgramacion{

        public void crearProducto();
        public void construirCine(cine : String);
        public void construirPeliculas(peli : String());
        public void construirFunciones(func : String()());
        public void construirVigenciaPromocion(ini : Date, fin : Date);
        public IVistaProgramacion getProducto();
}



//Participante: "Constructor Concreto"
public class ConstructorVistaCartelera implements IConstructorVistaProgramacion{
//atributos
        private InterfazCartelera presentacion;


//métodos
        public ConstructorVistaCartelera(){}

        public void crearProducto(){
                presentacion=new InterfazCartelera();
        }

        public void construirCine(cine : String){
                //se implementa vacío, ya que la Cartelera no tiene para mostrar el cine
        }

        public void construirPeliculas(peli : String()){
                presentacion.agregarPeliculas(peli);
        }

        public void construirFunciones(func : String()()){
                presentacion.agregarFunciones(func);
        }

        public void construirVigenciaPromocion(ini : Date, fin : Date){
                //se implementa vacío, ya que la Cartelera no tiene para mostrar la vigencia
        }

        public IVistaProgramacion getProducto(){
                return presentacion;
        }

}
```

```
interface IVistaProgramacion{

	public void publicar();
}



//Participante: "Producto"
public class InterfazCartelera implements IVistaProgramacion {
//atributos
	private String() peliculas;
	private String()() funciones;

//métodos
	public InterfazCartelera(){
	}



	public void agregarPeliculas(pelis : String){
		peliculas=pelis;
	}


	public void agregarFunciones(funcs : String()()){
		funciones=funcs;
	}


	public void publicar(){
		mostrarFunciones();
	}


	private void mostrarFunciones(){
		//aquí estará el código que actualizará las películas y funciones en la Cartelera
	}
}
```

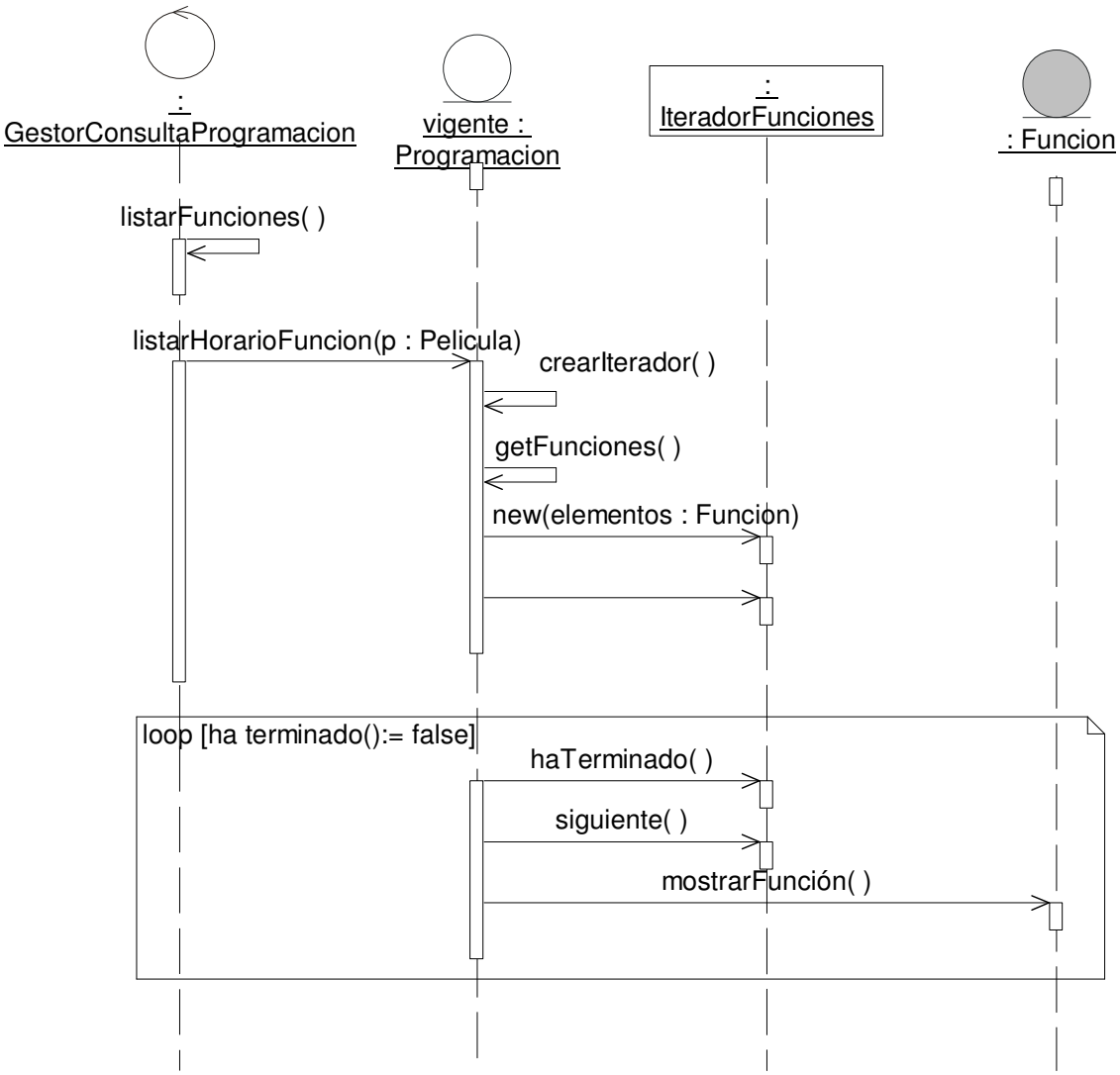
Considerar una manera de recorrer las funciones de una programación cuando se realiza la consulta de la misma.

**Patrón a Aplicar: Iterator**

**Estructura**



**Dinámica**


:
GestorConsultaProgramacion
vigente :
Programacion
:
IteradorFunciones
: Funcion
listarFunciones( )
listarHorarioFuncion(p : Pelicula)
crearIterador( )
getFunciones( )
new(elementos : Funcion)
loop [ha terminado():= false]
haTerminado( )
siguiente( )
mostrarFunción( )

**Implementación:**

```
//Participante: "Agregado"
interface IAgregado{
	public IIterador crearIterador();
}


//Participante: "Iterador"
interface IIterador{
	public Object primero();
	public Object siguiente();
	public Object actual();
	public boolean haTerminado();
}




//Participante: "Agregado Concreto"
class Programacion implements IAgregado{

//atributos
	private Funcion []funciones;


//métodos
	public IIterador crearIterador(){
		return new IteradorFuncion(funciones);
	}



	public String []listarHorarioFuncion(){
		Funcion []funcs=getFunciones();
		String []horaFuncs=new String[funcs.size()];
		int i=0;

		IIterador iter=crearIterador();
		Funcion f;

		while (iter.haTerminado()==false){
			f=(Funcion) iter.siguiente();
			horasFuncs[i]=f.mostrarFuncion().toString();
			i++;
		}
		return horaFuncs;
	}

	public Funcion []getFunciones(){
		return funciones;
```

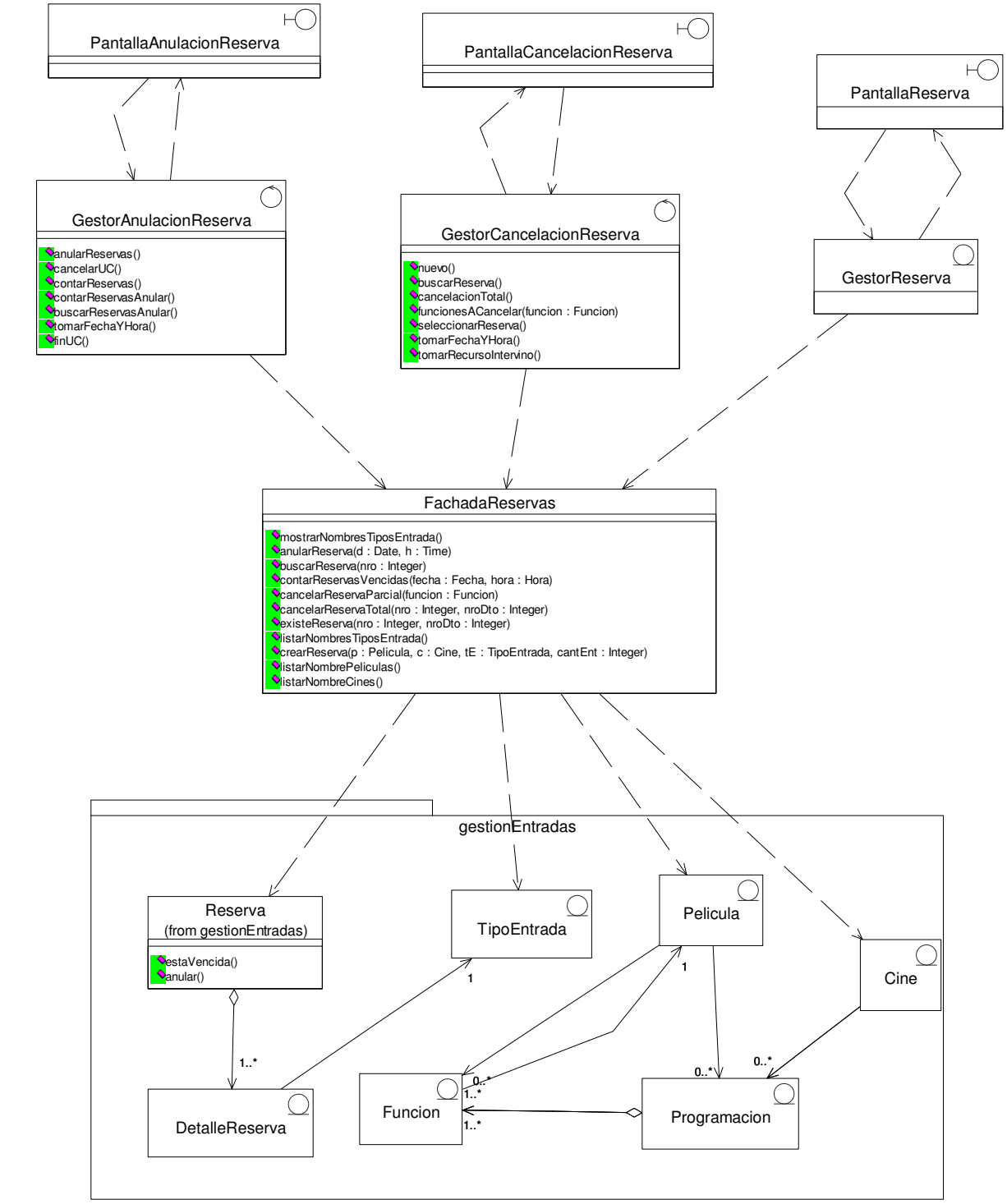
```
        }

}



//Participante: "Iterador Concreto"
class IteradorFunciones implements IIterador{

//atributos
        private int actual;
        private Object []elementos;


//metodos
        public new (Funcion []elementos){
                this.elementos=elementos;
        }


        public Object primero()
        {
                return elementos[0];
        }


        public Object siguiente(){
                actual=actual+1;
                return elementos[actual];
        }

        public Object actual(){
                return elementos[actual];
        }


        public boolean haTerminado()
        {
                boolean rta;
                if ( elementos.size()==actual+1 ) rta=true;
                else rta=false;
                return rta;
        }

}
```

```
public class Funcion{

//atributos
        private Date funcion;
        private Sala sala;
        private Estado estado;
        private Pelicula pelicula;
        ...


//métodos
        ...

        public Date mostrarFuncion(){
                return funcion;
        }

}
```

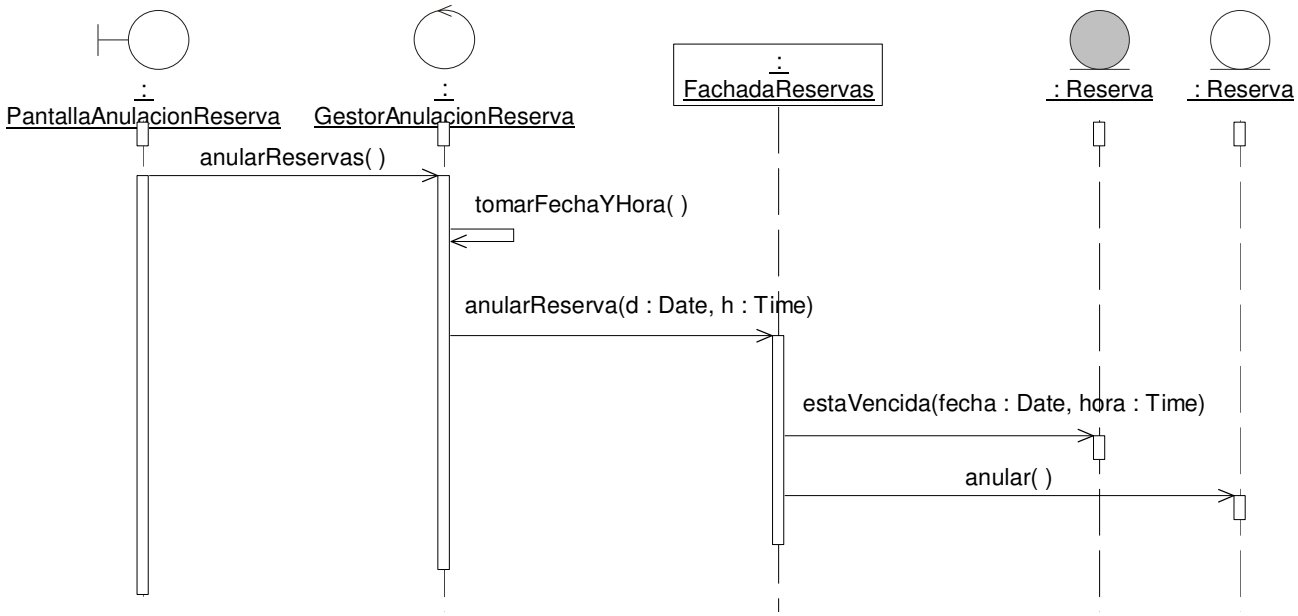
Considerar la manera de tener un acceso controlado a los servicios que brinda el subsistema de reservas, en lo que se refiere a la generación, anulación y cancelación de reservas.
Nota: modelar sólo la dinámica de Anulación de Reservas.

**Patrón a Aplicar: Façade**

**Estructura**



**Dinámica**




**Implementación:**

```
public class GestorAnulacionReserva{

//métodos
...
        public void anularReservas(){
                FachadaReserva fachada=new FachadaReserva();

                fachada.anularReserva( tomarFechaYHora() );
        }


        private Date tomarFechaYHora() {
           //determina la fecha y hora actual de la PC
           return new Date(System.getTime())
        }

...
}
```

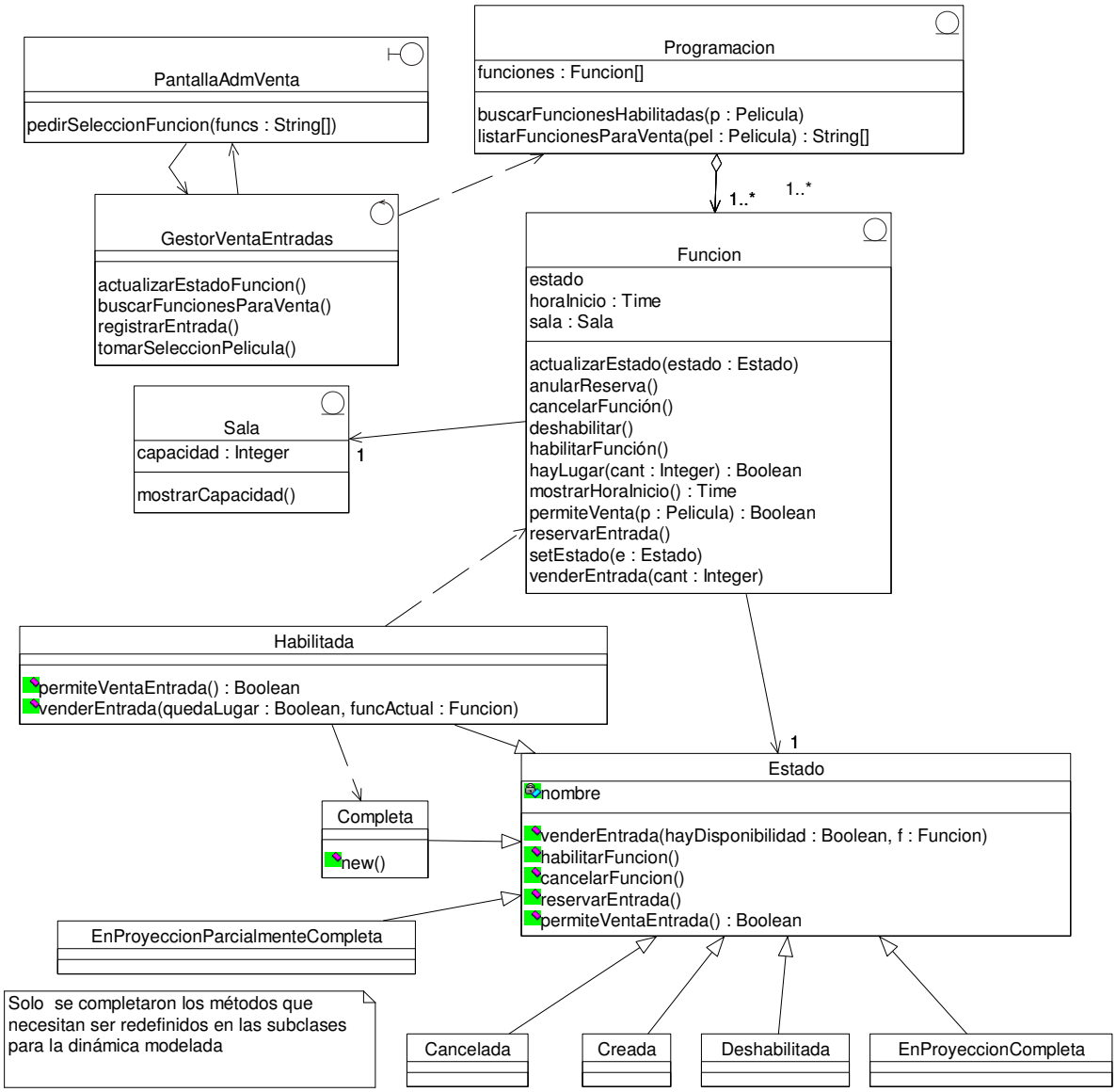
```
//Participante: "Fachada"
public class FachadaReserva{
```

```
  public void anularReserva(Date fechaYHoraActual ){

        //Lee las reservas del Esquema de Persistencia
        Reserva ()reservas=Cache.leerObjectos(Reserva);

        Reserva r;

        for (int i=0; i < reservas.lenght; i++ ) {
              r=reservas(i);

              //si la reserva está Vencida, se anula a la misma
              if ( r.estaVencida( fechaYHoraActual )==true) {
                    r.anularReserva( fechaYHoraActual );
              }
        }

  }


        public String ()listarNombresTiposEntrada(){
        //no se implementó
        }

        public Reserva buscarReserva (int nro){
        //no se implementó
        }


        public void cancelarReservaParcial( funcion : Funcion ){
        //no se implementó
        }

        public void cancelarReservaTotal( int nro ){
        //no se implementó
        }

        public boolean          existeReserva ( int nro ){
        //no se implementó
        }


        public void crearReserva (Pelicula p, Cine c, TipoEntrada ()tipoEnt, int ()cantEnt ){
        //no se implementó
        }

        public String ()listarNombresPeliculas(){
        //no se implementó
        }

        public String ()listarNombresCine(){
        //no se implementó
        }

}
```

```
public class Reserva(){
//atributos
        private Date fechaVigencia;
...



//métodos
        ...

        public boolean estaVencida(Date d) {

                if ( fechaVigencia.getTime() <= d.getTime() ) return true;

                return false;
        }
        ...

}
```

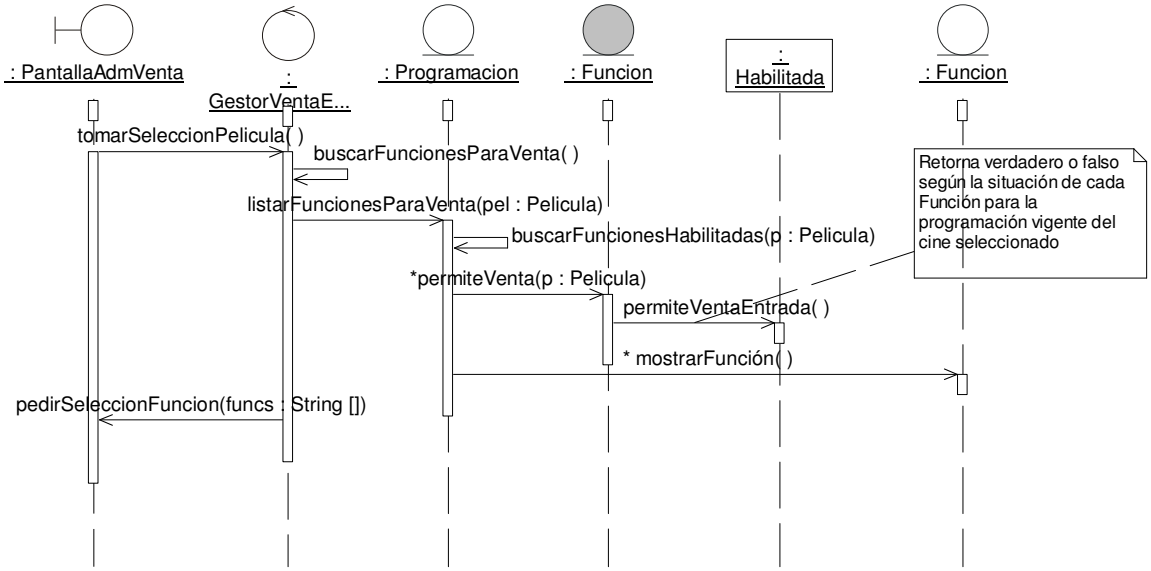
Resolver cómo debe comportarse la **función** de acuerdo al momento en que se encuentre la misma.
Nota: modelar la dinámica implicada en el contexto del C-U "Registrar Venta de Entradas".

**Patrón a Aplicar: State**
**Estructura**

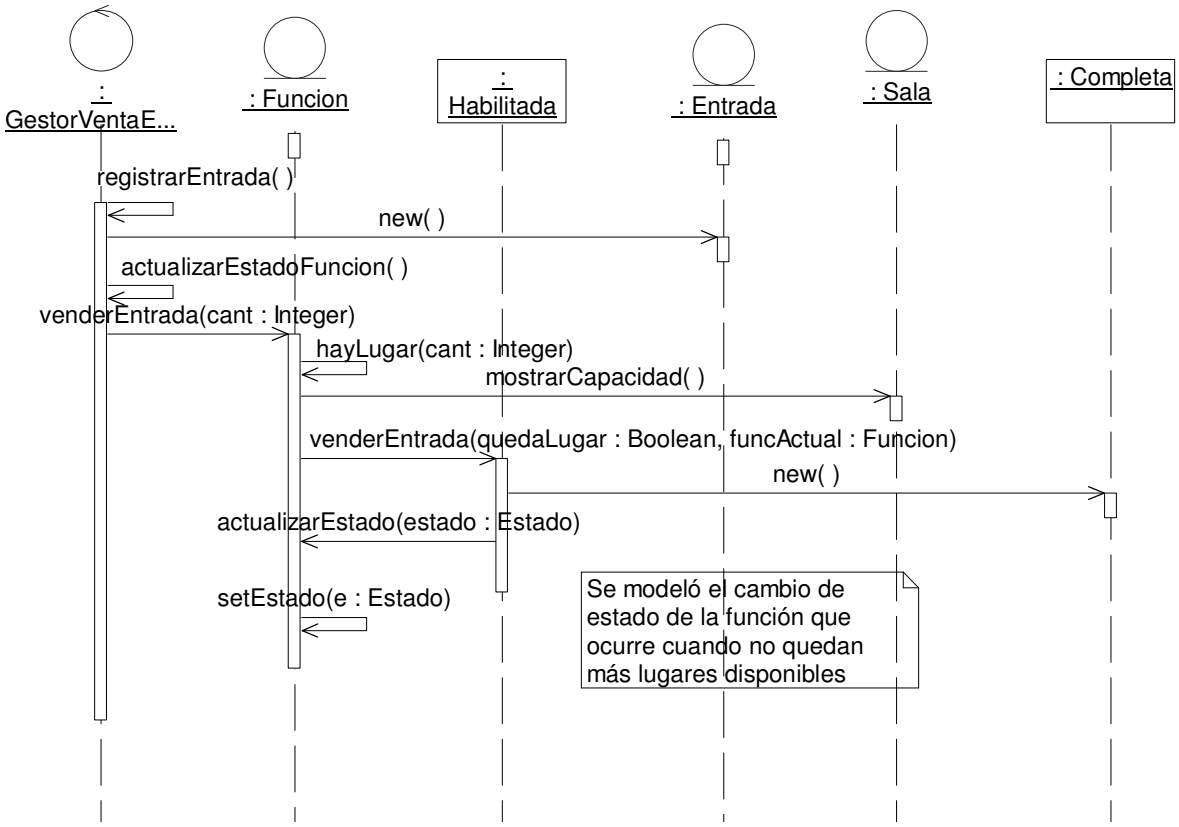


**Dinámica**

**Escenario de Venta con función habilitada y con disponibilidad de lugares**




**Escenario de Venta con función habilitada y con disponibilidad de lugares, quedando completa la función al momento de la venta.**



Implementación:

```
public class GestorVentaEntrada{

//atributos
        private Entrada entrada;
        private Funcion funcion;
        private int cantidad;
        private Programacion progVigente;
        private Pelicula pelicula;
        private PantallaAdmVenta pantalla;
        ...

//métodos
        ...

        public void tomarSeleccionPelicula(){
                String funcs=buscarFuncionesParaVenta();

                pantalla.pedirSeleccionFuncion(funcs);
        }


        public String ()buscarFuncionesParaVenta(){
                return progVigente.listarFuncionesParaVenta(pelicula);
        }

        public void registrarEntrada(){
                entrada= new Entrada();

                actualizarEstadoFuncion();

        }


        private void actualizarEstadoFuncion(){
                funcion.venderEntrada(cantidad);

        }

        ...
}



public class Programacion(){
//atributos
        private Funcion ()funciones;
        ...

//métodos
        ...
        public String() listarFuncionesParaVenta(Pelicula pel){
```

```
            ArrayList lista=buscarFuncionesHabilitadas(pel);
            //crea un vector de String para mostrar los distintos horarios
            String ()funcs=new String(lista.size());

            ListIterator iter=lista.listIterator();
            for (int i=0; iter.hasNext(); i++){
                  funcs(i)=((Funcion) iter.next()).mostrarFuncion();
            }

            return funcs;
      }



      public ArrayList buscarFuncionesHabilitadas(Pelicula p){
            ArrayList lista=new ArrayList();
            for (int i=0; i < funciones.lenght; i++){

                  if ( funciones(i).permiteVenta(p)==true ){
                        //si la funcion permite la Venta
                        lista.add(funciones(i));
                  }

            }
            return lista;
      }

      ...
}



//Participanete: Cliente
public class Funcion{

//atributos
      private Sala sala;
      private Estado estado;
      private Pelicula pelicula;
      ...


//métodos
      ...

      public boolean permiteVenta(Pelicula p){
            if (pelicula==p){ //verifica que la funcion corresponda a la pelicula
                  return estado.permiteVentaEntrada(); //pregunta al estado si permite la venta
            }
            return false; //la funcion no es de la pelicula
      }



      public void venderEntrada(int cant){
            boolean disponible=hayLugar(cant);
            estado.venderEntrada(disponible, this);
```

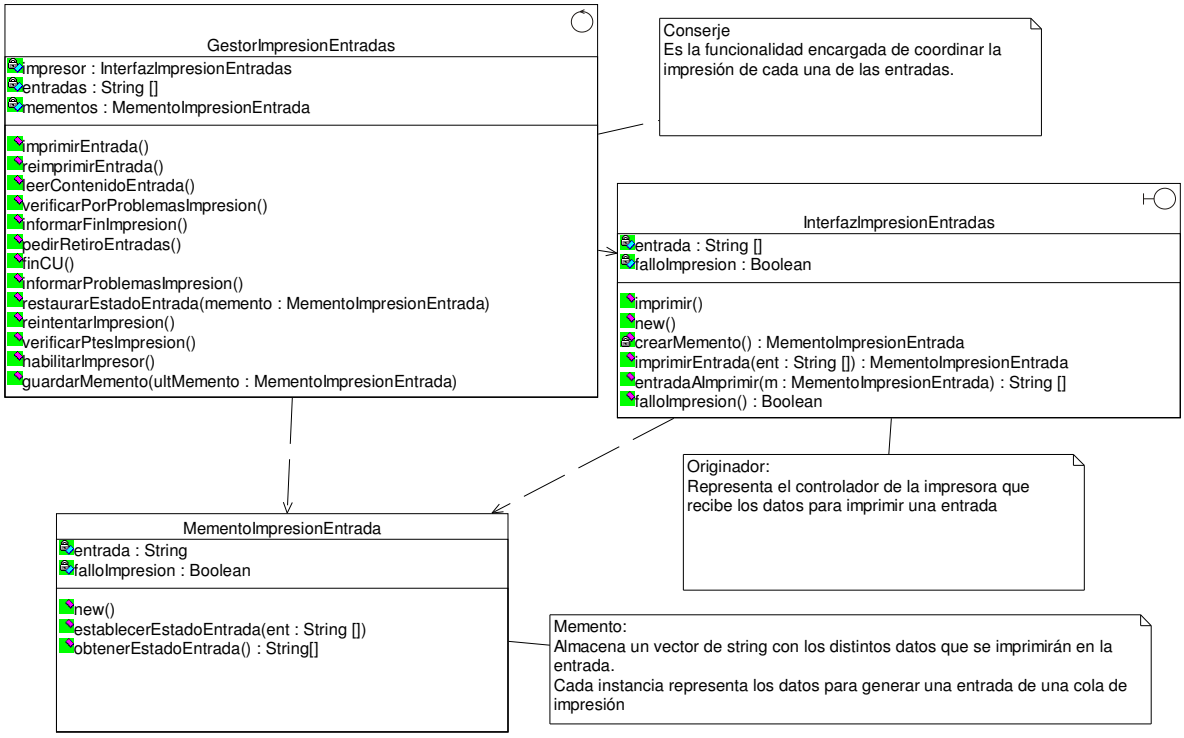
```
	}


	public boolean hayLugar(int cant){
		if ( cant < sala.mostrarCapacidad() ) return true;
		return false;

	}


	public void actualizarEstado(Estado estado){
		setEstado(estado);
	}

	private void setEstado(Estado e){
		estado=e;
	}

}



//Participante: Estado
public abstract class Estado{
//atributos
	private String nombre;

//métodos

	public Estado(String nom){
		nombre=nom;
	}

	public void venderEntrada(boolean hayDisponib, Funcion f){;}

	public void reservarEntrada(){;}

	public void cancelarFuncion(){;}

	public void habilitarFuncion(){;}

	public void permiteVentaEntrada(){
		//por defecto siempre responde false, a menos que esté redefinida en las clases
		//de los Estados que si permiten la venta de entradas
		return false;
	}
}



//Participante: Estado Concreto
public class Habilitada extends Estado{


	public Habilitada(){
		super.nombre="Habilitada";
```

```
	}


	public boolean permiteVentaEntrada(){
		return true;
	}


	public void venderEntrada(boolean quedaLugar, Funcion funcActual){
		if ( quedaLugar==false){
			Estado estado=new Completa();
			funcActual.actualizarEstado(estado);
		}
	}

}


//Participante: Estado Concreto
public class Completa extends Estado{

	public Completa(){
		super.nombre="Completa";
	}

}
```

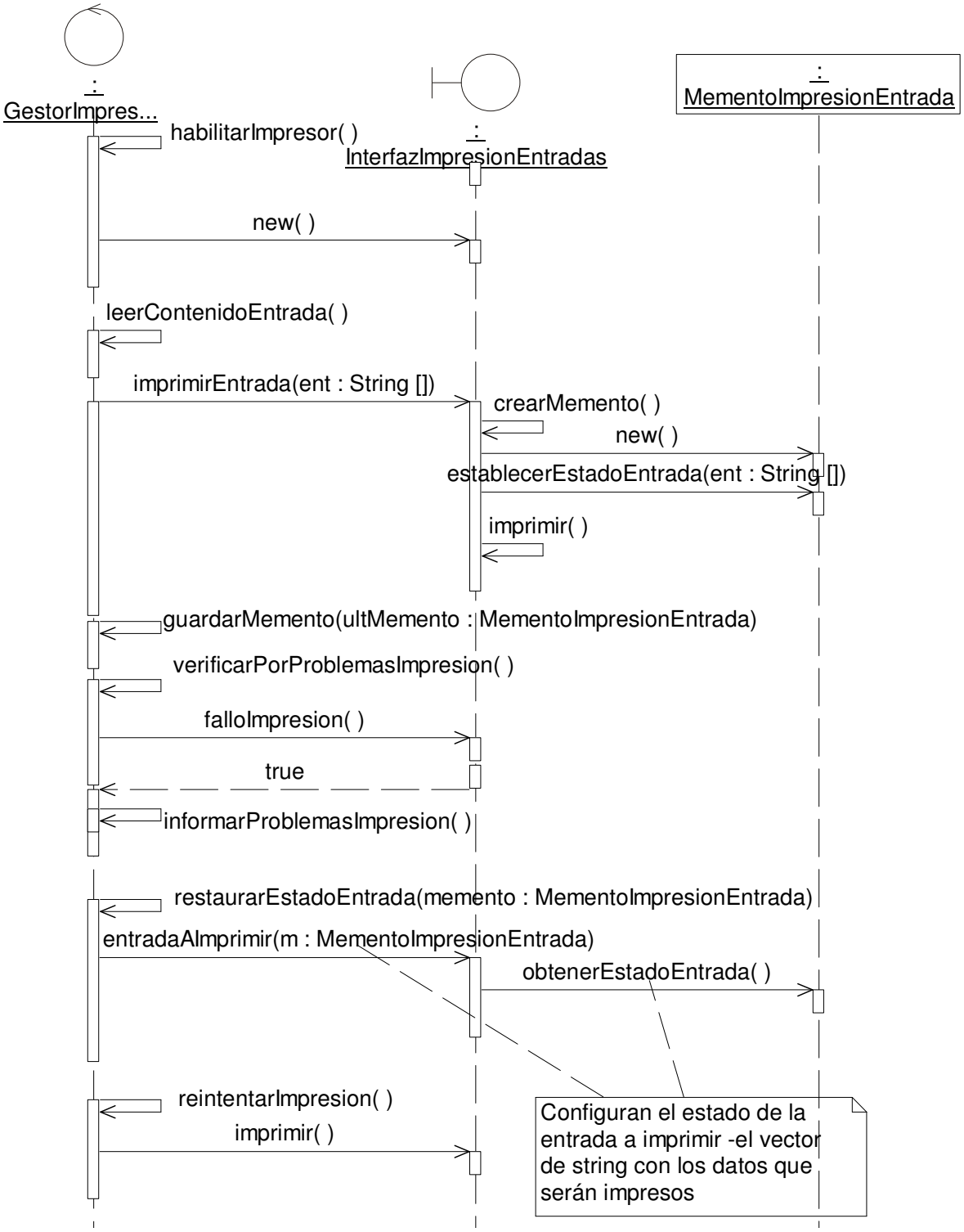
Resolver cómo la cola de impresión de la Interfaz de Impresión deberá gestionar la reimpresión de la/s entrada/s cuando ocurra algún problema a la hora de imprimir éstas. Deberá restaurarse todos los datos de la entrada a reimprimirse.

Nota: modelar sólo la dinámica para reimprimir la última entrada.

**Patrón a Aplicar: Memento**
**Estructura**



**Dinámica**


:
GestorImpres...
habilitarImpresor( )
:
InterfazImpresionEntradas
:
MementoImpresionEntrada
new( )
leerContenidoEntrada( )
imprimirEntrada(ent : String [])
crearMemento( )
new( )
establecerEstadoEntrada(ent : String [])
imprimir( )
guardarMemento(ultMemento : MementoImpresionEntrada)
verificarPorProblemasImpresion( )
falloImpresion( )
true
informarProblemasImpresion( )
restaurarEstadoEntrada(memento : MementoImpresionEntrada)
entradaAImprimir(m : MementoImpresionEntrada)
obtenerEstadoEntrada( )
reintentarImpresion( )
imprimir( )
Configuran el estado de la entrada a imprimir -el vector de string con los datos que serán impresos

Implementación:

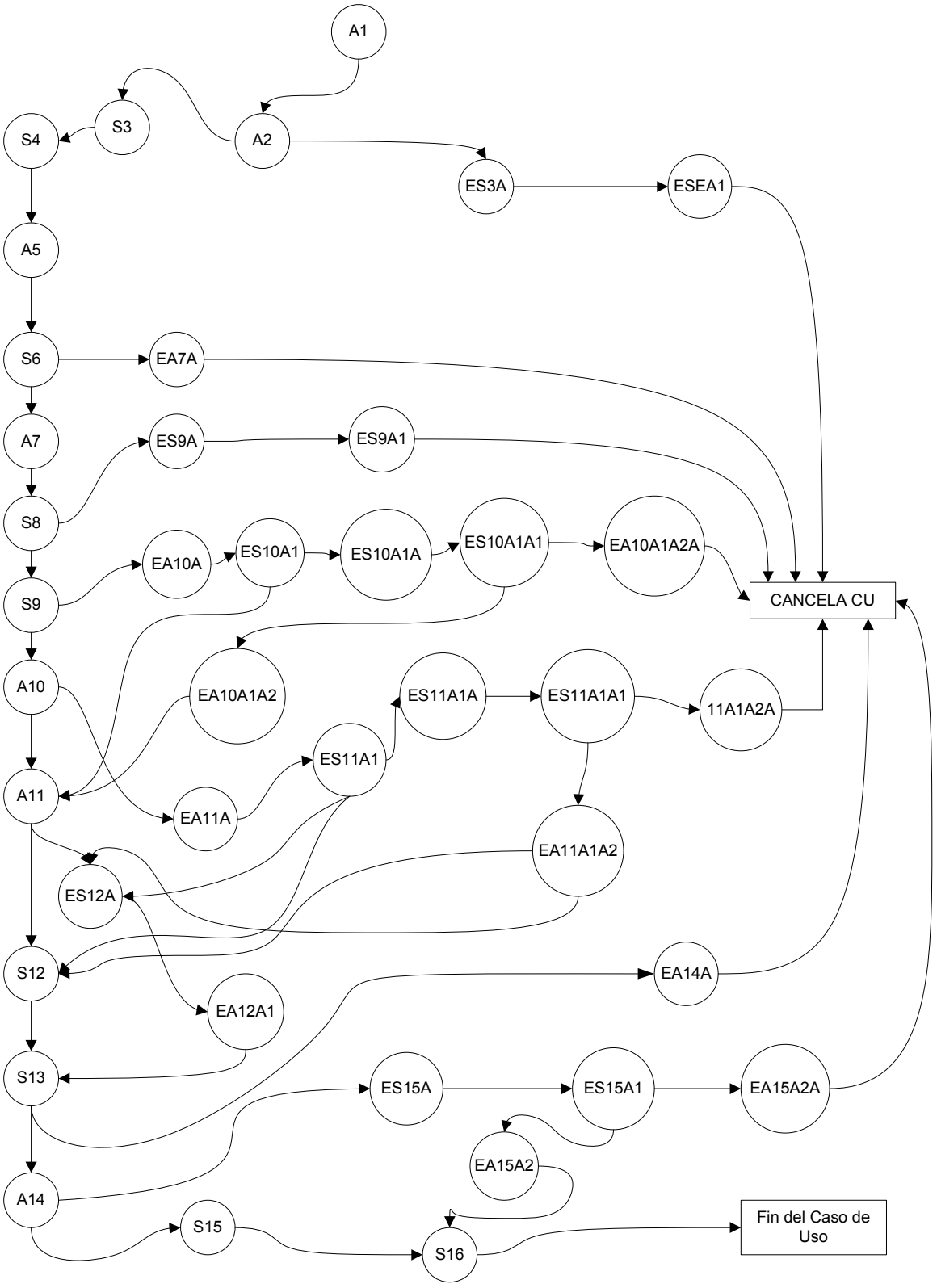
```
//Participante: "Conserje"
public class GestorImpresionEntradas{
//atributos
        private InterfazImpresorEntradas impresor;
        private String()() entradas;
        private ArrayList ()mementos;


//métodos
        ...

        public void imprimirEntradas(){
                //ejecuta las actividades para la impresión de las entradas
                habilitarImpresor();
                leerContenidoEntrada();

                //Por cada entrada a imprimir se realizará el siguiente proceso
                for( int i=0; i < entradas.lenght; i++) {

                        //dispara la orden de impresión y almacena el Memento devuelto de la Entrada
                        guardarMemento( impresor.imprimirEntradas( entradas(i) ) ) ;

                        //verifica posibles errores Interfaz de Impresión
                        boolean huboProblemas=verificarPorProblemasImpresion();
                        if ( huboProblemas==true ) {
                                //informa el problema de impresión al usuario
                                informarProblemasImpresion();

                                //restaura la entrada a imprimir -último Memento guardado
                                MementoImpEntrada ultimoMemento = mementos.get( mementos.size()-1 );
                                restaurarEstadoEntrada( ultimoMemento );
                                //y ejecuta su reimpresión
                                reintentarImpresion();
                        }
                }
        }


        public String()() leerContenidoEntrada(){
        //Aquí estaría el algoritmo para leen los datos que componen los valores a imprimir
        //en cada Entrada.
        entradas=datosEntradas;
        }

        private void habilitarImpresor(){
                impresor=new InterfazImpresorEntradas();
        }


        private void guardarMemento(ultMemento : MementoImpEntrada){
                mementos.add(ultMemento)
        }
```

```
        public void restaurarEstadoEntrada(memento : MementoImpEntrada){
                impresor.entradaAImprimir(memento);
        }

        public void verificarPorProblemasImpresion(){
                boolean rta=impresor.falloImpresion();
                if ( rta==true) informarProblemasImpresion();
        }

        public void reintentarImpresion(){
                impresor.imprimir();
        }

        public void informarProblemasImpresion(){
        //implementar la Excepción que notificará el error de impresión al Usuario.
        }


}



//Participante: "Originador"
public class InterfazImpresorEntradas{

//atributos
        private String ()entrada;
        private boolean falloImpresion;


//métodos
        public new(){}


        public MementoImpEntrada imprimirEntrada(ent : String()){
                entrada=ent;

                MementoImpEntrada memento=crearMemento();
                imprimir();
                return memento;
        }


        private MementoImpEntrada crearMemento(){
                MementoImpEntrada m= new MementoImpEntrada()
                m.establecerEstadoEntrada(entrada);

                return m;
        }


        public void entradaAImprimir(m : MementoImpEntrada){
                entrada=m.obtenerEstadoEntrada();
        }

        public void imprimir(){
        //código específico para imprimir las entradas en el dispositivo
```

```
        }

        public boolean falloImpresion(){
                return falloImpresion;
        }

}



//Participante: "Memento"
public class MementoImpEntrada{
//atributos
        entrada : String()

//métodos
        public new(){}


        public void establecerEstadoEntrada(ent : String()){
                entrada=ent;
        }

        public String() obtenerEstadoEntrada(){
                return entrada;
        }
}
```

## Diseño de Casos de Prueba Dinámica

### Grafo de Caminos para el caso de uso 14: Registrar Película



Caminos de Prueba Positivos

| | |
|---|---|
| 1. | A1 – A2 – S3 – S4 – A5 – S6 – A7- S8 – S9 – A10 – A11- S12 – S13 – A14- S15 - S16 |
| 2. | A1 – A2 – S3 – S4 – A5 – S6 – A7- S8 – S9 – EA10A – ES10A1 - A11- S12 – S13 – A14- S15 - S16 |
| 3. | A1 – A2 – S3 – S4 – A5 – S6 – A7- S8 – S9 – EA10A – ES10A1 – ES10A1A - ES10A1A1 - EA10A1A2 - A11- S12 – S13 – A14- S15 - S16 |
| 4. | A1 – A2 – S3 – S4 – A5 – S6 – A7- S8 – S9 – A10 – EA11A- EA11A1 - S12 – S13 – A14- S15 - S16 |
| 5. | A1 – A2 – S3 – S4 – A5 – S6 – A7- S8 – S9 – A10 – EA11A- EA11A1 - EA11A1A - EA11A1A1 - EA11A1A2 - S12 - S13 – A14- S15 - S16 |
| 6. | A1 – A2 – S3 – S4 – A5 – S6 – A7- S8 – S9 – A10 – A11- S12 – S13 – A14 - ES15A- ES15A1 - EA15A2 - S16 |
| 7. | A1 – A2 – S3 – S4 – A5 – S6 – A7- S8 – S9 – EA10A – ES10A1 - EA11A- EA11A1 - S12 – S13 – A14- S15 - S16 |
| 8. | A1 – A2 – S3 – S4 – A5 – S6 – A7- S8 – S9 – EA10A – ES10A1– EA11A- EA11A1 - EA11A1A - EA11A1A1 - EA11A1A2 - S12 – S13 – A14- S15 - S16 |
| 9. | A1 – A2 – S3 – S4 – A5 – S6 – A7- S8 – S9 – EA10A – ES10A1 - A11- S12 S13 – A14 - ES15A- ES15A1 - EA15A2 - S16 |
| 10. | A1 – A2 – S3 – S4 – A5 – S6 – A7- S8 – S9 – EA10A – ES10A1 – ES10A1A - ES10A1A1 - ES10A1A2 -- EA11A- EA11A1 - S12 S13 – A14- S15 - S16 |
| 11. | A1 – A2 – S3 – S4 – A5 – S6 – A7- S8 – S9 – EA10A – ES10A1 – ES10A1A - ES10A1A1 - ES10A1A2 -- EA11A- EA11A1 - EA11A1A - EA11A1A1 - EA11A1A2 - S12 – S13 – A14- S15 - S16 |
| 12. | A1 – A2 – S3 – S4 – A5 – S6 – A7- S8 – S9 – EA10A – ES10A1 – ES10A1A - ES10A1A1 - ES10A1A2 - A11- S12 – S13 – A14 - ES15A- ES15A1 - EA15A2 - S16 |
| 13. | A1 – A2 – S3 – S4 – A5 – S6 – A7- S8 – S9 – EA10A – ES10A1- EA11A- EA11A1 - S12 – S13 – A14 - ES15A- ES15A1 - EA15A2 - S16 |
| 14. | A1 – A2 – S3 – S4 – A5 – S6 – A7- S8 – S9 – EA10A – ES10A1- EA11A- EA11A1 - EA11A1A - EA11A1A1 - EA11A1A2 - S12 – S13 – A14 - ES15A- ES15A1 - EA15A2 - S16 |
| 15. | A1 – A2 – S3 – S4 – A5 – S6 – A7- S8 – S9 – EA10A – ES10A1 – ES10A1A - ES10A1A1 - ES10A1A2 – EA11A- EA11A1 - EA11A1A - EA11A1A1 - EA11A1A2 - S12 – S13 – A14- S15 - S16 |
| 16. | A1 – A2 – S3 – S4 – A5 – S6 – A7- S8 – S9 – EA10A – ES10A1 – ES10A1A - ES10A1A1 - ES10A1A2 – EA11A- EA11A1 - EA11A1A - EA11A1A1 - EA11A1A2 - S12 – S13 – A14 - ES15A- ES15A1 - EA15A2 - S16 |

Caminos de Prueba Negativos

| | |
|---|---|
| 17. | A1 – A2 – ES3A – ES3A1 |
| 18. | A1 – A2 – S3 – S4 – A5 – S6 – EA7A |
| 19. | A1 – A2 – S3 – S4 – A5 – S6 – A7- S8 – ES9A - ES9A1 |
| 20. | A1 – A2 – S3 – S4 – A5 – S6 – A7- S8 – S9 – EA10A – ES10A1 – ES10A1A - ES10A1A1 - EA10A1A2A |
| 21. | A1 – A2 – S3 – S4 – A5 – S6 – A7- S8 – S9 – A10 – EA11A- EA11A1 - EA11A1A - EA11A1A1 - EA11A1A2A |
| 22. | A1 – A2 – S3 – S4 – A5 – S6 – A7- S8 – S9 – A10 – A11- S12 – S13 –A14 - ES15A- ES15A1 - EA15A2A |
| 23. | A1 – A2 – S3 – S4 – A5 – S6 – A7- S8 – S9 – A10 – A11- S12 –S13 - EA14A |
| 24. | A1 – A2 – S3 – S4 – A5 – S6 – A7- S8 – S9 – EA10A – ES10A1 – ES10A1A - ES10A1A1 - EA10A1A2A - EA11A- EA11A1 - EA11A1A - EA11A1A1 - EA11A1A2A |
| 25. | A1 – A2 – S3 – S4 – A5 – S6 – A7- S8 – S9 – EA10A – ES10A1 - EA11A- EA11A1 - EA11A1A - EA11A1A1 - EA11A1A2A |
| 26. | A1 – A2 – S3 – S4 – A5 – S6 – A7- S8 – S9 – EA10A – ES10A1 – ES10A1A - ES10A1A1 - ES10A1A2 – EA11A- EA11A1 - EA11A1A - EA11A1A1 - EA11A1A2 - S12 – S13 –A14 - ES15A- ES15A1 - EA15A2A |

| Nombre del Caso de Prueba: | **Registración de una película con todos sus datos incluidos el elenco, los premios y comentarios.** | | |
|---|---|---|---|
| **ID del Caso de Prueba:** | 14/2 | | |
| **Juego de Prueba:** | **Registrar Película** | | |
| **Camino de Prueba:** | **A1 – A2 – S3 – S4 – A5 – S6 – A7- S8 – S9 – EA10A – ES10A1 - EA11A- EA11A1 - S12 – S13 – A14- S15 - S16** | | |
| **Prioridad:** | **Alta** | | |
| **Setup:** | Calificaciones cargadas (con la calificación "Prohibida para menores de 13 años" incluida).<br>Rubros cargados (con el rubro "Comedia").<br>Países de Origen cargados (con el país "Estados Unidos").<br>Estado No Disponible<br>Permisos como Responsable de Programación para el usuario Rubén Agüero. | | |
| **Resultado:** | Película registrada, con todos los datos de la misma. | | |
| **Paso** | **Descripción** | **Resultado** | **ID Problema** |
| CP- 14/2-A1 | El usuario Rubén Agüero con rol de Responsable de Programación selecciona la opción **"Película Nueva"** | | |
| CP – 14/2 -A2 | El RP ingresa el nombre de la película "La Gran Estafa" | | |
| CP –14/2- S3 | El sistema valida que no existe película con ese nombre. | | |
| CP – 14/2-S4 | El sistema carga los datos para las selecciones y pide los demás datos para la película. | | |
| CP – 14/2-A5 | El responsable de programación ingresa: **duración : 2:17", título original: Ocean Eleven, año de estreno: 2002** | | |
| CP – 14/2-S6 | El sistema solicita selección de calificación, género y país de origen de la película. | | |
| CP – 14/2-A7 | El RP selecciona **Calificación: Prohibida para menores de 13 años; país de Origen: EEUU; Género: Comedia** | | |
| CP – 14/2-S8 | El sistema llama al caso de uso Registrar Elenco | | |
| CP – 14/2-S9 | El sistema registró el elenco | | |
| CP – 14/2- EA10A | El RP selecciona la opción para registrar premios de película | | |
| CP – 14/2- ES10A1 | El sistema llama al caso de uso Registrar Premio y se registran el o los premios | | |
| CP – 14/2 - EA11A | El RP selecciona la opción para registrar comentarios | | |

| | | | |
|---|---|---|---|
| CP – 14/2 - EA11A1 | El sistema llama al caso de uso Registrar comentarios y se registran el o los comentarios | | |
| CP – 14/2 – S12 | El actor no modifica el estado no disponible de la película. | | |
| CP – 14/2- S13 | El sistema pide confirmación | | |
| CP – 14/2- A14 | El RP confirma la registración | | |
| CP – 14/2 -S15 | El sistema valida que se hayan especificado los datos mínimos (nombre, duración, calificación, género, país de origen, elenco), requeridos para realizar la registración de la película y han sido especificados | | |
| CP – 14/2 - S16 | El sistema registra la nueva película | | |
| | Fin del Caso de Uso | | |
| **Estado del Caso de Prueba** | | | |
| **Analista de Prueba**: | | | |
| **Fecha de Llenado**: | | | |
| **Diseñador del Caso de Prueba** | Judith Meles | | |
| **Fecha de Versión del Caso de Prueba:** 10/08/2008 | | **Versión del Caso de Prueba:** 1.1 | |

**Historia de Cambios**

| Fecha | Versión | Descripción | Autor |
|---|---|---|---|
| 2002 | 1.0 | Versión Original | Judith Meles |
| 2002 – 2003 | 1.1 | Varios cambios no registrados en detalle | Judith Ruiz, Eugenia Corthey<br>Judith Meles<br>Cecilia Scauso |
| 30/04/2004 | 1.2 | Caso de Uso: Vender entradas, se completó con el cobro de las entradas y con el escenario alternativo del punto 5.<br>Caso de Uso 2: se completa el punto 1.<br>Se cambio la forma en la que se indica la herencia en los caso de uso de Reserva Telefónica y via Web.<br>Caso de uso 12: cambio de Fin por cancelación en los escenarios de fracaso. | Judith Meles |
| 09/07/2004 | 1.3 | El cupón de la tarjeta de tarjeta se genera en el caso de uso de Autorización de Venta con Tarjeta de Crédito.<br>Se completaron los diagramas de colaboración con algunos objetos que faltaban.<br>Se corrigieron los modelos de rastreabilidad en función del cambio del cupón de la tarjeta de crédito.<br>Se agregó el diagrama de paquetes.<br>Se corrigieron las descripciones de casos de uso del sistema de información.<br>Se agregaron descripciones de caso de uso de trazo grueso. | Judith Meles |
| 24/08/2004 | 1.4 | Se modificó el caso de uso de Generación de Programación de Funciones.<br>Se incorpora el modelo de objetos del dominio, las diagramas de interacción, diagramas de transición de estados y el modelo de clases de análisis. | Judith Meles |
| 06/10/04 | 1.5 | Se agregan los casos de uso 88, 89 y 90. El 88 en el paquete de Administración del Complejo y los restantes en el paquete de Administración de Programación.<br>Se agregan Diagrama de Despliegue, DER. | Judith Meles |
| 22/12/2004 | 1.6 | Se corrige el caso de uso 14 para contemplar validación de datos mínimos antes de registrar una película. | Judith Meles |

| Fecha | Versión | Descripción | Autor |
|---|---|---|---|
| 15/04/2005 | 1.7 | Se corrige el caso de uso 14 para completar que datos son los mínimos requeridos para dar el alta de la película.<br>Se agrega lo referente al diseño del caso de prueba del caso de uso Registrar Película. | Judith Meles |
| 09/05/2005 | 1.8 | Correcciones de redacción caso de uso de negocio número 1. | Judith Meles |
| 21/05/2005 | 1.9 | Correcciones de redacción caso de uso de negocio número 1.<br>Se agregan las descripciones de los actores de negocio. | Judith Meles |
| 02/10/2005 | 1.10 | Se reemplazan las relaciones estáticas por dependencias en las clases de fabricación pura.<br>Se modifica la colaboración del caso de uso nro 11 para agregar el recurso que intervino. | Judith Meles |
| 27/04/2006 | 1.11 | Se actualizaron notaciones de relaciones entre casos de uso.<br>Se incorpora el caso de uso 91 y se modifica el 14. | Judith Meles |
| 11/08/2006 | 1.12 | Se agregó el estado de disponibilidad al Registrar Película. Se actualizaron el grafo, los caminos y el caso de prueba | Judith Meles y Elizabeth Jeinson |
| 04/09/2006 | 1.13 | Se modificó una relación en el der entre entrada y Tipo de entrada para la función | Judith Meles |
| 20/10/2006 | 1.14 | Se agrega los caso de uso 92 y 93 y se modifica el caso de uso 11 y se agrega la descripción del caso de uso 1 que cambia de nombre de Generar a Imprimir Entrada.<br>Se corrigió la descripción del caso de uso 9 y su correspondiente colaboración. | Judith Meles<br>Cecilia Massano<br>Laura Alarcón |
| 20/10/2006 | 1.15 | Se incorporan las consideraciones de diseño y los patrones de diseño.<br>Se agregan los casos de uso 94 y 95<br>Se corrige la colaboración del caso de uso 11. | Judith Meles<br>Cecilia Massano<br>Laura Alarcón<br>Gerardo Boiero<br>Leonardo Pütz |
| 24/04/2007 | 1.16 | Se agregan prototipos de interfaz. | Cecília Massano |
| 14/05/2007 | 1.17 | Se agrega la clase de dominio COMENTARIO | Judith Meles |

| Fecha | Versión | Descripción | Autor |
|---|---|---|---|
| 02/09/2007 | 2.0 | Se cambia en el Dominio el concepto de tipo de entrada y se agrega el concepto de tipo de función y las promociones que pueden afectar a la venta de las entradas, se cambia el modelo de dominio.<br>Se agrega la vista del modelo de dominio que marca los patrones utilizados<br>Se modifican casos de uso 11 y 2.<br>Se actualizan relaciones en el diagrama de casos de uso del Sistema de Negocio<br>Se elimina fila de post-condición de fracaso que estaba repetida.<br>Se reestructuran los paquetes de casos de uso, dejando los esenciales como una vista y agregando el paquete "Administración de Ventas y Reservas".<br>Se describe el caso de uso 93 y se arregla el 92.<br>Se agrega la columna de paquete en el listado de casos de uso.<br>Se agrega la fila paquete en el template de los casos de uso del sistema de información y se modifica la calificación de complejidad en los casos de uso a trazo fino del sistema de información.<br>Se actualizan los modelos de rastreabilidad de SN –SI<br>Se describe a trazo fino el caso de uso 3 y se hacen las realizaciones de análisis para el escenario del curso normal.<br>Se agregan los número de casos de uso a los diagramas de casos de uso.<br>Se agrega la clase de dominio SESION<br>Se completa el modelo de clases de análisis en función de los cambios realizados.<br>Se agrega la entidad Sesion en el DER.<br>Se agrega el caso de uso 96 y se reasignan los números de casos de uso 90 y 94.<br>Se agregan los casos de uso 97 al 106<br>Se completan las estructuras y dinámicas de los patrones de diseño.<br>Se presentan diagramas de clases de análisis modularizados por funcionalidad<br>Se agrega la descripción de los actores de SI<br>Se agrega el pseudocódigo para los patrones | Judith Meles<br>Gerardo Boiero<br>Laura Covaro<br>Cecilia Massano |