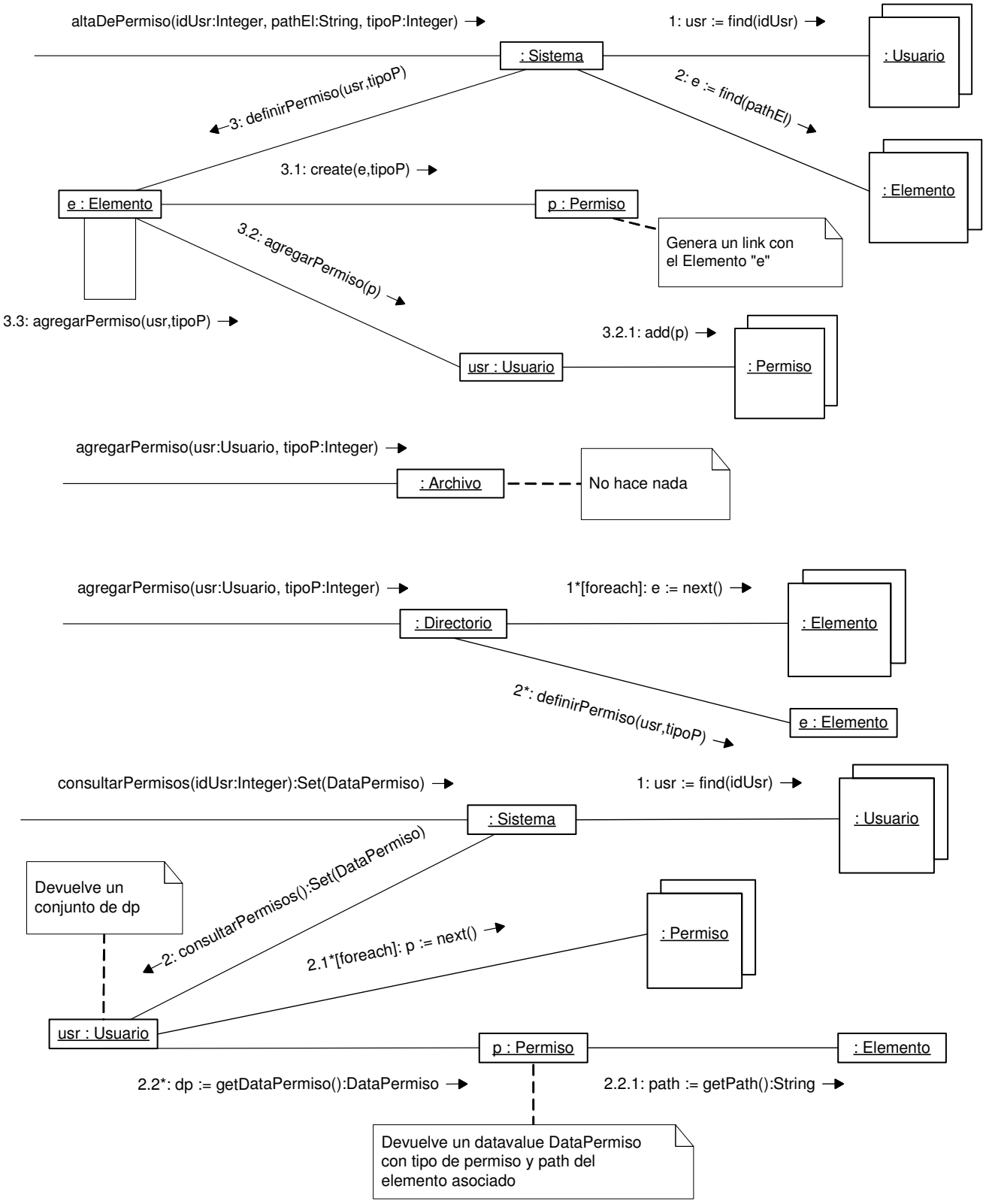
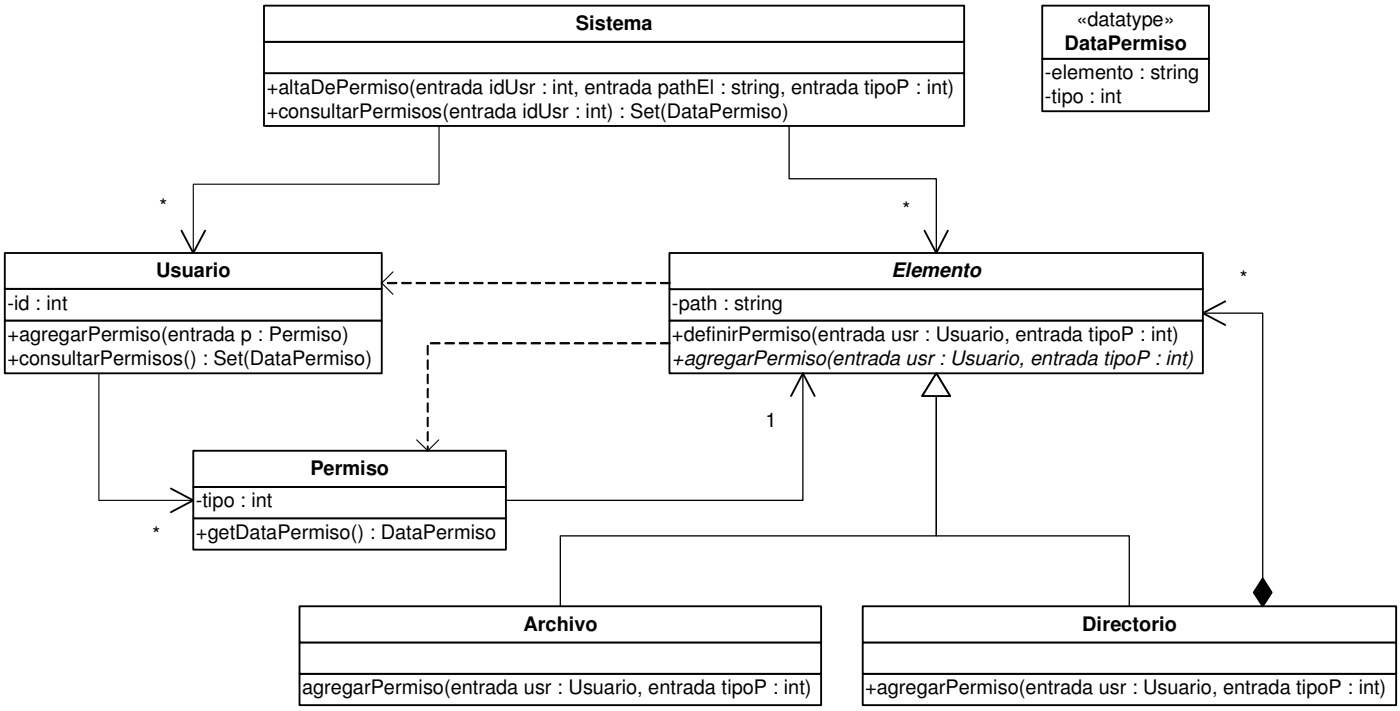
i) Diagramas de Comunicación



ii) Diagrama de Clases de Diseño



i) Implementar los .h de las clases Sistema, Cuenta, Común y Convenio.

```
class Sistema {
      public:
            static Sistema* getInstancia();
            ~Sistema();
            void borrarCuenta(int id);
            int comprarPuntos(int idCuenta, int monto);

      private:
            Sistema();
            static Sistema* instancia;
            IDictionary* cuentas;
            IDictionary* organizaciones;
};
//-----------------------------------------------------------

class Organizacion: public ICollectible {
      public:
            Organizacion(int id, int costoPorPunto);
            ~Organizacion();
            void quitarConvenio(Convenio* convenio);
            int getCostoPorPunto();
      private:
            int id;
            int costoPorPunto;
            IDictionary* convenios;
};
//-----------------------------------------------------------


class Cuenta: public ICollectible {
      public:
            Cuenta(int id);
            ~Cuenta();
            int comprarPuntos(int monto);
            virtual int getCostoPorPunto() = 0;
      private:
            int id;
            ICollection* movimientos;
};
//-----------------------------------------------------------


class Convenio: public Cuenta {
      public:
            Convenio(int id, Organizacion* org);
            ~Convenio();
            int getCostoPorPunto();
      private:
            Organizacion* organizacion;
};
//-----------------------------------------------------------

class Comun: public Cuenta {
      public:
            Comun(int id, int costo);
            int getCostoPorPunto();
      private:
            int costoPorPunto;
};
```

ii) **Implementar en C++ las operaciones especificadas anteriormente y todas aquellas operaciones necesarias para su implementación que pertenezcan a las clases Sistema, Cuenta, Común y Convenio.**

Operación borrarCuenta.

```
void Sistema::borrarCuenta(int id) {
	KeyInteger* key = new KeyInteger(id);
	ICollectible* cuenta = cuentas->find(key);

	if (cuenta == NULL) {
		delete key;
		throw invalid_argument("No existe la cuenta");
	}
	cuentas->remove(key);
	delete cuenta;
	delete key;
}


Cuenta::~Cuenta() {
	IIterator* it = movimientos->getIterator();
	ICollectible* elemento;

	while(it->hasCurrent()) {
		elemento = it->getCurrent();
		it->remove();
		delete elemento;
	}

	delete it;
	delete movimientos;
}


Convenio::~Convenio() {
	organizacion->quitarConvenio(this);
}
```

Operación comprarPuntos.

```
int Sistema::comprarPuntos(int idCuenta, int monto) {
	KeyInteger* key = new KeyInteger(idCuenta);
	Cuenta* cuenta = (Cuenta*) cuentas->find(key);
	delete key;

	if (cuenta != NULL) {
		return cuenta->comprarPuntos(monto);
	} else {
		throw invalid_argument("No existe la cuenta");
	}
}


int Cuenta::comprarPuntos(int monto) {
	int costoPorPunto = getCostoPorPunto();
	if (monto < costoPorPunto)
		throw out_of_range("monto insuficiente");

	int puntos = monto / costoPorPunto;
	Movimiento* mov = new Movimiento(puntos);
	movimientos->add(mov);
	return puntos;
}


int Convenio::getCostoPorPunto() {
	return organizacion->getCostoPorPunto();
}


int Comun::getCostoPorPunto() {
	return costoPorPunto;
}


Movimiento::Movimiento(int puntos) : puntos(puntos) {

}
```