Tartalomjegyzék

1. fejezet

Bevezetés

A szoftverfejlesztési folyamatban egyre nagyobb hangsúlyt kap a tervezés, a rendszer strukturálása a felelősségi körök szeparálásnak elve, az újrafelhasználhatóság, valamint a rendszer szereplőinek izoláltsága szerint. A végrehajtható UML ezen a ponton játszik nagy szerepet, melynek segítségével magas szinten leírhatjuk a rendszer struktúráját, felvehetjük az egyes szereplőket úgy, hogy már a rendszertervünk is tesztelhetővé, a kódgenerálás révén pedig közvetlen felhasználhatóvá válik.
A szereplők izolálása érdekében kommunikációs csomópontokra, portokra és azokat összekapcsoló absztrakt kommunikációs protokollokra, konnektorokra van szükségünk, melynek révén az adott szereplőnek nem szükséges ismernie a kommunikációban részt vevő felet. Valós idejű rendszerek tervezésnél fontos szerepet játszanak a portok és konnektorok (ftp://ftp.omg.org/pub/umlrtf/UML_rt_ext_F56dist.pdf)
A portok és konnektorok fordíthatóságához meg kell ismerni azok pontos UML szemantikáját,a szabvány szerinti reprezentációját, valamint kell találni egy olyan C++ reprezentációt, melynek API-ja letisztult, érhető a felhasználó számára, ugyanakkor az egyes műveletek leghatékonyabb megvalósítására törekszik. A következőkben ezeket a megvalósításokat fogjuk elemezni, egymással összehasonlítani, valamint kiérünk a szabványos UML leképezésre, annak problémáira is.

1.1. UML Komponens fogalmak

1.1.1. Enkapszulált elem

UML-ben az enkapszulált elem egy olyan abstrakt fogalom, egyfajta mechanizmust fejez ki, mellyel képesek vagyunk elszigetelni az objektumot a külvilágtól portok segítségével.
Ez a fogalom nem összekeverendő az UML komponenssel, mely egy enkapszulált elem, viszont több klasszifikált elemet fog össze, egy jól definiált interfésszel rendelkezik, mellyel valamilyen szolgáltatást nyújt a külvilág felé. A komponens fontos tulajdonsága, hogy az adott rendszerben könnyen cserélhető.
Az UML Osztályok is enkapszulált klasszifikációnak számítanak, a diplomamunka az osztályok portjainak exportálását járja körül.

1.1.2. UML Interfész

UML-ben az interfész egy olyan klasszifikáció, melynek a szokványos operációs műveleteken kívül speciális kommunikációs műveletei, úgynevezett reception-ei vannak. Ezek olyan speciális műveletek, melyek egy eseményt várnak, tehát egy interfész leírja, milyen eseményekkel kommunikálhatunk az interfészeken keresztül.

1.1.3. UML Port

Enkapszulált elemek egy adattagja, amely egy kommunikációs csomópontot reprezentál. A típusa lehet egy egyszerű primitív is, de gyakoribb eset, amikor a port egy interfészt valósít meg. Ilyen esetben megkülönböztetünk szolgáltatott és elvárt interfészeket. A szolgáltatott interfész a külvilágtól érkező üzenteteket foglalja magában, míg az elvárt interfész a külvilággal történő kommunikációra szolgál. Arra használjuk, hogy egy klasszifikációt függetlenítsünk a környezetétől, mivel így a külvilággal való kapcsolattartás egy bizonyos típusú porton keresztül történik, nem pedig referencia birtoklásával.

1.1.4. UML Connector

Egy asszociációt realizál két port között. Leírja, hogy milyen portokat köt össze, assembly vagyd delegáció kapcsolat van-e két port között, illetve a kommunikációra használt protokollt.

2. fejezet

Áttekintés

2.1. A probléma rövid meghatározása

A diplomamunka célja teljes körű támogatást nyújtani az interfész portokkal történő kommunikációra a txtUML keretrendszeren belül. A portokat és azok műveleteit az eszköz által helyesen és teljesen leírhatónak tekintjük, ennek problémájával nem foglalkozunk. A probléma megtalálni ennek a helyes UML reprezentációját, mellyel a kódgenerálás tovább tud dolgozni, valamint a megfelelő C++ kód reprezentációt, mely megfelelő hatékonyságú és tisztaságú.

2.2. Keretrendszerek áttekintése

2.2.1. fUML - ALF

A Foundateional UML (fUML) egy Object Managment Group (OMG) által specifikált végrehajtató UML szabvány, melynek szöveges akciónyelvi szinaxistát az ALF (Action Language for Foundational UML) definiálja. Mivel ALF az UML szabvány egy részhalmazán alapszik, így az akciónylvi elemekhez és a strukturális elemekhez (osztály, asszociációk) széleskörű támogatást nyújt, de az állapotgépekhez vagy a kompozíciókhoz nem definiál szintaxist.

2.2.2. Umple

Az Umple (link) szöveges alapú modellezőeszköz támogatja a komponens struktúrák leírását. Itt a portok egyszerű adattgonként jelennek meg, melyekhez úgynenevezett aktivátor metódusokat rendelhetünk, melynek hatása az attribútum változásakor érvényesül. A megváltozott érték pedig végig csorog a kapcsolatban álló portokon, melyre szintén egy aktivátor segítségével reagálhatunk. A connectorok pedig egyszerűen portok egymáshoz kötésével jellenek meg (binding) interfész kompatibilitási problémák nélkül. Ezzel szemben a txtUML-ben interfész portokat adhatunk meg elvárt és szolgáltatott interfésszel, egyszerű atribbútumokat nem, aktivátorok helyett pedig az állapotgép átmenteti vannak kiegészítve a portok dimenziójával. Az Umple nem foglalkozik a szabványos UML2-es repzrezentációval, a C++ kódot közvetlen a modell szövege alapján állítja elő.

Előnyei:

- Online eszközénk is elérhető, nem szükséges hozzá semmit telepíteni.
- Letisztult, jól definiált szöveges szintaxisa van, beleértve a portokat is.
- Tartalmaz kódgenerálást, mely kitrejed a portok akciónyelvére.

Hátrányai:

- Nem generál egy szabványos UML modellt a kódgenerálás előtt, közvetlen a szövegre építkezik.
- A portokat egyszerű adattagként képzi le a feltipúsozás alapján, nem lehet megadni szolgáltatott és elvárt intefészeket.
- Nincs lehetőség tesre szabni az üzenetküldés implementációját a porton keresztül, nincs konfigurációs leírás

2.2.3. StartUML

Támogatja a kompozit diagramok létrehozását, így a portok, konnektorok és interfészekét is. Vizuális modellezőeszköz, a létrehozott diagramokból C++ kódot tudunk exportálni. A Portok vázát exportálja, ahogy az interfészeket is, de primitív

szinten, a portok féltípusozása után egy adattagként generálja hozzá az osztályhoz. A konnektorokat már nem exportálja, ahogy a portokhoz kapcsolódó műveleteket sem.
Előnyei:

- Az UML szabványos elemeivel dolgozik
- Különböző kiterjesztések révén tartalmaz kódgenerálást

Hátrányai:

- Az eszköz mellett meg kell ismerni precízen az UML szabványos elemeit is, hogy fel tudjuk tölteni az elemek tulajdonságait
- A kódgenerálás csak a vázra terjed ki, mely tartalmazza a portoka, de az azokkal való műveleteket már nem.
- Nehézkes, vizuális használat, az elemeket szabadon létrehozhatjuk, és könnyen invalid modellt generálhatunk, míg a txtUML exporja a szabványos modellt exportálás révén hozza létre, így kisebb lehetőségünk van hibázni.

2.2.4. BrigePoint UML

A StartUML-hez hasonlóan az UML szabványos elemeiből építhetünk fel vizuálisan egy szabványos UML diagramot. Azonban nem tudunk osztályok között portokkal kommunikálni, csak a komponensek között, így ez a megoldás nem elég általános. Portot, illetve konnektort önmagában nem tudunk felvenni, az interfészek segítségével tudunk egy kapcsolatot húzni két komponens között, mely leképződik egy porttá és konnektorrá. Hasonlóan a StartUML-hez, az akciókat aktivitás nodeok révén kézzel kell összerakni, így nehéz szabványos port műveleteket létrehozni. Összességében a BridgePointról is elmondható, hogy kezdetlegesen támogatja a portokkal történő kommunikációt, de kiforratlan, hiányos, a kódgenerálás ellenére nehéz érdemben azok reprezentációjáról beszélni.

2.3. Összefoglalás

Összességében az figyelhető meg az egyes exportálási módok között, hogy a portokat mindig valamilyen primitív feltípusozott adattagként exportáljuk. A másik lehetőségünk, melyet mi is követni fogunk, az interfész portokat összetett típusként exportálni, melyben benne van a szolgáltatott és elvárt interfészt. Portok összekapcsolásának, a portokon történő kommunikáció végrehajtási szemantikájával pedig nem mindegyik keretrendszer foglalkozik, még ha részben támogatja is a portok exportálását. Mindebből az figyelhető meg, hogy egy viszonylag új területről beszélünk, melyben egy teljesen körű támogatás és elemzés referencia értékű lehet későbbi megoldások, fejlesztések számára. A diplomamunka kielemez több különböző, teljes körű támogatást a portok és interfészek C++ nyelvre történő exportáláshoz, mely magában foglalja a szabványos UML modell létrehozását egy jól definiált leírónyelv alapján.

3. fejezet

Reprezentációs lehetőségek

3.1. A megfelelő reprezentációk megtalálása

Az eddig leírtakból kiderült, hogy a szabványos reprezentáció megtalálásával is problémákba ütközünk, mely a fellelhető irodalomból és eszközökből is csak nehezen derül ki, így kitérünk ezek ismertetésére is. A C++ reprezentációhoz pedig egy saját Port könyvtárat és generálási módszert választunk, mely a felellethető módszerek finomításából és több különböző ötlet elemzésének következtében hozunk létre.

A továbbiakban a következő elemek reprezentációjával foglalkozunk:

- Interészek
- Portok és üzenet küldés/fogadás művelet
- Connectorok és connect művelet

3.2. Portok reprezentálása

3.2.1. UML-ben

A portok reprezentálása értelemszerű, egy speciális Property-t kell felvenni az osztályon belül. Amit fontos megemlíteni, hogy a szolgáltatott interfész segítségével fogjuk feltipúsozni.

3.2.2. C++-ban

A portokat minden esetben adattagként reprezentáljuk, ezt mi sem tudjuk másképp megtenni. Többféle képen használhatók kommunikációra, de minden esetben egyfajta ravaszként működik a rajtuk történő állapotváltozás. A port adattag értéke reprezentálja az üzentet. Ennek megfelelően általánosságban kétféle megoldás létezik a portokon történő kommunikációra:

- Hasonlóan az események feldolgozásához, úgy a portokon történű üzenetáramlást is átmeneti függvényekkel dolgozzuk fel.
- Külön aktivátor függvényeket valósítunk meg az osztályon belül az egyes portokra, melyek a portok állapotváltozások aktiválódnak.

A mi esetünkben az előbbi megoldás tűnt kézenfekvőnek, mivel az könnyen adaptálható a meglévő generálási szabályok közé, valamint a forrásnyelv is ezt a konvenciót követi.

3.3. Interfészek reprezentálása interfész portok esetén

3.3.1. Reprezentáció UML-ben

Az interfészeket UML-ben értelemszerűen reprezentáljuk, annyiban különböznek csak az osztályoktól, hogy megadhatunk neki. fogadási műveleteket, un. receptionöket.

3.3.2. Reprezentáció C++-ban

C++-ban számos lehetőségünk van kifejezni egy UML interfészt.

Nincs interfész

Egy lehetséges megoldás az is, ha nem vesszük figyelembe a szolgáltatott és elvárt interfészeket akkor, mikor egy adott portot manipulálunk, üzenetet küldünk rá. A

forrásnyelv validációja hivatott kiszűrni az ilyen nem helyes üzenetküldéseket, hivatkozásokat.
Előnyök:

- Kódgenerálás szempontjából nagyon triviális megoldás, nem szükséges foglalkozni egyáltalán vele.
- Alacsony szintű, hatékonysági kérdésekben a legjobb megoldás.

Hátrányok:

- Külső komponensek, portok esetén magunk sem tudunk felvenni interfészeket, ilyenkor már nem hagyatkozhatunk a forrásnyelv validációjára, könnyen hibázhatunk.
- Ha nem is vezetünk be új komponenst, külső kommutáció esetén tetszőleges üzentet küldhetünk a portra, ami szintén hibás lehet.
- A generált kód kevésbé lesz érthető, ha a felhasználó esetleg kíváncsi rá
- Az interfészeket sehol máshol sem tudjuk felhasználni, osztályok esetén sem, mely szintén a modellel való kommunikációban jelenthet hátrányt.

Van interfész

Ezen belül meg kell vizsgálni az interfész port szintaxisát, valamint a mögöttes végrehajtási szemantikát,illetve magának az interfésznek a leírási módjait.
Ahogy azt korábban írtuk, a port típusa az UML-ben a szolgáltatott interfész, ezért kézenfekvő megoldás lenne felvenni egy olyan adattagot, melynek ez az interfész a típusa. Ennek azonban több hátránya is van, sokkal nehezebben tudjuk kifejezni, milyen elvárt interfésze van a portnak, valamint nem tudjuk kifejezni a port típuson belül az üzenetküldés végrehajtási szemantikáját, valamilyen külső implementációra kényszerülünk. Ezáltal bonyolultjuk a megvalósítást, veszítünk az absztrakcióból, de a hatékonyságot nem növeljük.
Így a másik lehetőségünk bevezetni a Port típusát, melynek sablon paramétere a szolgáltatott és elvárt interfész. Ezáltal a következőképpen vehetünk fel egy portot egy osztályba:

// ProvededInf, RequiredInf Definitios.. Port¡ProvededInf, RequiredInf¿ p;

Ezek után technikailag többféle lehetőségünk van megvalósítani a validációt: Generáljunk egy üres osztályt az interfészekből, majd a szignálok származzanak le aszerint, hogy melyik interfész receptionjében vannak benne:
PL: UML interfész neve: PingPongInterfce, a receptionjai pedig a következő szignálokat fogadják: Ping, Pong
Ekkor a generált kód: class PingPongInterfce; class Ping: public PingPongInterfce .. class Pong: public PingPongInterfce ..

Majd a portnak legyen egy send művelete, ami az elvárt interfész szignáljait várja, vagyis olyan szignálokat, melyek implementáják a megfelelő interfészt. send(PingPongInterfce ..)