CORBA komponenty

Jan Outrata

29. října 2002

Abstrakt

Tento dokument představuje Model CORBA komponent (CORBA Component Model - CCM) a jeho použití v implementaci MicoCCM. Předpokládá znalost architektury CORBA. Čtenář je seznámen se základními koncepty CCM vytvořením jednoduchého příkladu komponenty „Hello World“. Jsou představeny nástroje implementace MicoCCM pro vytvoření a použití komponent.

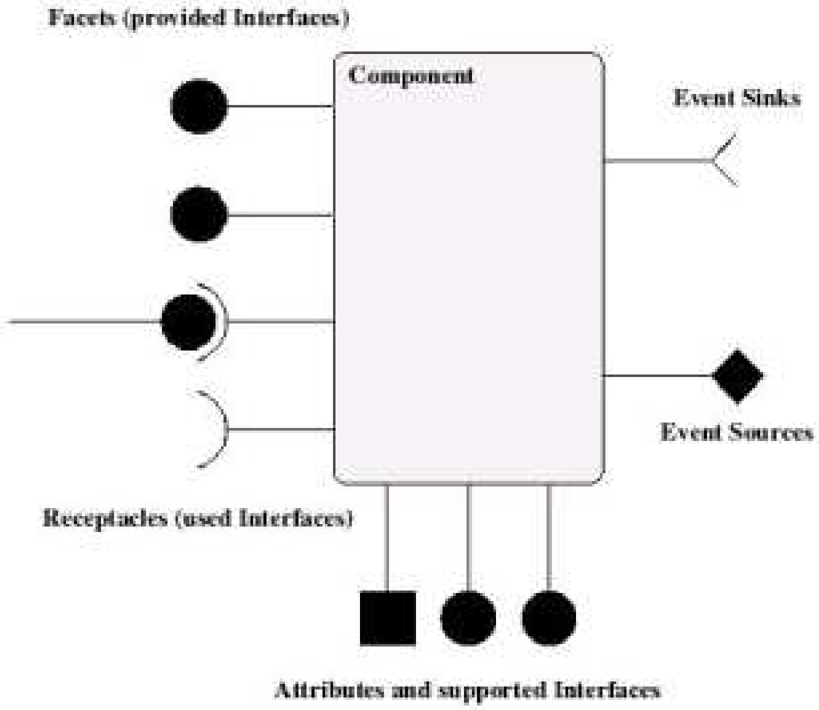
1 Úvod

Model CORBA komponent je hlavní novou součástí standardu CORBA 3.0. CORBA komponenty poskytují nové způsoby integrace komponent. To ovlivňuje jak klientskou část, jak jsou komponenty používány, tak i serverovou část, jak jsou implementovány.

CCM má dvě části, *model komponenty (Component Model)* a *model kontejneru (Container Programming Model)*. Model komponent používá *porty* pro popis vzájemného propojení komponent; komponenta může poskytovat určitá rozhraní nebo používat určitá rozhraní. Např. automat na kávu poskytuje uživatelské rozhraní a používá elektrické rozhraní - to je poskytováno elektrárnou. K dispozici jsou také porty událostí; automat může přijímat události zapnutí a vypnutí od vypínače. Může také vysílat události pokud došla voda nebo káva.

Kontejnery jsou prostředí běhu pro komponentu na straně serveru. Bylo zjištěno, že v CORBA 2 je spousta kódu psána znova pro běžné úlohy jako jsou správa objektových referencí nebo registrace u služby jmen (Name Service). Tyhle monotonní úlohy jsou řešeny automaticky. Implementace CORBA komponenty se nikdy přímo nezabývá ORBem nebo POA. Více běžných služeb („ne-funkčních částí“) je integrováno do prostředí kontejneru, např. bezpečnost nebo transakce. Implementaci pak stačí poskytnout pouze „funkční části“.

Open Source projekt MicoCCM (`http://www.fpx.de/MicoCCM/`) se snaží poskytnout implementaci CCM založenou na Open Source ORBu Mico.



Obrázek 1: Vnější pohled na komponentu

2 Model komponenty

2.1 Úvod

Na obrázku 1 je vnější pohled na komponentu. Komponenta může mít atributy, poskytovat rozhraní a mít jakýkoliv z pěti typů portů. Atributy a rozhraní mohou být používány klientskou aplikací stejně jako dříve. Porty se používají pro propojení komponent. Pak se může použít Assembly tool pro propojení portů komponent, z toho vyplývající skládání aplikací z existujících komponent místo vývoje aplikací.

Typy portů vyjadřují jejich klíčová slova v IDL:

provides Komponenta může poskytovat rozhraní, která se nazývají *facets*.

uses Komponenta může používat rozhraní. Komponenta chce, nebo potřebuje, referenci na jinou komponentu, aby mohla fungovat. Používaná rozhraní se nazývají *receptacles*. Existují *simplex receptacles*, kde může komponenta používat nejvýše jednu objektovou referenci, a *multiplex receptacles*, kde může být připojeno více referencí.

publishes Komponenta může zveřejnit události (zdroj události). Klienti (nebo jiné komponenty používající tuto komponentu) se mohou přihlásit k události.

emits Komponenta může také vysílat události (zdroj události). Zatímco zveřejnění znamená vztah 1:n, ve kterém je událost zaslána více přihlášeným, vyslané události mohou být přijaty pouze jedním příjemcem.

consumes Komponenta může přijímat události určitého typu (cíl události).

Porty nenabízejí nic, co by nešlo udělat „klasickými“ operacemi a atributy - např. atributy jen pro čtení pro facets a operace na propojení receptacles. CORBA komponenty nejsou rozšířením CORBA objektů, ale spíše jejich specializací.

Komponenty explicitně vyjadřují propojení, která nabízejí, a propojení, která očekávají. To je velký rozdíl oproti CORBA serverům, které se zabývají jednoduchými službami.

2.2 Komponenty v IDL

Komponenty jsou popsány pomocí nového klíčového slova *component*. Deklarace komponenty vypadá takto:

```
component <name>
    [ : <base> ]
    [ supports <iface> [, <iface>]* ]
{
  <attribute declaration> *
  <port declaration> *
};
```

Komponenta může rozšiřovat jednu bázovou komponentu - vícenásobné dědění není povoleno - a může podporovat rozhraní. Podporování rozhraní znamená, že všechny atributy a operace rozhraní jsou dostupné v komponentě, můžeme tedy mluvit o „zdědění“ rozhraní.

Deklarace atributů vypadá stejně jako u rozhraní. Deklarace portů jsou velmi podobné atributům, používají se výše uvedená klíčová slova pro specifikaci typu portu.
Příklad komponenty:

```
interface Hello {
  void sayHello();
};

component HelloWorld supports Hello {
  attribute string message;
};
```

2.3 Pohled ze strany klienta

Existují dva typy klientů, *component-aware* a *component-unaware*. Component-aware klienti jsou často sami komponentami a používá se u nich nějaký nástroj k propojení jejich portů. Component-unaware klienti se nestarají o porty komponenty, protože se o ně postará nějaký nástroj v pozadí, dříve než se klient rozběhne.

Ke komponentám v component-unaware klientech lze přistupovat stejnými metodami komunikace jako pro klasické CORBA objekty, pomocí objektových referencí.

Pohled ze strany klienta na komponentu je definován *ekvivalentním IDL*. Komponenta je přeložena, interně v IDL překladači, do ekvivalentního rozhraní. Vedle atributů a operací obsahuje toto rozhraní ekvivalentní operace pro porty komponenty, výsledkem je CORBA 2 rozhraní, které může být použito i component-unaware klienty.

2.4 Domovy

Komponenty jsou spravovány *domovy (homes)*. Domov hraje roli továrny na instance komponent: klient nikdy nevytváří komponentu sám, musí kontaktovat domov komponenty a požádat jej buď o existující instanci komponenty, nebo o vytvoření nové instance. Vývojář komponenty musí vždycky implementovat domov komponenty. V IDL je domov deklarován takto:

```
home <name>
  [ : <base> ]
  [ supports <iface> [, <iface>]* ]
  manages <component_name>
{
  <factories and finders> *
  <operations and attributes> *
};
```

Domov spravuje jeden typ komponenty. Může mít operace a atributy jako rozhraní. *Factories* a *finders* jsou speciální operace, které vracejí nové nebo existující instance komponent. Mají parametry jako normální operace , ale žádný návratový typ se u nich nedeklaruje, implicitně se vrací reference na spravovanou komponentu.

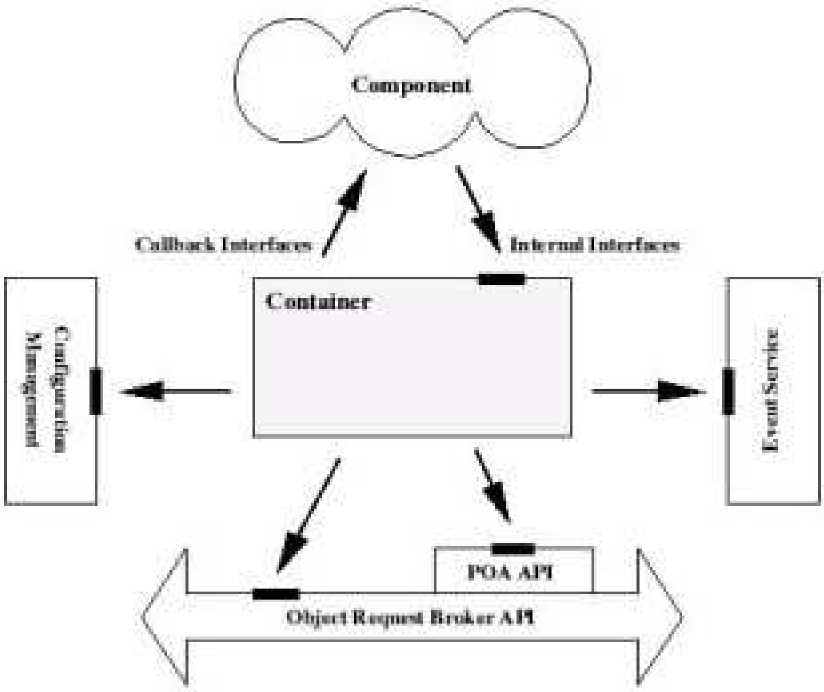
Stejně jako u komponent se i rozhraní domovů dá vyjádřit pomocí ekvivalentního IDL.

Rozlišují se implicitní a explicitní rozhraní domovů. Implicitní rozhraní obsahuje jednu operaci pro vytvoření nové instance komponenty, zatímco explicitní rozhraní obsahuje operace a atributy definované pro domov a ekvivalentní operace pro finders a factories.
Příklad domova:

```
home HelloHome manages HelloWorld {
  attribute string initial_message;
};
```

2.5 Klient komponenty

Klienty moc nezajímá, že server je CORBA komponenta, využívají službu, která nabízí CORBA rozhraní. Napsání klienta komponenty je tedy přímočaré. Přístup k domovu a potom i ke komponentě přes ekvivalentní rozhraní není překvapením. Nejdříve je ale potřeba získat referenci domova komponenty, např. od služby jmen (Naming Service).



Obrázek 2: Model kontejneru

3 Model kontejneru

Ve srovnání s CORBA 2 serverem je implementace CORBA komponenty mnohem jednodušší. Komponenty běží uvnitř prostředí běhu nazývaného *kontejner (container)*. Kontejner je pro komponentu jediný kontakt zvenčí, komponenty nepřistupují k ORBu nebo k objektovým adaptérům přímo, objektové reference jsou spravovány automaticky kontejnerem. Kontejnery mohou poskytovat různé služby jako Event Service nebo Persistance Service.

Komponenta a kontejner interagují přes rozhraní definovaná v IDL. Pro vyhnutí se náročné režii vzdálených volání jsou používána lokální rozhraní. Rozhraní nabízená komponentou a používaná kontejnerem jsou nazývána *callback* rozhraní, rozhraní nabízená kontejnerem a používaná komponentou jsou nazývána *internal* rozhraní (viz obrázek 2).

Implementace komponenty je redukována na minimum, stačí specifikovat výkonné metody, tj. atributy a metody rozhraní, atributy vlastní komponenty, a ekvivalentní metody pro porty. Nemusí se implementovat `main`, protože komponenta není spouštěna, ale umístěna do běžícího serveru komponent.

Kontejnery mohou vytvářet instanci komponenty vždy, když je volána metoda domova `create`, nebo mohou vytvořit jednu instanci komponenty, která je sdílena všemi klienty (obvykle bezstavové komponenty).

3.1 Pohled ze strany serveru

Vývojář komponenty musí implementovat callback rozhraní, v terminologii CORBA komponent je to *executor*, který nahrazuje pojem *servant* v programování POA.

Stejně jako na klientské straně komponenty, je implementace založena na ekvivalentním IDL. Pro každou komponentu a domov jsou definována lokální rozhraní a porty jsou přeloženy jako operace těchto rozhraní.

Pro každou komponentu jsou definována dvě rozhraní, executor callback rozhraní a *context* internal rozhraní. První implementuje vývojář, druhé je implementováno kontejnerem a může být používáno implementací komponenty.

3.2 Implementace komponenty

Vývojář komponenty vytváří implementační třídu, kterou dědí z ekvivalentního executor rozhraní komponenty. Tato třída poskytuje implementace pro všechny atributy a metody.

Implementační třída domova se také dědí z ekvivalentního rozhraní a implementuje metody.

3.3 Vstupní bod

Zbývá ještě dodat *vstupní bod (entry point)*. Kontejner neví, jak vytvořit instanci domova: pro vytvoření instance C++ třídy je potřeba znát celou deklaraci třídy, tu ale kontejner nemá. Proto se musí zadat známý vstupní bod, který kontejner zavolá pro vytvoření instance executoru domova.

Tento vstupní bod je funkce se stejným jménem jako jméno domova, plus prefix `create_`. Musí existovat jeden vstupní bod pro každý domov.

4 Komponenty v MicoCCM

4.1 Generování kódu

Generování částí stub a skeleton komponent:

```
idl hello.idl
```

Překladač interně provede transformaci z IDL 3 do IDL 2 pro vygenerování ekvivalentního IDL. Vytvoří se soubory `hello.h` a `hello.cc`. Stub je pro klienta, u serveru se nepoužívá skeleton přímo, implementuje se executor komponenty.

Kontejner potřebuje kód pro interakci s implementací komponenty:

```
mico-ccm hello.idl
```

Tento nástroj má volbu `--standalone`, pomocí které lze vytvořit samostatný server; přidá `main` ke generovanému kódu a můžeme vytvořit spustitelný program místo sdílené knihovny. Vytvoří se soubory `hello_ccm.h` a `hello_ccm.cc`, zkompilované je potřeba je přilinkovat k implementaci komponenty.

4.2 Kompilace a linkování

Komponenty mohou být zkompilovány do sdílené knihovny (loadable component), která je připojena k serveru komponent (Component Server), nebo do samostatného programu serveru (standalone server), to záleží na uvedení volby `--standalone` výše. Sdílená knihovna musí být linkována proti knihovně `micoccm`, samostatný program ještě oproti knihovně `micocoss`, protože potřebuje Naming Service.

Celý proces sestavení samostatného programu komponenty HelloWorld (implementace je v souboru `hello_impl.cc` a klient v souboru `client.cc`):

```
idl hello.idl
mico-c++ -c hello.cc
mico-c++ -c client.cc
mico-ld -o client client.o hello.o \
-lmicoccm2.3.7 -lmicocoss2.3.7 -lmico2.3.7

mico-ccm --standalone hello.idl
mico-c++ -c hello_ccm.cc
mico-c++ -c hello_impl.cc
mico-ld -o server hello_impl.o hello_ccm.o hello.o \
-lmicoccm2.3.7 -lmicocoss2.3.7 -lmico2.3.7
```

4.3 Vykonávání komponent

4.3.1 Samostatné servery

Komponenty sestavené jako samostatný server lze spustit přímo. Všechny domovy jsou umístěny a aktivovány, a registrovány v Naming Service pomocí jména každého domova.

4.3.2 Sdílené knihovny

Sdílené knihovny se musí připojit k serveru komponent, v MicoCCM je to program `componentserver`. Ten se ale nestartuje ručně, spouští jej podle potřeby jiný server, MicoCCM démon `mico-ccmd`. Program `ccmload` pak kontaktuje tohoto démona pro vytvoření serveru komponent a připojení komponenty k němu.

Následující kroky by připojily („nahrály") komponentu HelloWorld:

```
mico-ccmd --ior ccmd.ior
ccmload --ccmd file://`pwd`/ccmd.ior --ns HelloHome HelloHome hello.so
```

4.4 Skládání komponent

Jak bylo zmíněno dříve, komponenty by měly být snadněji (znovu)použitelné než klasická CORBA rozhraní. Komponenty jsou určeny k propojování a tím tedy skládání aplikace. K tomuto skládání existuje grafický nástroj MicoCCM Assembly and Deployment Toolkit.