RMI

Protože jedním z cílů projektu bylo umožnit uživatelům práci se vzdálenými databázemi, bylo nutné navrhnout způsob síťové komunikace. Nabízely se dvě možnosti: **vlastní komunikační vrstva** a využití některé z existujících technologií, konkrétně **RMI**.

Vlastní komunikační vrstva

Původně jsme zvažovali implementaci vlastní komunikační vrstvy pro práci se vzdálenými databázemi. To by ovšem vyžadovalo návrh vlastního rozhraní a jeho implementaci, která by zahrnovala způsob rozkládání objektů tak, aby je bylo možné posílat po síti a opět rekonstruovat. V aplikaci pracujeme s holder objekty Hibernate, které jsou složitě provázány, museli bychom si umět poradit jednak s nimi a jednak s vracením kolekcí (polí) těchto objektů. Krom toho by bylo zapotřebí vyřešit uspokojivě rozdíly v lokální komunikaci a v komunikaci vzdálené - pokud bychom nechtěli používat dvojí systém práce, museli bychom navrhnout speciální adaptéry, které by pro vyšší vrstvy, tj. těmi, které komunikační vrstvu používají, sjednocovaly síťové a lokální použití vrstvy bezprostředně pod vrstvou komunikační. Postupně se ukázalo, že k tomuto účelu lze stejně dobře použít již existující technologii - technologii vzdáleného volání metod, RMI, která je v každém virtuální stroji (JVM) standardně obsažena.

Další nevýhodou by se pravděpodobně ukázal i návrh serveru a komunikace s klienty pomocí dalších vláken nebo synovských procesů. Výhoda takového řešení by spočívala v ověřené architektuře (dva vývojáři Plantlore měli zkušenosti s psaním serverů - programovali vlastní FTP server).

RMI

Důvody pro použití RMI vyplývaly přímo z kladených požadavků na řešení síťové komunikace, zejména na to, aby

1. celá komunikační vrstva byla maximálně transparentní pro vyšší vrstvy,
2. bylo co možné jednoduše sjednotit pohled na komunikaci síťovou i lokální, a aby nedocházelo ke zbytečné režii v případě lokální komunikace.

Hlavní výhodou, kterou RMI nabízí, je vlastní síťový protokol, tedy vlastní transport všech objektů po síti. Jako programátoři bychom se tedy nemuseli starat o to, jak mnohdy komplexní objekty poslat přes síť - RMI tento problém řeší z hlediska programátora velmi elegantně, plně postačuje použít Serializable interface (tento interface nemá žádné metody). Téměř všechny objekty třídy java.land i java.utils jsou již serializovány, mj. i námi používané kolekce. Nebylo by tedy zapotřebí vyvinout žádné větší úsilí.

Další výhodou, která se nám jako programátorům velmi hodila, byl jednotný způsob práce s objekty lokálními (tedy objekty "žijícími" ve stejné JVM, ve které je i volající) a objekty vzdálenými. Jediná penalizace spočívala v odchytávání výjimky RemoteException. Byli bychom s to velmi elegantně, vyřešit problém s komunikační vrstvou splňujíce oba kladené požadavky.

Nevýhodou byla praktická neznalost technologie a jejích úskalí. Zejména bylo potřeba dořešit způsob vytváření objektů ve vzdálené JVM a předávání zpátky na klientskou JVM. Dále bylo nutné zajistit korektní rušení těchto objektů a to i v případě, že vzdálený klient spadl nebo že došlo k dlouhodobému výpadku síťového spojení, neboť se jednalo o objekty databázové vrstvy, které uchovávaly otevřená DB spojení a mnohdy měly provedeny dotazy nad databází.

Závěr

Postupně se ukázalo, že použití RMI bylo dobré rozhodnutí - mohli jsme se soustředit na problémy spojené s celkovou komunikací, ne na detaily implementace přenosu objektů nebo řízení zpracování požadavků na serveru, neboť RMI mechanismus zaručuje vykonání vzdáleného volání v různých vláknech automaticky, uzná-li to za vhodné.

Zejména jsme však dosáhli splnění obou dvou kladených požadavků.

Vyskytly se potíže pouze technického rázu vycházející z různé implementace síťování v různých operačních systémech, zmiňme např. zvláštní chování RMI při vyhození ConnectionException (výjimka při ztraceném připojení) v OS Windows XP, které způsobovalo viditelné zpomalení celé aplikace. Další problémy souvisely s problematickou identifikací serverů, nastavování cest k třídám pro stuby (v middleware terminologii proxy).