TECHNOLOGIE RTSJ A JEJÍ UPLATNĚNÍ PŘI TVORBĚ APLIKAČNÍ A PREZENTAČNÍ VRSTVY PORTÁLOVÝCH ŘEŠENÍ

Ivo Martiník
Ekonomická fakulta VŠB-TU Ostrava, Sokolská třída 33, 701 21 Ostrava 1, ČR,
ivo.martinik@vsb.cz

ABSTRAKT:
Příspěvek se zabývá základními principy technologie RTSJ (*Real-Time Specification for Java*) zejména v oblasti členění a správy paměťové haldy a jejím možným uplatněním při implementaci aplikační a prezentační vrstvy portálových řešení. Nasazení technologie RTSJ má rovněž zajímavé aspekty v souvislosti se specializovaným návrhovým vzorem *Brick-Box Model-View-Controller* a jeho použitím při realizaci prezentační vrstvy portálových řešení. Uvedený návrhový vzor je prakticky využíván pro implementaci portálového řešení EkF VŠB-TUO s cílem sjednocení návrhu a implementace prezentační vrstvy v prostředí tenkého a tlustého klienta.

KLÍČOVÁ SLOVA:
RTSJ, programová vlákna reálného času, paměťové oblasti, Model-View-Controller, prezentační vrstva, tenký klient

1. ÚVOD

Celosvětový úspěch programovacího jazyka Java vedl v posledním období k zájmu o rozšíření jeho specifikace o funkcionality umožňující vývoj a běh programů reálného času. Uvedené snahy pak dospěly v rámci aktivit JCP (*Java Community Process*) v lednu 2002 ke schválení první verze specifikace RTSJ (*Real-Time Specification for Java* – viz [1], [2]). První komerčně dostupná implementace příslušných knihoven a dalších rozšíření platformy Java byla k dispozici v jarních měsících roku 2003. V současné době je aktuálně platná specifikace RTSJ v. 1.0.2 schválená v létě roku 2006 a implementovaná v programovém balíčku `javax.realtime`. Specifikace RTSJ provádí rozšíření platformy programovacího jazyka Java především v následujících oblastech:

- správa paměti,
- práce se zdroji reálného času,
- plánování v reálném čase,
- programová vlákna reálného času,
- správa asynchronních událostí,
- synchronizace a sdílení zdrojů v reálném čase,
- možnost přímého přístupu k vybraným oblastem paměti.

Specifikace RTSJ nadále podporuje princip WORA (*Write Once, Run Anywhere*), i když její aktuální implementace vyžadují běh programů nad operačními systémy, které podporují práci s programovými vlákny reálného času (např. Sun Solaris, Linux, IBM AIX, apod.). Specifikace RTSJ pak rovněž nevyžaduje žádná syntaktická rozšíření aktuální specifikace programovacího jazyka Java a při psaní programů je možno využívat i existujících knihoven jazyka, které nejsou primárně určeny pro běh programů v reálném čase.

2. SPRÁVA PAMĚTI V RÁMCI SPECIFIKACE RTSJ

Jedním z klíčových problémů řešených v rámci specifikace RTSJ je správa programové haldy. Standardní implementace JVM (*Java Virtual Machine* – viz [5], [6]) vyžívá pro tyto účely služeb programového vlákna realizujícího funkcionality správce programové haldy (*garbage collector*). Správce programové haldy provádí alokaci paměti pro potřebu vytváření nových instancí tříd a rovněž její dealokaci v případě destrukce objektů. Dealokace paměti může být realizována jednak v okamžiku, kdy není na haldě k dispozici již žádná volná oblast paměti, případně v souvislosti s každou alokací paměťové oblasti pro nově vytvářené instance tříd nebo jako reakce na asynchronní událost. Ve všech případech programové vlákno správce programové haldy pozastaví po dobu nutnou k dealokaci paměti činnost všech ostatních aktivních programových vláken, což může mít samozřejmě velmi negativní důsledky pro korektní činnost programových vláken reálného času. I když v této souvislosti bylo v minulých letech věnováno velké úsilí směřující k implementaci speciálního správce haldy reálného času (*real-time garbage collector*), aktuální implementace RTSJ 1.0.2 stále ještě tímto prostředkem nedisponuje.

Pro potřeby korektního běhu programových vláken reálného času proto RTSJ specifikace zavádí pojem paměťových oblastí (*memory area*), z nichž některé jsou alokovány mimo oblast standardní programové haldy a nejsou dotčeny aktivitami správce programové haldy. Z důvodu interakce programových vláken reálného času se standardními programovými vlákny pak specifikace umožňuje pozastavení činnosti správce programové haldy programovým vláknem reálného času a rovněž shora omezenou dobu čekání na toto pozastavení činnosti ze strany příslušného programového vlákna od okamžiku vznesení požadavku.

Abstraktní třídou, jejíž podtřídy jsou určeny k práci s jednotlivými typy paměťových oblastí, je v rámci RTSJ specifikace třída `MemoryArea`. Jestliže programové vlákno reálného času využije ke své práci instanci třídy reprezentující příslušný typ paměťové oblasti, jsou pak již veškeré alokace paměti potřebné k vytváření objektů realizovány uvnitř této oblasti. Deklarovány jsou pak následující podtřídy abstraktní třídy `MemoryArea`:

- `HeapMemory` – třída reprezentující standardní programovou haldu v prostředí JVM,
- `ImmortalMemory` – paměťová oblast sdílená všemi programovými vlákny. Instance tříd alokované v této paměťové oblasti nejsou dotčeny činností správce programové haldy (a tedy nebudou po svém vytvoření a inicializaci podrobeny destrukci po celo dobu životního cyklu příslušné aplikace). Tato paměťová oblast je pak rovněž určena pro alokaci statických proměnných a jejich hodnot deklarovaných a inicializovaných v rámci jednotlivých tříd programového systému a konstantních řetězců (*intern string*).

- `ScopedMemory` – paměťová oblast přístupná pouze entitám reálného času (programovým vláknům a ovladačům asynchronních událostí). S každou instancí (úsekem) paměťové oblasti tohoto typu je asociován speciální čítač uchovávající informaci o tom, kolik entit reálného času ji aktuálně využívá. Pokud je hodnota čítače rovna nule (a tedy žádná entita reálného času již danou paměťovou oblast nevyužívá), zabezpečí správce programové oblasti provedení destruktorů všech objektů v ní rezidentních a uvolní tuto paměťovou oblast pro další alokaci. Svým charakterem chování připomíná tento typ paměťové oblasti některé vlastnosti standardního zásobníku návratových adres. `ScopedMemory` je deklarována jako abstraktní třída s následujícími podtřídami:

    - `VTMemory` – třída reprezentuje paměťovou oblast, v níž není doba potřebná k alokaci nově vytvářené instance třídy úměrná velikosti alokovaného objektu,

    - `LTMemory` – třída reprezentuje paměťovou oblast, v níž jsou doba alokace nově vytvářené instance třídy a velikost alokovaného objektu lineárně závislé.

Entitám reálného času je umožněno při jejich inicializaci a poté v rámci jejich životního cyklu měnit vybrané parametry paměťových oblastí, mezi něž pak zejména patří:

- Maximální rozsah paměti, kterou smí entita reálného času alokovat ve své standardně přidělené paměťové oblasti pro své potřeby,

- Maximální rozsah paměti, který smí entita reálného času využívat prostřednictvím instance třídy `ImmortalMemory`,

- Maximální rychlost (v bytech za sekundu), s jakou smí entita reálného času alokovat paměť v dané paměťové oblasti.

Tyto údaje pak může využívat plánovač programových vláken reálného času při své činnosti.

3. SPECIFIKACE RTSJ A PROGRAMOVÁ VLÁKNA REÁLNÉHO ČASU

Plánování běhu programových vláken reálného času je jedním z kritických aspektů programových systémů reálného času. JVM umožňuje přiřazovat standardním programovým vláknům 10 úrovní jejich priorit a samotné přidělování času procesoru jednotlivým programovým vláknům může být realizováno až na úrovni operačního systému, nad nímž pracuje prostředí JVM ve formě samostatného procesu. Příslušný operační systém přitom vůbec nemusí podporovat preemptivní na prioritách založené přidělování času procesoru jednotlivým programovým vláknům. Na úrovni JVM proto nelze garantovat, že programové vlákno, které má v daném časovém okamžiku nejvyšší prioritu a je připraveno k běhu, bude mít skutečně přidělen plánovačem úloh potřebný čas procesoru. Tento stav je z pohledu provozu systémů reálného času samozřejmě neakceptovatelný.

Specifikace RTSJ proto zavádí několik mechanismů s cílem dosáhnout kvalitativně odlišného deterministického plánování aktivit v prostředí podporujícím běh úloh v reálném čase. Prvním krokem je zobecnění pojmu programového vlákna, které je předmětem činnosti plánovače úloh v prostředí standardní implementace JVM, a jeho nahrazení pojmem plánovatelného objektu (*schedulable object*). Deklarace třídy příslušná každému

plánovatelnému objektu je povinna implementovat rozhraní `Schedulable`. Každý plánovatelný objekt musí rovněž deklarovat jeho specifické

- požadavky na zahájení běhu (za jakých podmínek je objekt spustitelný),
- požadavky na alokaci paměťových oblastí,
- požadavky pro plánovač objektů reálného času (např. priorita běhu plánovatelného objektu).

Požadavky na zahájení běhu jsou specifikovány prostřednictvím instancí podtříd třídy `ReleaseParameters`. U všech podtříd je možno zadat např. hodnotu parametru `cost` specifikujícího maximální dobu běhu plánovatelného objektu po přidělení času procesoru. Jednotlivými deklarovanými podtřídami pak jsou:

- `PeriodicParameters` – instance této třídy deklaruje požadavky na zahájení běhu spustitelného objektu, jehož spouštění se opakuje s přesně stanovenou periodou,
- `AperiodicParameters` – instance třídy deklaruje požadavky na zahájení běhu spustitelného objektu, jehož spouštění se realizuje v náhodných časech,
- `SporadicParameters` – instance třídy deklaruje požadavky na zahájení běhu spustitelného objektu, jehož spouštění se realizuje v náhodných časech, ale je definován minimální časový interval mezi jednotlivými přiděleními času procesoru.

Plánovač objektů reálného času je implementován jako podtřída abstraktní třídy `Scheduler`. Specifikace RTSJ povoluje využívat v rámci daného procesu JVM dokonce několik plánovačů objektů reálného času současně, nicméně typicky je v reálných programových systémech využívána pouze jediná instance plánovače implementovaná s využitím třídy `PriorityScheduler`, která je založena na filozofii preemptivního plánování běhu objektů reálného času disponujícím 28 úrovněmi priorit.

Jedním z významných typů objektů reálného času je samozřejmě programové vlákno reálného času, které lze vytvořit prostřednictvím instance třídy `RealtimeThread` (podtřídy třídy `java.lang.Thread` reprezentující standardní programové vlákno v prostředí JVM). Třída `RealtimeThread` samozřejmě implementuje rozhraní `Schedulable` a její významnou podtřídou je třída `NoHeapRealtimeThread`, při jejímž použití je garantováno, že programové vlákno reálného času nebude žádným způsobem využívat alokace paměti na programové haldě a vyhne se tak interakci se programovým vláknem správce programové haldy.

Jednoduchý příklad programu obsahujícího deklaraci a spuštění programového vlákna reálného času včetně inicializace jeho základních parametrů (priorita běhu vlákna bude mít hodnotu standardní hodnotu `NORM_PRIORITY`, vlákno bude spouštěno neperiodicky v náhodných časech, bude pro alokaci paměťových oblastí využívat standardní programovou haldu, nebude požadovat alokace v paměťové oblasti typu `ImmortalMemory`, její spustitelný kód je uložen v metodě `run` deklarované třídy `HelloThread` a po svém spuštění vypíše na konzolu nápis *Hello, world!*) je uveden ve výpisu č. 1.

Výpis č. 1

```
import javax.realtime.*;

public class Hello
{
  public static void main(String[] args)
  {
    SchedulingParameters schedulingParams =
      new PriorityParameters( PriorityScheduler.NORM_PRIORITY );
    ReleaseParameters releaseParams =
      new AperiodicParameters( null, null, null, null);
    MemoryParameters memoryParams =
      new MemoryParameters( MemoryParameters.NO_MAX, // Heap
                            0);                      // Immortal
    MemoryArea area = HeapMemory.instance();
    Thread thread = new MyThread();
    RealtimeThread rt =
       new RealtimeThread( schedulingParams, releaseParams,
                           memoryParams, area, null, thread );
    rt.start();
  }


  private class MyThread extends Thread
  {
    public void run()
    {
      System.out.println( "Hello, world!" );
    }
  }
}
```

3. SPECIALIZOVANÝ NÁVRHOVÝ VZOR BRICK-BOX MODEL-VIEW-CONTROLLER PRO IMPLEMENTACI PREZENTAČNÍ VRSTVY NEVEŘEJNÉ ČÁSTI FAKULTNÍHO PORTÁLU

Jedním z významných návrhových vzorů, jež se již mnoho let výrazně uplatňují při návrhu a implementaci nejen informačních systémů, představuje *Model-View-Controller* (MVC – viz [7]). Koncept MVC je od jeho samotných počátků svázán s paradigmatem objektově-orientovaného programování, lze jej samozřejmě uplatnit i v souvislosti s vícevrstvými distribuovanými programovými systémy s využitím konceptů budování portálů na bázi Java EE architektury, kde portlet (viz [3], [4]) tradičně vystupuje v roli *controlleru*, komponenta *view* je typicky implementována technologií Java Server Pages (JSP – viz [8]) a v úloze komponenty *model* nalezneme instance tříd Java Bean, příp. Enterprise Java Bean.

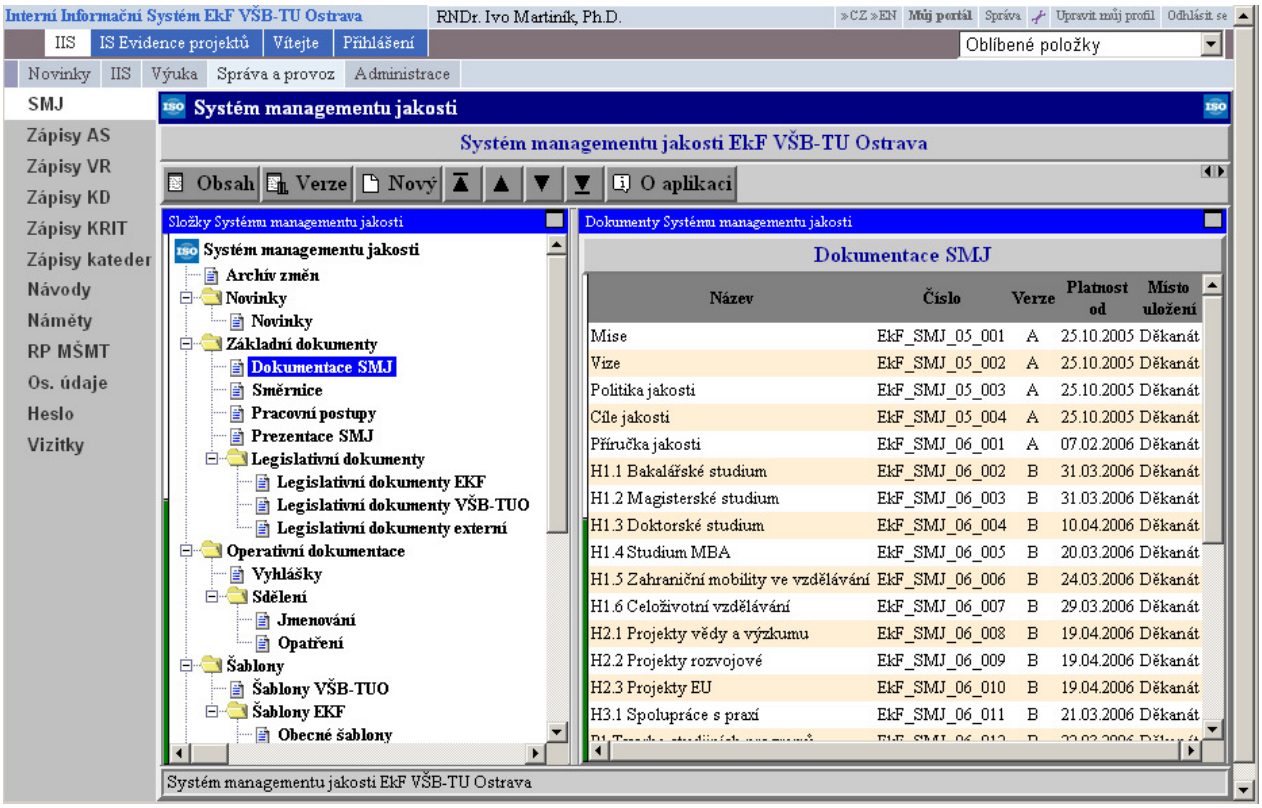
Implementace kvalitní prezentační vrstvy na straně tenkého klienta je pak jedním z klíčových problémů realizace neveřejné části portálu. Hlavním cílem v této oblasti je dosažení maximálně kvalitního uživatelského interface (včetně grafických komponent typu *dialog box*, *pop-up menu*, *tree* apod., jež nejsou standardně v rámci technologie XHTML dostupné) na straně uživatele bez možnosti využití instalovaných plug-in komponent WWW prohlížeče (zejména technologie Java Applets, Flash apod.). Autory projektu byla tedy

navržena a implementována poměrně rozsáhlá knihovna objektových komponent v programovacích jazycích Java na straně aplikačního serveru a JavaScript na straně tenkého klienta k realizaci uživatelského rozhraní na straně tenkého klienta s následujícími vlastnostmi:

- Filozofie tvorby grafického uživatelského rozhraní strany tenkého klienta je obdobná filozofii tvorby grafického uživatelského rozhraní tlustého klienta (známé např. z technologií *Java AWT* a *Java Swing*). Za tím účelem byla implementována knihovna tříd Java Bean, pomocí nichž je prováděna tvorba komponent *view* strany portálového serveru.

- Při generování grafického uživatelského rozhraní je kladen maximální důraz na rychlost vytváření konkrétního fragmentu WWW stránky, který typicky obsahuje značné množství grafických uživatelských komponent. Z tohoto důvodu bylo upuštěno od nasazení technologie Java Server Pages v roli komponent *view* návrhového vzoru, u nichž např. probíhá při prvním odkazu ze strany uživatele portálu kompilace zdrojového kódu JSP stránky za účelem vytvoření instance Java servletu, což značně zpomaluje generování fragmentu výsledné WWW stránky v prostředí portálových technologií s využitím Java portletů.

- Z obdobných důvodů rovněž není využito technologie *Java Server Faces* (JSF – viz [9]) i když způsob generování prezentační vrstvy portálu respektuje mnohé ze zde známých a zavedených principů (např. práce s událostmi, využití komponent Java Beans apod.).

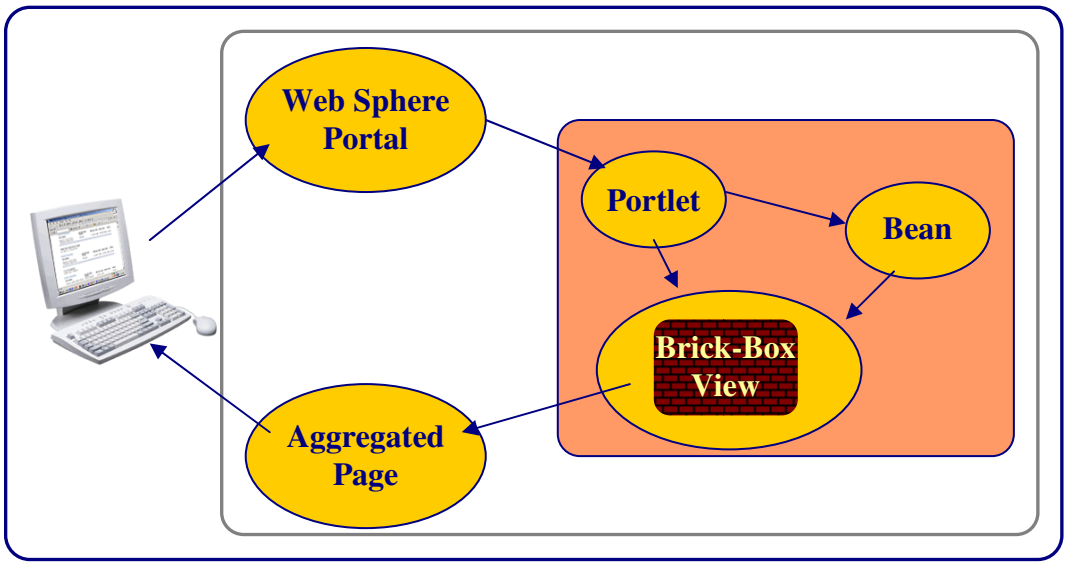


Obr. 1: Prezentační vrstva neveřejné část portálu

Pro vlastní implementaci prezentační vrstvy neveřejné části fakultního portálu je tedy využito speciálně pro tento účel vyvinutého návrhového vzoru *Brick-Box Model-View-Controller*, v němž v roli komponenty *controller* vystupuje instance Java portletu, komponenta *model* je reprezentována instancí třídy Enterprise Java Bean (příp. Java Bean)

provádějící interakci s databázovým systémem a komponenta *model* není v tomto případě reprezentována tradičním způsobem instancí stránky JSP, ale objektem splňujícím specifikace standardu pro Java Beans.

Specializovaný návrhový vzor *Brick-Box Model-View-Controller* a jeho funkcionality vyplynuly především z požadavku na rychlost generování fragmentu WWW stránky portálového prostředí. V důsledku komplikovanosti a rozsahu XHTML kódu výsledného fragmentu se ukázalo velice účelným ustavení programové cache paměti, která uchovává již vygenerované části jednotlivých fragmentů stránek s *konstantním obsahem* reprezentujících jednotlivé grafické komponenty (např. tlačítka, panely, menu apod.) příp. jejich **části** (záhlaví tabulek, tabulátory apod.). Tyto předem vygenerované elementy výsledného fragmentu WWW stránky s konstantním obsahem v rámci obecně dynamicky generované WWW stránky si lze představit jako cihly v budované zdi, které budou při její stavbě slepeny jistým pojivem. Tímto pojivem jsou *dynamicky generované části* fragmentu WWW stránky (např. obsah tabulek, rolovacích menu apod.), jejichž obsah produkovaný komponentou *model* návrhového vzoru musí být přesně uzpůsoben již vygenerovaným konstantním částem fragmentu stránky (viz obr. 2).



Obr. 2: Návrhový vzor Brick-Box Model-View-Controller

Praktická realizace fakultního portálového řešení s využitím návrhového vzoru *Brick-Box Model-View-Controller* prokázala jeho funkčnost při významné podpoře rychlosti generování fragmentů WWW stránek obsahující velice komplexní uživatelské rozhraní.

4. MODIFIKOVANÝ NÁVRHOVÝ VZOR BRICK-BOX MODEL-VIEW-CONTROLLER VE VÍCEVLÁKNOVÉM VÝPOČETNÍM PROSTŘEDÍ S VYUŽITÍM TECHNOLOGIE RTSJ

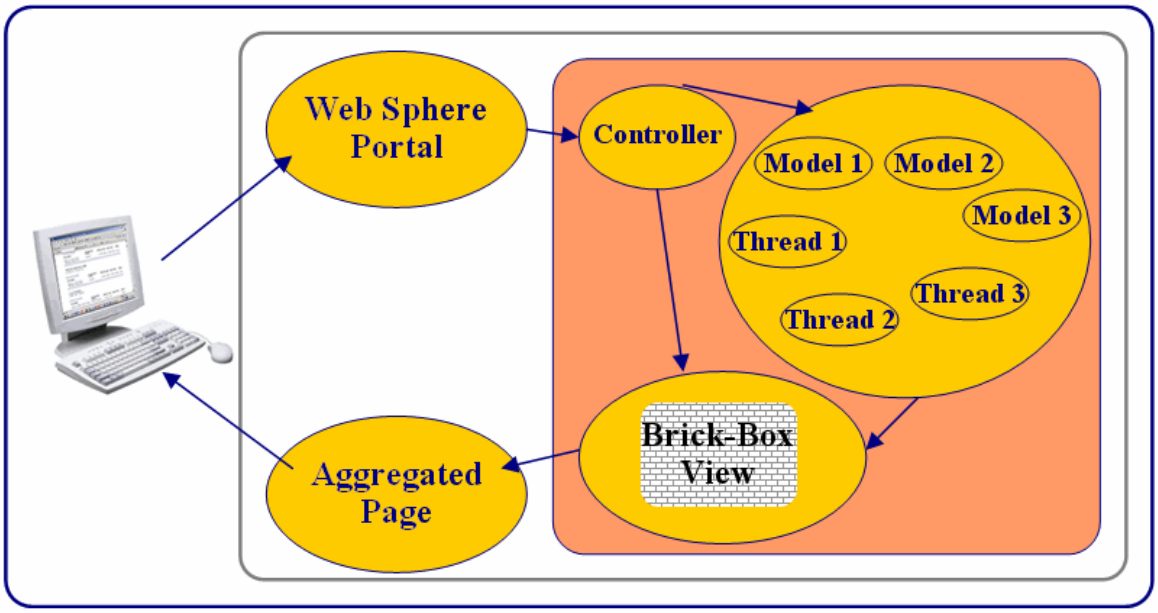
Vícevláknové výpočetní prostředí *Java Virtual Machine* a specifikace RTSJ umožňuje další modifikaci návrhového vzoru *Brick-Box Model-View-Controller* s cílem zvýšení efektivnosti vytváření výsledných fragmentů WWW stránek. Jedná se především o následující úpravy:

- Předem vygenerované části jednotlivých fragmentů stránek s *konstantním obsahem* reprezentující jednotlivé grafické komponenty lze uložit ve formě instancí tříd v paměťové oblasti reprezentované objektem `ImmortalMemory`. Tyto objekty pak existují po celou dobu běhu aplikace a přístup k nim nebude blokován programovým vláknem správce programové haldy,

- Komponenta *model* může být navržena a implementována s využitím technologie programových vláken reálného času, které umožní souběžné, příp. paralelní generování jednotlivých nezávislých částí dynamických fragmentů stránek. Tato programová vlákna budou pro svoji činnost využívat výhradně paměťové oblasti reprezentované objektem `ScopedMemory`, což při velkém počtu souběžně pracujících uživatelů portálového prostředí urychlí správu a fázi destrukce velkého počtu dočasných instancí tříd (zejména objektů reprezentujících výsledky dotazů vůči databázi).

Typický příklad takové situace je znázorněn na obr. 1, kde jsou dynamické části prezentační vrstvy portletu určeného k administraci dokumentů *Systému managementu jakosti EkF VŠB-TUO* složeny ze dvou nezávislých částí generovaných komponentou *model*, z nichž první je zobrazena ve formě hierarchie složek *Systému managementu jakosti* a druhá jako seznam dokumentů *Systému managementu jakosti* aktuální složky. Obě části mohou být generovány dvěma samostatnými programovými vlákny komunikujícími souběžně, příp. paralelně s příslušným databázovým systémem.

Schéma modifikovaného návrhového vzoru *Brick-Box Model-View-Controller*, v němž je při implementaci komponenty *model* využito několik programových vláken, je znázorněno na obr. 3.



Obr. 3: Návrhový vzor Brick-Box Model-View-Controller ve vícevláknovém prostředí

4. ZÁVĚR

Závěrem lze tedy konstatovat vysokou aktuálnost více než 20 let starého návrhového vzoru *Model-View-Controller* při návrhu a implementaci portálových řešení s vícevrstvou architekturou. Souběžné, příp. paralelní generování nezávislých částí fragmentů WW stránek s využitím programových vláken reálného času a nasazení knihoven technologie RTSJ může výrazným způsobem zefektivnit celý proces reakce na požadavek strany klienta.

LITERATURA

1. WELLINGS A.: Concurrent and Real-Time Programming in Java, Willey, 2004, ISBN 0-470-84437-X
2. http://jcp.org/en/jsr/detail?id=282
3. ABDELNUR A., HEPPER S.: Java Portlet Specification – Version 1.0 (JSR 168), Sun Microsystems, 2003
4. LINWOOD J., MINTER D: Building Portals with the Java Portlet API, Apress, 2004, ISBN 1-59059-284-0
5. http://java.sun.com/javase/
6. http://java.sun.com/javaee/
7. BURBECK S.: Applications Programming in Smalltalk-80: How To Use Model-View-Controller, 1992, ParcPlace, http://st-www.cs.uiuc.edu/users/smarch/st-docs/mvc.html
8. HALL M.: More Servlets and Java Server Pages, Sun Microsystems Press, 2002, ISBN 0-13-067614-4
9. http://java.sun.com/javaee/javaserverfaces/