Ondanks de uitvoerige set aan API's voor het ontwikkelen van gedistribueerde applicaties, biedt J2EE geen standaardoplossing voor een applicatiebrede sessiecontext. Een dergelijke context is wenselijk, omdat er in een J2EE-applicatie vaak metagegevens op meerdere plaatsen beschikbaar moeten zijn. Webapplicaties hebben bijvoorbeeld faciliteiten in de vorm van een http-sessie maar deze wordt niet automatisch overgedragen naar de EJB-laag.

Applicatiebrede sessiecontext: Rampart

Framework voor J2EE-applicaties

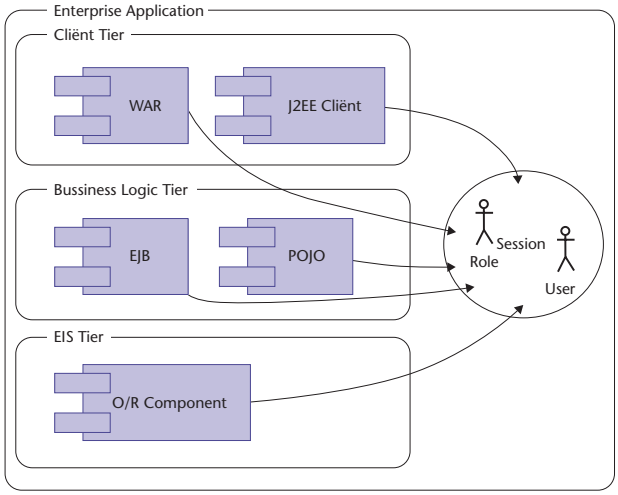
Om het toch voor elkaar te krijgen bouwen ontwikkelaars een eigen oplossing, waarbij programmeertrucs soms niet worden geschuwd. Dit leidt vaak tot slecht onderhoudbare en slecht uitbreidbare code. Aangezien het gebruik van een applicatiebrede sessiecontext in bijna alle J2EE-applicaties wenselijk is, biedt het Rampart framework een standaardoplossing.

In zijn artikel 'Create an application-wide user session', (JavaWorld). legt Kåre Kjelstrøm uit hoe het Rampart framework een standaard oplossing biedt. "Door middel van het combineren van code generatie met een vleugje aspect-oriented programming en een aantal design patterns, is het mogelijk een applicatiebrede context voor de verschillende componenten van een enterprise applicatie te bieden.", aldus Kjelstrøm.

Tijdens de ontwikkeling aan een online offertesysteem hebben we met deze sessie problematiek te maken gehad. Een probleem wat we uiteindelijk met het Rampart-framework hebben opgelost, is het kunnen gebruiken van account- en transactiegegevens (horend bij de gebruikerssessie) vanuit alle applicatieonderdelen. In dit artikel beschrijven we het Rampart framework hebben gebruikt, en waarom wij hier tevens een aantal uitbreidingen op hebben gemaakt.

APPLICATIEBREDE SESSIECONTEXT Enterprise applicaties bestaan uit meerdere lagen. Meestal zijn ze opgebouwd uit een 3 lagen (tier) structuur; een client tier, business tier en EIS-tier. In deze lagen bevinden zich verschillende componenten welke samen de eigenlijke applicatie vormen. In de meeste J2EE-applicaties is er bijvoorbeeld sprake van een autorisatie en authenticatiemechanisme. Een gebruiker moet inloggen in de applicatie en kan dan op basis van de rol die hij of zij heeft gebruik maken van bepaalde delen van de applicatie. Tijdens het inloggen worden de gebruikersgegevens opgehaald en gevalideerd. Deze gegevens moeten binnen de gehele applicatie bekend zijn.

Met behulp van een applicatiebrede sessiecontext kunnen de sessiegegevens van de gebruiker (zoals userId, transactie id, rol et cetera) van een applicatie op een eenduidige manier worden vastgehouden. De sessiecon-



FIGUUR 1. De behoefte aan een applicatiebrede sessie.

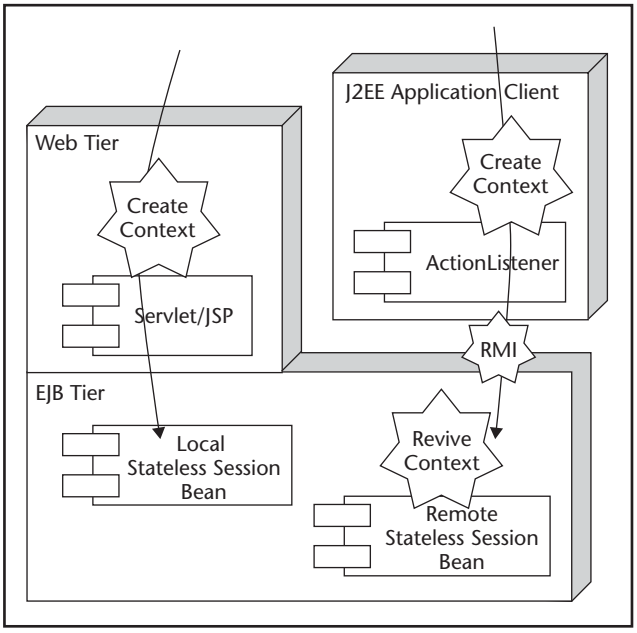
text representeert de toestand van de gebruiker, terwijl deze verschillende componenten van eenzelfde applicatie gebruikt. Figuur 1 illustreert de behoefte aan de toegang tot een gedeelde sessie over de verschillende soorten componenten, met, in dit geval, gebruiker en rol informatie.

FRAMEWORK In deze paragraaf worden de belangrijkste punten van het artikel van Kjelstrøm samengevat. Voor een goed begrip is het echter verstandig het JavaWorld-artikel te lezen. Een J2EE-applicatie bevat vele componenten die onderling middels standaarden (JNDI, JAAS, JTX) met elkaar samenwerken. Functioneel gezien kunnen deze componenten alle acties uitvoeren, wat echter ontbreekt, is een overkoepelende gebruikers sessie die door de verschillende componenten (bijvoorbeeld Web, EJB en Swing) kan worden gebruikt.

Het vasthouden en doorgeven van het RequestContext object kan binnen de J2EE-client zelf worden gedaan

Het Rampart-framework is ontwikkeld door Silverbullet uit Denemarken en is een open source framework die een implementatie biedt van een applicatie brede sessie context. Rampart definieert een structuur voor de request- sessie context en biedt een infrastructuur om deze contexten te beheren.

Rampart werkt met een centraal object, de RequestContext, waarmee gedurende de interactie met het systeem de gegevens van de gebruiker worden bijgehouden. De sessie-informatie wordt opgeslagen in een ApplicationSession object. Voor ieder request wordt deze vervoerd in het RequestContext object. De RequestContext wordt beheerd door de een RequestContextFactory.



FIGUUR 2. Creëren en ophalen van de RequestContext.

De strategie van het framework is om vanuit de (bestaande) J2EE-componenten uit verschillende lagen (Web, EJB) van de applicatie-architectuur toegang te hebben tot de RequestContext. Op deze wijze wordt een overkoepelende sessie-context verkregen.

Rampart gebruikt hiervoor een aantal verschillende technieken (zie het JavaWorld artikel voor details). Voor J2EE-clients (Swing) wordt een RequestContext gegenereerd voor iedere actie (bijvoorbeeld een button-click) alvorens de achterliggende logica (middels de ActionListener interface) wordt aangeroepen. Dit wordt gedaan door gebruik te maken van Aspect Oriented programming. Voor de webapplicaties wordt er een ServletFilter gebruikt voor het toevoegen van de RequestContext aan het request van de cliënt naar de server.

Het vasthouden en doorgeven van het RequestContext object kan binnen de J2EE-client zelf worden gedaan. Voor webapplicaties wordt hiervoor de HTTP sessie gebruikt. Er wordt gebruik gemaakt van een RequestContextFactory voor het beheren van de RequestContext instanties aan de cliënt kant. De RequestContextFactory wordt beheerd vanuit de Spring container. Het doorgeven aan de EJB laag wordt gedaan met behulp van het Business Delegate design pattern (`Sun Blueprints Core J2EE Patterns Catalog`). De implementatie van de Business Delegate valt buiten de scope van Rampart en moet door de applicatie ontwikkelaar zelf gedaan worden. Rampart biedt een Proxy class, de RequestContextPropagationInterceptor, welke door de Business Delegate wordt gebruikt. De Proxy vangt het request af en als er een RequestContext beschikbaar is dan wordt deze doorgegeven naar de EJB-componenten.

ONLINE OFFERTESYSTEEM Onlangs hebben we voor een van onze klanten een Enterprise Applicatie ontworpen en geïmplementeerd waarmee verschillende diensten vanuit de backoffice applicaties van de klant voor gebruikers beschikbaar worden gesteld. Gebruikers kunnen via het internet zelf berekeningen en offertes aanmaken een aanvragen doen. Tevens worden er met behulp van een Java Swing client verschillende diensten geboden. De applicatie is aangesloten op een aantal back-end systemen, waarin de daadwerkelijke diensten worden uitgevoerd en de gegevens worden verwerkt. Daarnaast is er nog een Scheduler component waarin een aantal administratieve taken zijn gedefinieerd.

Binnen deze applicatie wordt de functionaliteit voor de gebruiker beschikbaar gesteld door middel van een

Swing-cliënt-applicatie en een aantal web-applicaties. Deze front-end applicaties kunnen via de J2EE enterprise-beans applicatie verschillende back-offices benaderen. Ook kan de gebruiker, na een keer te zijn ingelogd, meerdere webapplicaties benaderen en als het ware overstappen van de ene naar de andere webapplicatie.

Het Scheduler-project draait los van de andere applicaties en maakt gebruik van de functionaliteit van de Enterprise Applicatie. Om gebruik te kunnen maken van de Enterprise Applicatie moeten de taken van het Scheduler project inloggen, deze gegevens worden via de RequestContext meegegeven aan de Business Delegate.

Uitbreidingen en aanpassingen Rampart biedt een goede oplossing voor het sessiecontext probleem. In complexere applicaties kan het echter zo zijn dat de door Rampart geboden oplossing niet afdoende is. In deze paragraaf bespreken we vier aanpassingen en uitbreidingen die we voor onze applicatie hebben gedaan.

1. Rampart gaat uit van eenrichtingsverkeer. Aan de cliënt-kant wordt de ApplicationSession gepersisteerd en bij elke request meegestuurd naar de server. In onze situatie wilden we ook op de server de inhoud van de ApplicationSession kunnen manipuleren.
2. Daarnaast moest het doorgeven van de RequestContext bij de overgang van de ene webapplicatie naar de andere worden gefaciliteerd.
3. Verder wordt in onze applicatie gebruik gemaakt van een aantal Message Driven Beans en is er een scheduler-project met daarin een serie taken. Deze hebben ook een RequestContext en een ApplicationSession nodig.
4. Als laatste wordt er in onze applicatie gebruik gemaakt van AspectJ in plaats van AspectWerkz.

Response-object Om op de server de inhoud van de ApplicationSession te kunnen manipuleren en terug te geven aan de clients hebben we een Response-object toegevoegd. Hiervoor hebben we een resultaatobject gedefinieerd. Deze bevat het resultaat van de aangeroepen methode van de server plus het aangevulde ApplicationSession object. In de Interceptor wordt de teruggekregen RequestContext opgeslagen in de Factory en aan het einde van het ServletFilter en de Aspect wordt de teruggekregen ApplicationSession bijgewerkt in de ApplicationSessionHolder.

De ApplicationSessionHolder is het Rampart-object waarbinnen de RequestContext persistent wordt opgeslagen en bewaard.

```
public class RequestContextPropagationInterce
ptor implements InvocationHandler {
    private RequestContextInterceptor
interceptor;
        private RequestContextResponse
        response = null;

public RequestContextPropagationInterceptor(R
equestContextInterceptor interceptor) {
        this.interceptor = interceptor;
    }
    public Object invoke(Object proxy, Method
method, Object[] args) throws Throwable {
        if ("toString".equals(method.
        getName())) {
            return getClass() + ":" +
            interceptor.toString();
        }
        RequestContextResponse response =
getRequestContextResponse();
        Object result;
        try {
            RequestContextFactory
requestContextFactory =
ConfigHelper.getRequestContextFactory();
            RequestContext requestContext =
requestContextFactory.getRequestContext();
            if (args == null || args.length
== 0) {
                                response =
(RequestContextResponse)interceptor.
exec(method.getName(), args, new String[]{},
requestContext);
            } else {
                String[] argTypes =
RequestContextUtil.getNames
(method.getParameterTypes());
                response = (RequestContext
Response)interceptor.exec(method.getName(),
args, argTypes, requestContext);
            }
       requestContextFactory.setRequestContext
       (response.getRequestContext());
            return response.getResult();
// null if void
        } catch (RemoteException ex) {
            throw ex;
        } catch (InvocationTargetException
ex) {
            throw ex.getTargetException();
        }
    }

        private RequestContextResponse
getRequestContextResponse(){
                if(null == response){
                        response = new
               RequestContextResponse();
                }
                return response;
        }
}
```

Context tabel voor ondersteuning overgang tussen webapplicaties Binnen de applicatie worden meerdere webclients gebruikt. Een gebruiker kan de functionaliteit van de verschillende webclients gebruiken. Het is dus mogelijk om van de ene webclient over te gaan naar de andere. Bij deze overgang ontstaat een nieuwe instantie van de HttpSessie, waarbij alleen het sessie ID wordt overgenomen. Alle overige informatie die op de sessie stond, waaronder de ApplicationSession, wordt niet meegenomen en is niet meer beschikbaar. Daarom is er voor deze overgang in de J2EE serverapplicatie een RequestContextLookup gemaakt. De RequestContextLookup is een opslag voor de Request context en wordt gebruikt om de RequestContext door te geven bij de overgang van de ene webapplicatie naar een andere. Na het inloggen in wordt de RequestContext in de RequestContextLookup gezet, bij overgang naar een ander Web client wordt hij hier uitgelezen en afgehaald.

Message Driven Beans en Scheduler project Naast de WebApplicaties en de J2EE Cliënt worden er een aantal Message Driven Beans gebruikt. Deze Beans maken gebruikt van een Business Delegate en hebben ook een ApplicationSession en een RequestContext nodig.

```
public aspect CreateRequestContextAspect {
        pointcut mdbOnMessage() :
execution(public void onMessage(..)) &&
this(javax.ejb.MessageDrivenBean);

        before() : mdbOnMessage() {
                RequestContextFactory
requestContextFactory = ConfigHelper.
getRequestContextFactory();
                RequestContext context =
requestContextFactory.create();
                context.setApplicationSession
(new DefaultApplicationSession());
                requestContextFactory.setReq
uestContext(context);
        }
        after() : mdbOnMessage() {
                ConfigHelper.getRequest
                ContextFactory().delete();
        }
}
```

Het Scheduler project maakt gebruik van de functionaliteit van de Enterprise Applicatie. De functionaliteit van de Enterprise Applicatie is voor de taken van de Scheduler beschikbaar via de Business Delegate. Daarom hebben de taken een ApplicationSession en een RequestContext nodig.

```
Object around(): addRequestContext(){
        Object result = null;
        try {
              RequestContext request Context
           = requestContextFactory.create();
                requestContext.setApplication
   Session(new DefaultApplicationSession());
                result = proceed();
        } finally {
              requestContextFactory.delete();
        }
        return result;
}
```

Voor zowel de Message Driven Beans als de taken in het Scheduler project hebben we Aspects geschreven.

AspectWerkz Rampart gebruikt AspectWerkz, binnen de applicatie van onze klant wordt AspectJ gebruikt. Hiervoor moesten de aspects van Rampart herschreven worden. Daarnaast is binnen onze J2EE clientapplicatie een oplossing bedacht om het bevriezen van schermen tijdens een request te voorkomen. Binnen de actionPerformed van de ActionListeners wordt in onze applicatie een SwingWorker gebruikt. Dit betekent dat je overstapt naar een andere thread. Hierdoor werken binnen onze applicatie de standaard aspects voor de J2EE clientapplicatie van Rampart niet en moeten voor de aspects de pointcuts anders worden gedefinieerd.

Discussie Tijdens de implementatie van ons systeem kwam een aantal mogelijke alternatieven naar voren;

JAAS (Java Authorization and Authentication Services) biedt sessie context middels de zgn. UserPrincipal. In ons geval konden we in verband met klant-specifieke serverinstellingen geen gebruik maken van JAAS. Spring 2.0 biedt ook een oplossing voor de applicatiebrede sessiecontext in de vorm van een abstractielaag waarvoor de applicatieontwikkelaar zelf een implementatie kan schrijven. Tijdens de ontwikkeling van het online offertesysteem was Spring 2.0 nog niet gereleased, daarom konden we deze oplossing niet gebruiken. Voor zover bekend bieden Java5 en EJB 3.0 geen directe oplossing voor de applicatiebrede sessiecontext.

Tijdens de implementatie van ons systeem kwam een aantal mogelijke alternatieven naar voren

Het Rampart framework geeft de RequestContext als parameter mee aan de EJB. Om dit voor elkaar te krijgen biedt Rampart een speciale EJB-methode, namelijk de exec(). Alle EJB calls worden gerouteerd over deze exec methode, met als gevolg dat de transactional settings en security settings voor je overige EJB-methodes geen effect hebben! Dit was in onze situatie geen probleem, maar is wel iets waarvan je je bewust moet zijn.

Conclusie Binnen een J2EE Enterprise Applicatie wil je vanuit alle componenten de beschikking hebben over de administratieve sessie-informatie, zoals een UserId en de rol. Rampart is een framework welke met behulp van AOP, het Business Delegate pattern en Spring hiervoor een oplossing biedt. Voor sommige applicaties kan het echter zijn dat de door Rampart geboden functionaliteit niet afdoende is. Aangezien Rampart een open source framework is kan de code zonder problemen worden aangepast en uitgebreid. Het Rampart framework is een krachtig en eenvoudig te gebruiken framework. Het gebruikt standaardtechnieken en tools om een gedeelde sessie aan de componenten te bieden. Het is momenteel een van de weinige, zo niet het enige, framework dat deze dienst biedt.

Referenties

- Kåre Kjelstrøm, "Create an application-wide user session for J2EE", JavaWorld http://www.javaworld.com/javaworld/jw-03-2005/jw-0314-usersession.html
- De makers van het Rampart framework www.silverbullet.dk
- Core J2EE Patterns - Business Delegate: http://java.sun.com/blueprints/corej2eepatterns/Patterns/BusinessDelegate.html

Rein Benno de Jager en Wico Mulder zijn werkzaam bij Logica CMG.