Faculteit Toegepaste Wetenschappen
Vakgroep Informatietechnologie
Voorzitter: Prof. Dr. Ir. P. LAGASSE

Java Web services en security voor e-procurement

door

Thomas VAN PARYS

Promotoren: Prof. Dr. Ir. E. LAERMANS en Prof. Dr. Ir. F. GIELEN
Scriptiebegeleiders: Ir. W. HAERICK en Ir. S. VAN HOECKE

Scriptie ingediend tot het behalen van de academische graad van
licentiaat informatica, optie software-ontwikkeling

Academiejaar 2004–2005

Voorwoord

Deze thesis is het product van twee zeer drukke en stresserende zomermaanden van schrijf- en programmeerwerk, maar vooral van eindeloos zoekwerk, trial and error. Mijn dank gaat hier dus vooral uit naar mijn vriendin Veroniek Van Driessche voor de steun en het geduld tijdens deze periode, voor het inruilen van de geplande trektochtjes tegen vele uren achter de laptop en uiteindelijk voor het herlezen van deze scriptie.

Daarnaast ook een woordje van dank voor mijn begeleiders en promotoren die me alsnog de kans gegeven hebben dit project tot een goed einde te brengen.

Thomas Van Parys, augustus 2005

Toelating tot bruikleen

Java Web services en security voor e-procurement

door

Thomas VAN PARYS

Scriptie ingediend tot het behalen van de academische graad van licentiaat informatica, optie software-ontwikkeling

Academiejaar 2004–2005

Promotoren: Prof. Dr. Ir. E. LAERMANS en Prof. Dr. Ir. F. GIELEN
Scriptiebegeleiders: Ir. W. HAERICK en Ir. S. VAN HOECKE

Faculteit Toegepaste Wetenschappen
Universiteit Gent

Vakgroep Informatietechnologie
Voorzitter: Prof. Dr. Ir. P. LAGASSE

Samenvatting

In deze thesis wordt onderzocht hoe procurement van ziekenhuizen te automatiseren en dit proces te laten verlopen via Java Web services. Naast de eigenlijke functionaliteit word de nadruk gelegd op de beveiliging van de applicatie. Om dit te bereiken word gebruik gemaakt van XML and Web Services Security (XWS-Security), de Sun-implementatie van Web Service Security (WSS), en van encryptie en authenticatie via SSL.

Trefwoorden

Java, e-health, e-procurement, J2EE, Web services, JAX-RPC, XWS-Security, SSL

Inhoudsopgave

1 Inleiding en doel **1**

2 Analyse en architectuurstudie **3**

2.1 Vision Statement . . . 3

2.1.1 Mission Statement . . . 3

2.1.2 Target end User . . . 4

2.1.3 Measure of Success . . . 4

2.1.4 Key Features . . . 4

2.2 Use Cases . . . 5

2.2.1 Ziekenhuis . . . 5

2.2.2 Leverancier . . . 10

2.2.3 Procurement Administrator . . . 11

2.3 Sequentiediagram . . . 11

2.4 Architectuur . . . 12

3 Web services **17**

3.1 Web services algemeen . . . 17

3.2 SSL vs. WSS . . . 19

3.3 JAX-RPC . . . 22

4 Security **24**

4.1 Threats and solutions . . . 24

4.1.1 Opstelling . . . 24

4.1.2 Verificatie . . . 25

4.1.3 Vertrouwelijkheid . . . 26

4.1.4 Berichtintegriteit . . . 26

4.1.5 Niet-bestrijdbaarheid . . . 27
4.1.6 Bevestiging . . . 28
4.1.7 Replay-attacks . . . 29
4.1.8 Gebruikersbeheer . . . 29
4.1.9 Administratie en registratie . . . 30
4.1.10 Andere gaten in de beveiliging . . . 30
4.1.11 Conclusie . . . 31
4.2 Technologieën . . . 32

5 Implementatie 34
5.1 Installatie . . . 34
5.1.1 Platform . . . 34
5.1.2 Builden van de procurement-applicatie . . . 35
5.2 Databank . . . 37
5.3 Enterprise JavaBeans . . . 39
5.3.1 Entity Beans . . . 39
5.3.2 Session Beans . . . 42
5.4 WebServices: JAX-RPC . . . 43
5.5 Architectuur client . . . 46
5.6 Security . . . 47
5.6.1 Certificaten . . . 48
5.6.2 SSL . . . 49
5.6.3 XWS-Security . . . 51
5.6.4 MD5 . . . 54
5.7 Ontwikkelingsomgeving . . . 55
5.7.1 Programmeren . . . 55
5.7.2 Modelleren . . . 56
5.7.3 Scriptie . . . 56

6 Verder onderzoek en conclusie 57

Hoofdstuk 1

Inleiding en doel

De aankopen van ziekenhuizen worden vandaag nog steeds op de klassieke manier geregeld. Wanneer de aankoopdienst van een ziekenhuis beslist dat er nood is aan een bepaald product, worden de verschillende leveranciers van dit product gecontacteerd. Zij krijgen de nodige voorwaarden en vereisten toegestuurd (telefonisch, per post, of in het beste of snelste geval per e-mail) en krijgen dan de tijd om hierop te reageren. Uit de brieven of e-mails die binnen een vooropgestelde periode terug zijn binnengekomen bij de aankoopdienst, wordt vervolgens de offerte met het beste voorstel geselecteerd. In het geval dat geen enkele offerte zou voldoen aan de voorwaarden, kunnen alle bedrijven nogmaals gecontacteerd worden voor een 'volgende ronde', waarop dan weer antwoorden met betere voorstellen kunnen gestuurd worden.

Het spreekt voor zich dat deze methode, behalve behoorlijk onoverzichtelijk, ook zeer traag en arbeidsintensief kan zijn. Er is kans op menselijke foutjes, verloren inzendingen,... Bovendien is er het gevaar voor een zeker bevoordelen van bedrijven die bvb. snel reageren, er in hun correspondentie een betere schrijfstijl op nahouden of extra cadeautjes aan hun offerte koppelen, terwijl het in het voordeel van het ziekenhuis is om het goedkoopste en/of kwalitatief beste product te kiezen.

In deze thesis wordt een proof of concept ontworpen van een applicatie die deze aankoponderhandelingen (Eng.: procurement) via elektronische weg kan laten verlopen (vandaar e-procurement). Hiervoor werd geopteerd om gebruik te maken van Java Web services.

Bij het ontwikkelen werd nadruk gelegd op de beveiliging van de applicatie: het is noodzakelijk om de biedingen zowel voor het ziekenhuis als voor de leveranciers geheim te houden tot na het verstrijken van een op voorhand vastgelegde periode, bedrijven moeten verhinderd worden om in elkaars naam te bieden, het moet onmogelijk zijn voor een partij om te ontkennen dat hij

een bepaalde handeling heeft verricht, enz...

Deze thesis moet dus bewijzen dat het mogelijk is om d.m.v. Java Web services op een betrouwbare en veilige manier aan reverse auctioning te kunnen doen.

Hoofdstuk 2

Analyse en architectuurstudie

Een interessant en leerrijk aspect aan deze thesis is dat hij werd opgevat als een werkelijk software-project en dat dus alle stadia van het ontwerp werden doorlopen. De grote lijnen van de applicatie werden getekend tijdens de analyse-fase, waar werd bepaald wat de gebruiker zou kunnen doen en hoe dit moest gebeuren. Ook werden alle beveiligingsproblemen overlopen om tot een security policy (zie § 4.1) te kunnen komen. In een latere fase werd dan de architectuur ontworpen, gevolgd door het eigenlijke implementeren en testen.

2.1 Vision Statement

Het Vision Statement geeft weer hoe de applicatie bij het aanvatten van het project werd gezien. De globale opzet, de beoogde gebruikers, de noodzakelijke features en de basisfunctionaliteit worden hierin beschreven. Het oorspronkelijke Vision Statement bevatte ook een planning waar mijlpalen een deadline toegekend kregen die hier nu echter niet meer relevant zijn. Hoewel aan de basisvereisten voldaan werd, zijn sommige aspecten licht gewijzigd in de loop van het project en werden de beveiligingsvereisten veel grondiger geherdefinieerd.

2.1.1 Mission Statement

Het Mission Statement drukt het oorspronkelijk beoogde doel van het project uit: het ontwerpen en implementeren van een e-procurement-systeem met de nadruk op het gebruik van de applicatie in de e-health. Het is de bedoeling om, in Java, Web services uit te werken voor een systeem met als drie voornaamste componenten: de koper (in dit geval het ziekenhuis), de verkopers en de e-procurement-server, die als derde, onafhankelijke partij de onderhandelingen regelt. Het

systeem moet in elk geval zo zijn opgebouwd dat alle transacties beveiligd verlopen en worden gelogged, zodat de e-procurement-applicatie kan gebruikt worden als een veilig en betrouwbaar medium om aan reverse auctioning te doen.

2.1.2 Target end User

Dit onderdeel definieert de beoogde gebruikers van de applicatie: aan de ene kant ziekenhuizen of overheidsinstellingen die regelmatig te maken krijgen met grote aankopen die zo voordelig mogelijk moeten worden afgesloten, of eventueel aan andere selectiecriteria moeten voldoen. Aan de andere kant zijn er uiteraard de verkopers of fabrikanten die meedingen naar de aanbesteding.

2.1.3 Measure of Success

- Beveiliging
  - Enkel geautoriseerde gebruikers mogen een onderhandeling kunnen starten of mogen kunnen bieden op een aanbesteding.
  - Een bod mag niet kunnen gewijzigd worden eenmaal het is geplaatst. Het moet dus onmogelijk zijn het bod te 'onderscheppen' en te wijzigen wanneer het wordt verzonden. Eenmaal het bod de e-procurement-server heeft bereikt mag het uiteraard ook niet meer wijzigbaar zijn.
  - De verschillende mededingers mogen de concurrerende biedingen niet te zien krijgen indien dit niet de bedoeling is.
  - Van alle transacties moet er een bewijs (log) voorhanden zijn.
- Uitbreidbaarheid
  - De applicatie moet zo zijn opgebouwd dat hij eenvoudig uitbreidbaar is (met extra features zoals bvb. het gebruik van business regels bij het automatisch bieden of selecteren van de winnaar, verder afhandelen van de bestelling na het selecteren van de winnaar,...).

2.1.4 Key Features

- Koper (ziekenhuis)
  - Registreren bij de e-procurement-server.

  - Starten van de onderhandelingen door een aanbesteding uit te schrijven (product, hoeveelheid, maximumprijs, duur van het aanbod, andere voorwaarden,...)
  - Doornemen van de inkomende biedingen (al dan niet met vermelding van de naam van het bedrijf).
  - Selecteren van de 'winnaar', de verkoper waarvan de offerte voldoet aan de selectiecriteria en dus de aanbesteding krijgt. Alle mededingers worden van het resultaat op de hoogte gebracht.

- Verkopers (leveranciers ziekenhuis)
  - Registreren bij de e-procurement-server.
  - Ontvangen en bekijken aanbestedingen.
  - Al dan niet automatisch reageren op een aanbesteding door indienen offertes.
  - Indien dit is toegestaan: doornemen van de concurrerende biedingen (al dan niet met vermelding van de naam van het bedrijf). Anders: opvragen van laagste tegenbod.
  - In een eventuele tweede ronde het bod aanpassen.
  - Ontvangen van de melding of de offerte al dan niet is aanvaard (hierna verder afhandelen van de bestelling ligt buiten het bestek van deze thesis).

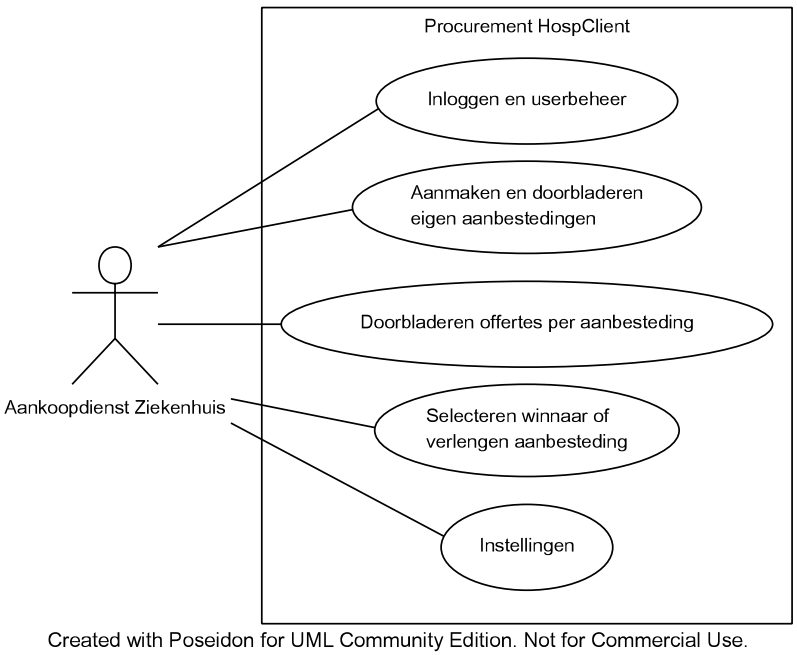
2.2 Use Cases

D.m.v Use Cases worden interacties tussen de gebruiker en het systeem schematisch weergegeven. Daardoor kan de ontwerper een beeld krijgen van wat er moet gebeuren en krijgt de klant (die in normale omstandigheden betrokken wordt in het proces) een beter zicht op de mogelijkheden van de applicatie en ziet hij of deze overeenstemmen met zijn eigen verwachtingen.

2.2.1 Ziekenhuis

Het is de bedoeling dat met de ziekenhuis-client de aankoopdienst van het ziekenhuis al zijn aankopen kan beheren en voor elke aanbesteding de procurement kan monitoren. Voor deze thesis werd de functionaliteit voor een eenvoudig scenario uitgewerkt.

Allereerst moet de gebruiker de instellingen kunnen bepalen: de URL van de server kan worden opgegeven, net zoals de locatie van zijn eigen private key en het certificaat van de

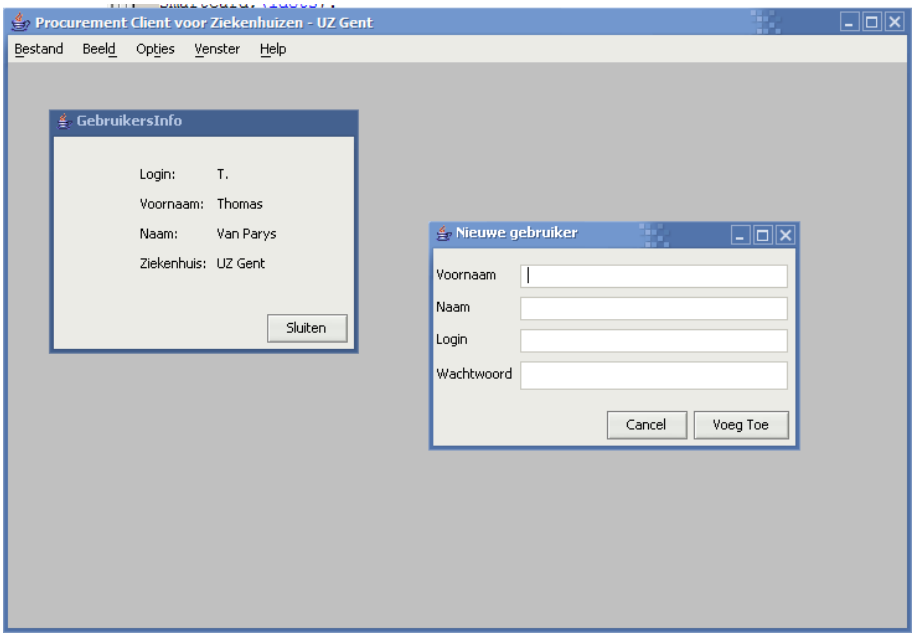


Figuur 2.1: Use Cases voor de aankoopdienst van het ziekenhuis

server. Deze laatsten kunnen op de harde schijf staan, maar kunnen in veiliger omstandigheden op een andere gegevensdrager worden bijgehouden (USB-stick, diskette, SmartCard,...).

Eens deze instellingen in orde zijn, kan de gebruiker inloggen met zijn persoonlijke loginnaam en wachtwoord. Bij het registreren van een nieuw bedrijf krijgt de klant een account mee voor één gebruiker. Die 'default'-gebruiker kan, indien nodig, naar hartelust nieuwe gebruikers aanmaken en een kopie van de certificaten geven. Elke nieuwe gebruiker wordt automatisch gekoppeld aan hetzelfde bedrijf. Na het inloggen verschijnt de naam van het bedrijf in de titelbalk en kunnen de gebruikersgegevens bekeken worden (Fig. 2.2).

Eens de gebruiker is ingelogged, krijgt hij toegang tot de andere functies. Er kan een nieuwe aanbesteding worden aangemaakt, waarbij het product omschreven wordt en er een gevraagd aantal en een maximumprijs kan worden opgegegeven (Fig. 2.3). Die maximumprijs is niet verplicht, maar kan dienen om leveranciers een startprijs te geven om het opbieden te beginnen. Dit is enkel nuttig in het geval er meerdere bied-rondes worden gehouden en wanneer de deelnemers elkaars tegenboden kunnen zien (het typische geval van reverse auctioning). Leveranciers die meedingen kunnen in dat geval niet boven deze maximumprijs beginnen (de prijs slaat op de volledige bestelling, niet per stuk). De applicatie is momenteel echter zo ingesteld dat elk bedrijf slechts één bod per ronde kan uitbrengen, er slechts twee rondes zijn, en leveranciers nooit



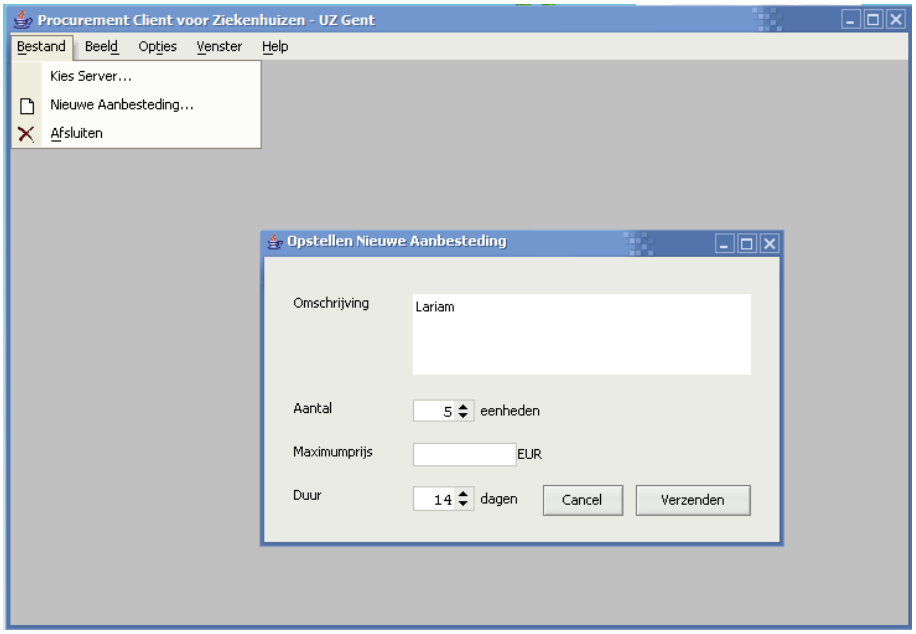
Figuur 2.2: Aanmaken nieuwe gebruiker

elkaars bod te zien krijgen. In dit geval is het natuurlijk niet aan te raden om als ziekenhuis een startbod, een budget, op te geven, aangezien alle geboden bedragen dan ook zo dicht mogelijk tegen die bovengrens zullen liggen. Het maximumprijs-veld wordt dus opengelaten zodat elk bedrijf onafhankelijk zijn offerte kan opstellen.

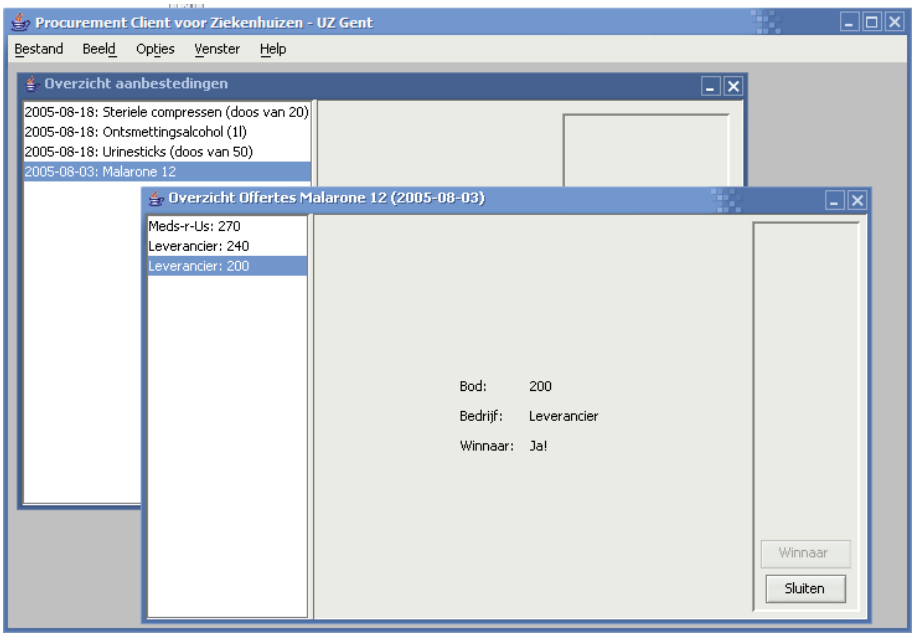
Verder kan worden opgegeven hoeveel dagen een ronde duurt, dus hoeveel tijd de leveranciers krijgen om te antwoorden.

De gebruiker kan altijd een overzicht raadplegen van alle aanbestedingen die zijn bedrijf heeft uitgeschreven. Van elke aanbesteding verschijnen de eigenschappen van zodra je erop klikt, samen met de start- en de einddatum ervan. Eens de periode verlopen is, wordt de knop 'Toon Offertes' beschikbaar. Nu kan per aanbesteding een lijst van de binnengekomen offertes worden bekeken (Fig. 2.4). Eén offerte kan worden uitgekozen die de aanbesteding 'wint'. Als er geen enkel bod laag genoeg is, kan de aanbesteding worden verlengd voor een opgegeven aantal dagen (Fig. 2.5). Met de huidige instellingen kan dit verlengen zoals gezegd slechts eenmalig. De offertes kunnen bij een volgende ronde weer niet meer worden bekeken totdat ook de nieuwe periode is verlopen.

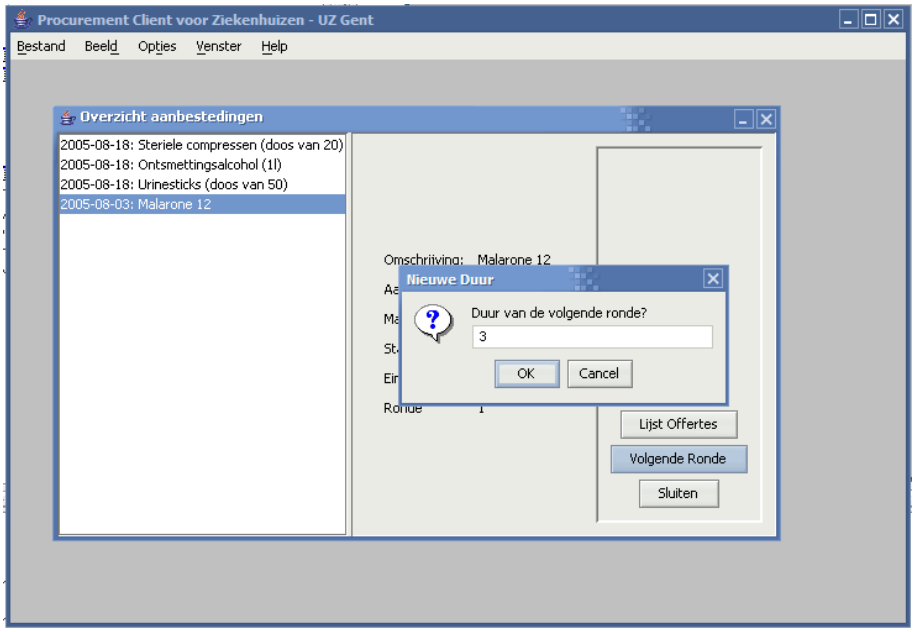
Bij het uitloggen worden alle vensters gesloten en worden de gebruikers- en bedrijfsgegevens uit het geheugen gewist.



Figuur 2.3: Aanmaken nieuwe aanbesteding

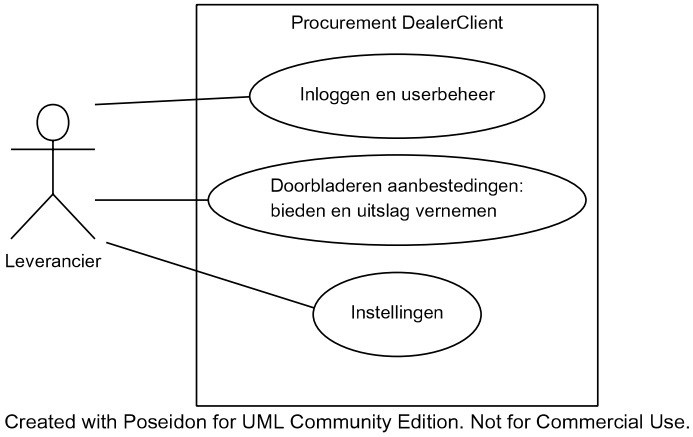


Figuur 2.4: Een lijst met offertes per aanbesteding



Figuur 2.5: Starten van een volgende ronde

2.2.2 Leverancier

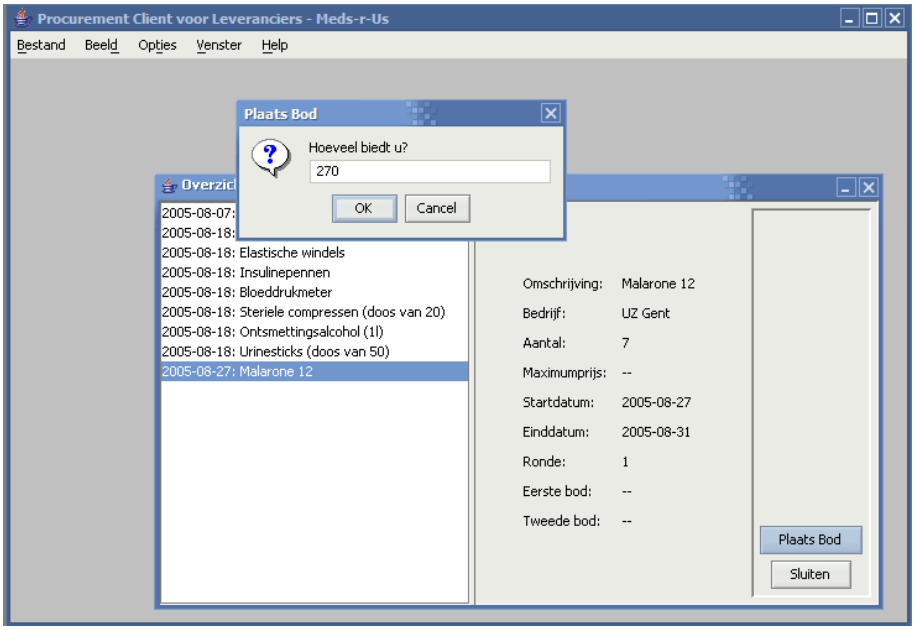


Figuur 2.6: Use Cases voor de leverancier

De client voor de leveranciers lijkt sterk op die voor de ziekenhuizen. Veel klassen van de ziekenhuis-client werden overigens gewoon gekopieerd en licht gewijzigd om te gebruiken in de leverancier-client. Andere werden zo geschreven dat ze zonder wijzigingen in beide clients konden worden opgenomen.

Leveranciers moeten via hun client alle informatie over aanbestedingen kunnen ontvangen en moeten hierop hun bod kunnen uitbrengen.

Het gebruikersbeheer bij de leveranciers is identiek aan dat voor de ziekenhuizen. Eenmaal ingelogged kan een leverancier alle uitgeschreven aanbestedingen (dus van de verschillende ziekenhuizen) te zien krijgen. Per aanbesteding, en per ronde, kan één maal worden geboden (Fig. 2.7). Indien er een maximumprijs werd opgegeven, moet het bod hier onder liggen. Wanneer de periode voor een aanbesteding verlopen is, wordt het onmogelijk om nog te bieden. Wanneer de aankoopdienst van het ziekenhuis beslist dat er een volgende ronde komt (er zijn twee rondes mogelijk met de huidge instellingen), dan kan de leverancier, binnen de opgegeven periode, nog eens bieden. Bij een bod dat door het ziekenhuis werd uitgekozen als laagste of aantrekkelijkste bod, verschijnt een duidelijk 'WINNAAR!'. Afhandelen van de aankoop zou een mooie uitbreiding van de applicatie zijn, maar werd niet ontworpen. De leverancier dient nu dus zelf contact op te nemen met het ziekenhuis om de verdere levering en betaling te bespreken.



Figuur 2.7: Bieden op een aanbesteding

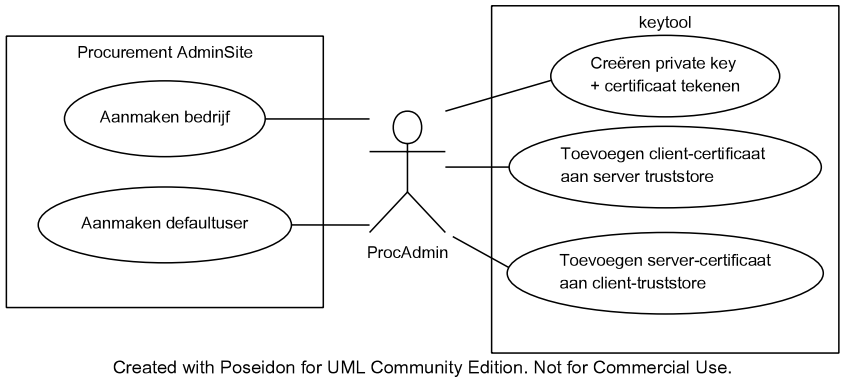
2.2.3 Procurement Administrator

Zoals hierboven al duidelijk is geworden, kunnen bedrijven (zowel ziekenhuizen als leveranciers) zich om veiligheidsredenen niet zelf inschrijven bij de procurement-server. Ze dienen hiervoor contact op te nemen met de adminstrator van de procurement-service. Nadat de identiteit van de nieuwe klant grondig werd geverifieerd, wordt via een JSP-pagina (Fig. 2.9) het nieuwe bedrijf toegevoegd, samen met een default-gebruiker. Het bedrijf kan dan een eerste maal inloggen met de login van deze gebruiker en dan zelf beslissen welke personeelsleden een eigen account krijgen en die vervolgens aanmaken.

De adminstrator dient ook de sleutels en certificaten met een aparte tool aan te maken en geeft die mee op een gegevensdrager (USB-stick, CD, diskette, SmartCard,...). Het is dan de keuze aan de klant om deze certificaten op de pc's te installeren of om er eventuele kopieën van te maken.

2.3 Sequentiediagram

Een sequentiediagram beschrijft het verloop van bepaalde interacties volgens een tijdslijn. In onderstaand diagram (Fig. 2.10) wordt een normaal gebruik van de applicatie, zoals voorgesteld in voorgaande use cases, beschreven.



Figuur 2.8: Use Cases voor de administrator van de procurement-server

Het diagram druk chronologisch uit wat in de vorige paragraaf bij de use cases werd beschreven: een aanbesteding wordt eerst aangemaakt en bekeken. Daarna wordt er een bod geplaats, wat in een tweede ronde nog eens wordt overgedaan, waarna een winnaar wordt gekozen.

Uit het schema kun je opmaken dat JAX-RPC, de technologie voor Java Web services, volledig werkt met synchrone PULL-oproepen. Het is telkens de client die het initiatief neemt om gegevens te gaan opvragen en hij wacht telkens op de server tot er een antwoord komt. Er werd in deze applicatie voor gezorgd dat alle oproepen waarop geen expliciet antwoord verwacht wordt, toch een bevestiging krijgen.

2.4 Architectuur

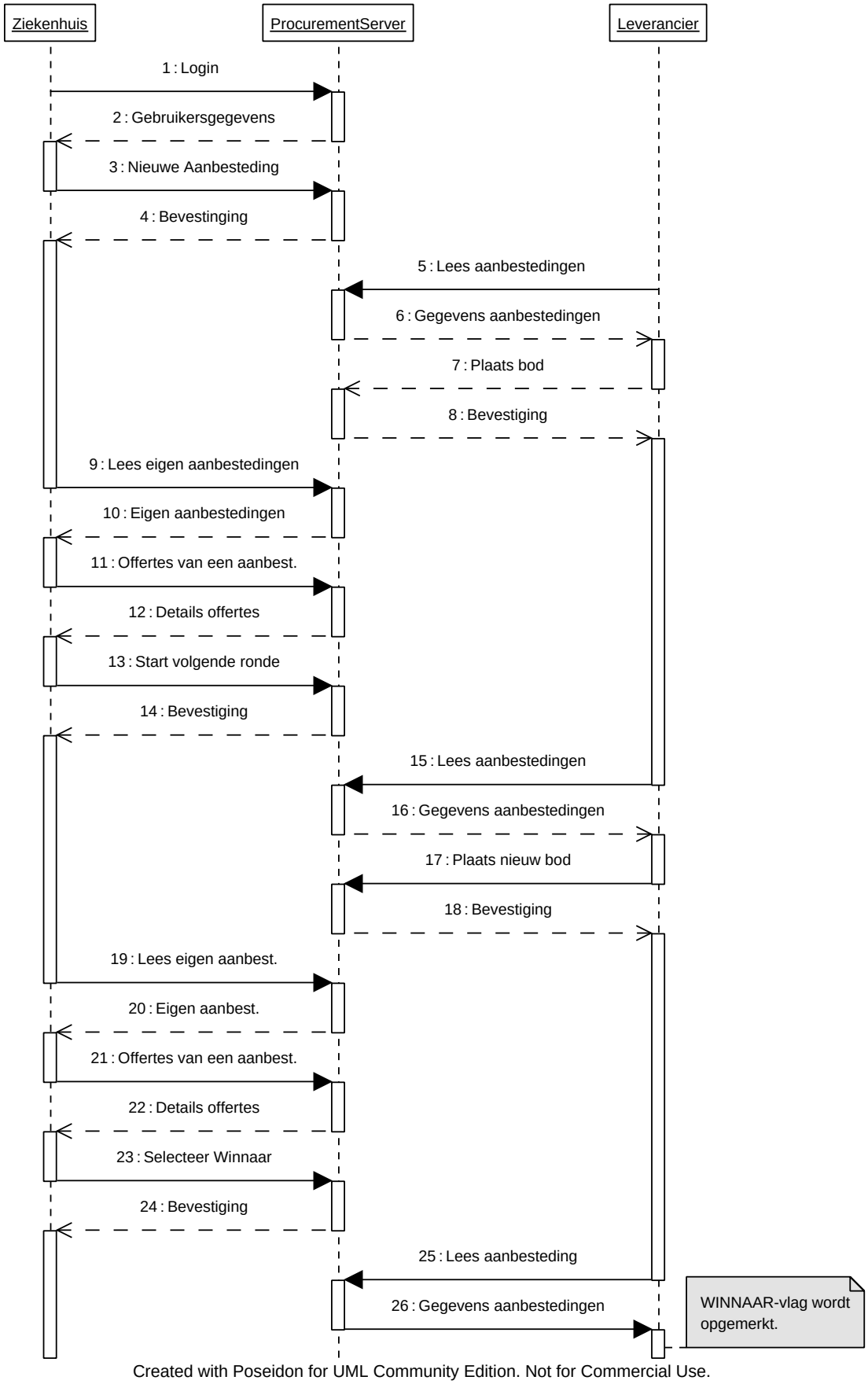
Het groot architectuuroverzicht in figuur 2.11 geeft weer hoe de applicatie globaal werd opgebouwd. Er zijn drie hoofdmodules: de Hospital Client, de Dealer Client en de Procurement Server. De gebruikers van de applicatie, aan de ene kant de aankoopdienst van het ziekenhuis en aan de andere kant de leveranciers, komen enkel in aanraking met de client-GUI's. Deze clients communiceren met telkens twee van de drie lopende Web services: het ziekenhuis connecteert met de HospitalProcService en de leverancier met de DealerProcService. Beiden maken voor het gebruikersbeheer gebruik van de diensten van de UserService.

De Web services gaan op hun beurt de engines (SessionBeans, zie §5.3.2) aanspreken. Deze maken nog enkel een onderscheid tussen methodes voor procurement (de ProcEngine) en methodes voor gebruikersbeheer (de UserEngine). De engines halen hun informatie uit de EntityBeans (zie §5.3.1) die op hun beurt de gegevens in de databank raadplegen.

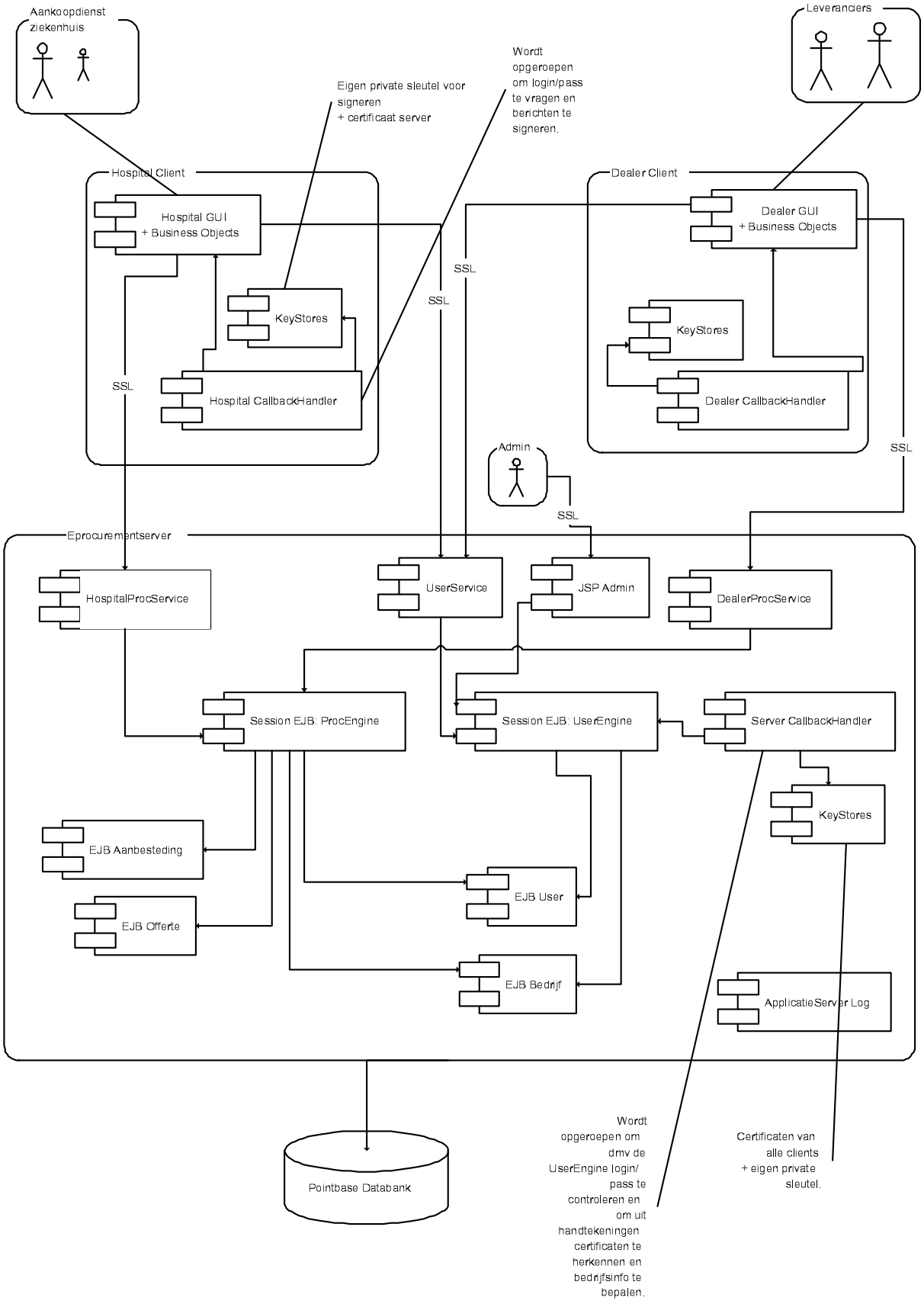
De adminstrator van de procurement-server logt rechtstreeks in op de jsp-admin-pagina, die aangeboden wordt op de server zelf. Deze jsp-pagina heeft geen Web service nodig, maar kan onmiddellijk de UserEngine aanspreken om nieuwe bedrijven en gebruikers aan te maken.

Je kunt zien dat alle verbindingen van en naar de clients over een veilige SSL-verbinding verlopen. Verder beschikt, op de administrator na, elke entiteit over keystores om certificatien in op te slaan en over CallbackHandlers, die over de security van de Web service waken (zie §4). Om te voorkomen dat transacties vanwege de gebruikers op later tijdstip ontkend worden, houdt de applicatie-server zelf log-bestanden bij.

Figuur 2.9: De JSP-pagina voor de administrator



Figuur 2.10: Sequentiediagram voor een normaal procurement-verloop



Figuur 2.11: Overzicht van de architectuur

Hoofdstuk 3

Web services

3.1 Web services algemeen

Web services maken het mogelijk om platform-onafhankelijke client-server-verbindingen op te zetten om gebruik te maken van bepaalde services die een server kan leveren. De communicatie tussen client en server gebeurt via SOAP-berichten (Simple Object Access Protocol). Dit zijn XML-bestanden waarin op eenvoudige manier wordt beschreven welke service er wordt opgeroepen, welke parameters er worden meegegeven en welk type resultaat wordt verwacht. In een SOAP-bericht kunnen verder ook oa. security-headers worden opgenomen. Deze SOAP-berichten worden gewoon over het http(s)-protocol naar de server verstuurd. Dit heeft als voordelen dat er geen bijzondere extra software nodig is (elk bedrijf weet immers hoe een http-bericht te versturen en te ontvangen) en dat bestaande firewall-configuraties niet moeten worden aangepast. In het SOAP-bericht hieronder vraagt de client aan de server om de voornaam van de gebruiker op te zoeken, aan de hand van een primaire sleutel. De parameter die aan de oproep wordt meegegeven is van het SOAP-type enc:long en komt overeen met de primaire sleutel van de betreffende gebruiker in de databank.

```
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:enc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:ns0="urn:userws"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
env:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
```

```
<env:Body>
<ns0:getUserVoornaam>
<Object_1 xsi:type="enc:long">1124056391595</Object_1>
</ns0:getUserVoornaam>
</env:Body>
</env:Envelope>
```

De server stuurt het antwoord terug in een gelijkaardig SOAP-bericht. Het antwoord is hier 'Thomas' en is van het SOAP-type xsd:string.

```
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:enc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:ns0="urn:userws"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
env:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<env:Body>
<ns0:getUserVoornaamResponse>
<result xsi:type="xsd:string">Thomas</result>
</ns0:getUserVoornaamResponse>
</env:Body>
</env:Envelope>
```

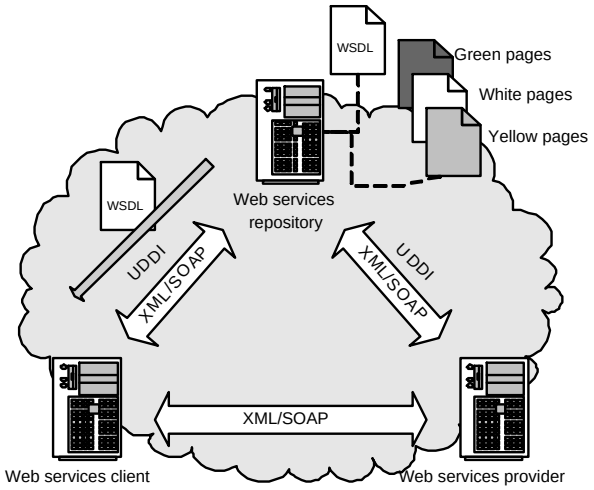
Het zijn net deze SOAP-berichten die Web services platform-onafhankelijk maken. Elke client die zijn berichten opstelt in SOAP-formaat, kan berichten sturen naar een server die deze XML-bestanden kan lezen, onafhankelijk van in welke taal de applicatie is geschreven of welke technologieën er verder worden gebruikt of op welk platform hij draait.

Een server biedt zijn Web services aan aan de buitenwereld door middel van WSDL-files. WSDL (Web Service Description Language) is een gestandaardiseerd XML-formaat voor het beschrijven van Web services. In een WSDL-file staat de naam van de service, de locatie ervan en hoe ermee te communiceren. De WSDL omschrijft dus van de aangeboden methoden de signatuur: naam, types van parameters, type van het antwoord,...

In deze applicatie gaan we ervan uit dat de client weet waar de Web service zich bevindt. Hij geeft de URL van het service endpoint op en kan direct beginnen berichten sturen. Het is echter

ook mogelijk dat een client enkel weet wat voor service hij nodig heeft, stel het oplossen van een groot stelsel vergelijkingen, het boeken van een vliegreis,.... Voor dergelijke welomschreven problemen kan hij beroep doen op een reeds bestaande service. Dergelijke services registreren zichzelf bij een repository of broker. Ze doen dit door, met het UDDI-protocol (Universal Description, Discovery and Integration), hun WSDL-file daar te publiceren. UDDI-berichten worden eveneens verstuurd, verpakt in SOAP-berichten, en vormen dus nog een extra laag bovenop SOAP. Naast de zuivere technische informatie, die gepubliceerd wordt in de 'Green Pages', omvat UDDI ook 'White Pages', die naam en adres van de service-leverancier bevatten, en 'Yellow Pages', waar de services per categorie en omschrijving worden gerangschikt. De client kan dan, ook via UDDI-berichten, de service voor het probleem in kwestie, gaan opzoeken bij de repository, die dan de WSDL-file aflevert.

Samenvattend bestaat een Web service dus uit een client, een service provider en een repository, die alledrie met elkaar communiceren via SOAP-berichten (Fig. 3.1).



Figuur 3.1: Web services

3.2 SSL vs. WSS

In deze thesis werd beveiliging op twee verschillende lagen van het OSI-model toegepast. Enerzijds gaat alle communicatie naar de Web services over een SSL-verbinding, waarbij SSL (Secure Sockets Layer) kan gezien worden als een extra laag tussen de transport- en de applicatielaag. De server (en in dit geval ook de client) wordt hierdoor geauthenticeerd en alle communicatie verloopt volledig geëncrypteerd en beveiligd.

Die SSL-beveiliging duurt echter maar tot wanneer het bericht ontvangen wordt door de applicatieserver. Het bericht wordt bij het binnenkomen in de server terug gedecodeerd en het SOAP-bericht, terug een plain-text XML-bestand, wordt verder verwerkt door de Web service. In een aantal gevallen, zoals ook hier, kan het nodig zijn dat de beveiliging ook ná het ontvangen verzekerd wordt. O.a. hiervoor wordt WSS (Web Service Security) gebruikt. WSS werkt op berichtniveau en gaat dus beveiliging inbouwen in de SOAP-berichten zelf. WSS staat toe om heel gericht delen van de SOAP-berichten te gaan coderen en ondertekenen, tot op methode-niveau. Verder is het mogelijk om in de SOAP-berichten extra beveiligingstokens te gaan invoegen, zoals gebruikersnamen en wachtwoorden (UsernameTokens), Timestamps,.... Doordat de beveiliging verwerkt zit in de XML van het bericht, blijft deze ook bestaan na het binnenkomen in de server. De Web service kan dan zelf verder beslissen wat met de berichten en de beveiliging gebeurt. In de procurement-server worden de berichten bvb. met ondertekening inbegrepen opgeslagen in een log-bestand. Zo blijft ook in het log-bestand de integriteit van de berichten gewaarborgd. Ook de UsernameTokens blijven aan het bericht gekoppeld. Hieronder zie je de XML-code van dezelfde SOAP-aanvraag als in § 3.2. Hier werd echter een UsernameToken toegevoegd (de Username is 'T.') en zowel token als de inhoud van het bericht werden ondertekend. Hier en daar werd de code ingekort en vervangen door '...'.

```
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
...
```

Hier begint de header van het SOAP-bericht. De security-header krijgt de vlag 'mustUnderstand' mee, wat wil zeggen dat de server niet mag antwoorden als hij zelf niet op de hoogte is van security-opties.

```
<env:Header>
<wsse:Security ... env:mustUnderstand="1">
<wsse:BinarySecurityToken
...
```

De handtekening begint met op te geven dat hij de RSA-SHA1-methode gebruikt (zie hoofdstuk Security § 4). Vervolgens worden referenties opgegeven naar alle delen van het SOAP-bericht die ondertekend worden. Op die delen wordt telkens een hash-functie uitgevoerd, die later kan ondertekend worden.

```
<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
<ds:SignedInfo>
...
<ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
<ds:Reference URI="#XWSSGID-112508902292120024482 34">
<ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
<ds:DigestValue>EPW55vkB6PZ7Cv4qUU11sFl9qJs=</ds:DigestValue>
</ds:Reference>
<ds:Reference URI="#XWSSGID-1125089022921-1195360992">
<ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
<ds:DigestValue>PtAfsg304yKyT6yNIkpZS7i85iQ=</ds:DigestValue>
</ds:Reference>
<ds:Reference URI="user-token">
<ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
<ds:DigestValue>CfMhK4sz0O6eEygeNnyfMVE/Wlk=</ds:DigestValue>
</ds:Reference>
</ds:SignedInfo>
<ds:SignatureValue>KCf6Sfh9UxsDNTuoHBV1rYooXZPkwpa
U5tWfegU5r7+7HmMBQ8fYIfxbIM7hRxjh6JrlNUo5OxIN
LMJUQB6AemPZMCUQFBX7ancUSAHPH2BBcFAS/ImqQ1VhF
4DSqjyStBhcPQNRu36rRU2L3Md1ttPq
3h/m5n05NLuREP3cXys=</ds:SignatureValue>
...
</ds:Signature>
```

Bij het ondertekenen wordt automatisch een Timestamp toegevoegd. Daaronder staat het UsernameToken met hier als Username de waarde 'T.'. Let erop dat de id's van zowel Timestamp als UsernameToken overeenkomen met de te ondertekenen references uit de handtekening.

```
<wsu:Timestamp ... wsu:Id="XWSSGID-112508902292120024482 34">
<wsu:Created>2005-08-26T20:43:42Z</wsu:Created>
<wsu:Expires>2005-08-26T20:48:42Z</wsu:Expires>
</wsu:Timestamp>
<wsse:UsernameToken ... wsu:Id="user-token">
```

```
<wsse:Username>T.</wsse:Username>
```

Het wachtwoord dat bij de UsernameToken hoort wordt onleesbaar weergegeven. Verder wordt hieraan nog een nonce toegevoegd, waarna de security-header wordt afgesloten.

```
<wsse:Password Type="...">****</wsse:Password>
<wsse:Nonce EncodingType="...">55Fqiec/LcE3MH0mExXx9Lml</wsse:Nonce>
<wsu:Created>2005-08-26T20:43:42Z</wsu:Created>
</wsse:UsernameToken>
</wsse:Security>
</env:Header>
```

Op het einde komt de eigenlijke body van het bericht (ook deze id werd opgenomen bij de references). De eigenlijke vraag: getUserVoornaam, krijg als parameter een long mee, de primaire sleutel van de gevraagde gebruiker in de databank.

```
<env:Body ...  wsu:Id="XWSSGID-1125089022921-1195360992">
<ns0:getUserVoornaam>
<Object_1 xsi:type="enc:long">1124056391595</Object_1>
</ns0:getUserVoornaam>
</env:Body>
</env:Envelope>
```

Dit volledige bericht komt in het log-bestand terecht. Het is dus duidelijk dat zo ook alle beveiligingsopties bewaard blijven. Het antwoord van de server zal hetzelfde zijn als hoger beschreven. Het wordt immers niet van de server gevraagd om de berichten te ondertekenen of om een wachtwoord op te geven.

3.3 JAX-RPC

JAX-RPC (Java API for XML-based RPC) is de Web service-implementatie die in J2EE wordt gebruikt om Web services mee op te zetten. RPC staat voor Remote Procedure Call, wat zoveel wil zeggen als het oproepen van procedures op een andere host. Die oproepen worden verstuurd als SOAP-bericht.

Voor de programmeur zijn Java Web services eerder transparant. Het is nooit nodig om de XML voor de SOAP-berichten zelf te gaan opstellen. De client maakt immers gebruik van

een proxy (stub), een klasse die de interface van de Web service beschrijft en die zelf voor de vertalingen naar SOAP-berichten en voor de verbinding met de server zorgt. De client initialiseert dus enkel het stub-object en roept dan hierop methoden op, alsof het om een lokale klasse gaat. Deze stub wordt bij compilatie gegenereerd (door de tool wscompile), op basis van de WSDL-file. Er bestaan ook meer geavanceerde methodes, waarbij de client tijdens het uitvoeren de WSDL-file gaat ophalen om at runtime een stub te genereren. Dit is echter eerder nodig wanneer het mogelijk is dat de interface van de Web service ondertussen veranderd is of bij het compileren van de client nog niet helemaal is vastgelegd. Hier volstaat het om de stubs statisch aan te maken.

Aan de serverkant beschikt de server over de interface die de service aanbiedt (diezelfde interface die ook door de WSDL wordt omschreven) en de klassen en Enterprise Java Beans (zie §5.3) die deze interface implementeren. De communicatie met de client wordt verzorgd door de ties, structuren die per methode instaan voor het ontvangen en vertalen van SOAP-berichten. Bij het deployen van de service worden ook deze structuren zonder tussenkomst van de programmeur gegenereerd.

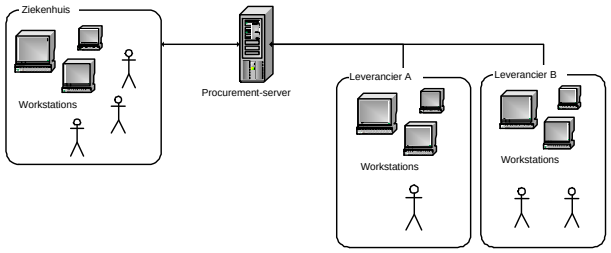
Hoofdstuk 4

Security

4.1 Threats and solutions

4.1.1 Opstelling

Zoals ondertussen duidelijk is, hebben we te maken met drie componenten (de server en twee soorten clients, nl. het ziekenhuis en de leveranciers), waartussen het dataverkeer beveiligd moet worden (Fig. 4.1). Beide entiteiten moeten er telkens zeker van zijn dat ze wel degelijk met elkaar communiceren (verificatie), dat de boodschappen niet gewijzigd werden (integriteit) en dat eventuele concurrentie de boodschappen niet kan lezen (vertrouwelijkheid). Verder moet er een bewijs bestaan van bepaalde transacties (niet-bestrijdbaarheid) en moeten de verzenders er zeker van zijn dat hun berichten wel degelijk aankomen.



Figuur 4.1: Opstelling

4.1.2 Verificatie

Ziekenhuis - server

Een aanbesteding mag enkel worden aangemaakt door het ziekenhuis. Het ziekenhuis moet er tevens zeker van zijn dat het met de server communiceert, zodat een bedrijf dat zich zou voordoen als procurement-server niet voortijdig op de hoogte kan zijn van een nieuwe aanbesteding (en hieraan bvb. zijn prijzen aanpassen?). Ook de administratortaken die het ziekenhuis kan uitvoeren (regels van de aanbesteding instellen), mogen uiteraard enkel en alleen toegankelijk zijn voor dat ziekenhuis. Zowel de client in het ziekenhuis als procurement-server moeten dus geverificeerd (geauthenticeerd) worden.

Leverancier - server

Het ligt voor de hand dat een leverancier bij het registreren bij de server zijn identiteit moet bewijzen en dit dan telkens moet doen bij elke transactie. Het is immers niet de bedoeling dat er in naam van een ander bedrijf geboden wordt. Ook de leverancier moet zeker weten dat hij met de procurement-server zelf onderhandelt en niet met bvb. een concurrent die zich als server voordoet.

Oplossing

Een grote stap in de goede richting is het gebruik van SSL. Zowel de server als alle clients krijgen een certificaat waarmee ze zich kunnen authenticeren. Dit heeft als neveneffect dat een werknemer die toegang heeft tot de service dit niet van thuis uit kan doen, tenzij hij kan beschikken over een kopie van het certificaat (wat geen goede security policy is). Als het certificaat echter van een Smart Card wordt gelezen, kan eenvoudig worden bepaald wie al dan niet een kaart en dus toegang krijgt (en dit van om het even welke plaats). De certificaten kunnen getekend worden door een betrouwbare CA (Certificate Authority). Aangezien het bedrijf dat instaat voor de procurement-server echter ook door alle partijen wordt vertrouwd, kan dit bedrijf onmiddellijk optreden als CA en zo een extra kost uitsparen.

Eens de certificaten zijn uitgewisseld en de SSL-connectie is opgezet, kunnen alle berichten nog steeds afzonderlijk worden ondertekend door het gebruik van XWS-Security (XML and Web Services Security). Dit is een implementatie van WSS (Web Service Security) die end-to-end-beveiling voorziet in de SOAP-berichten zelf. Het ondertekenen van de SOAP-berichten wordt gedaan door XML Digital Signature (DSig), dat gebruik maakt van Apaches XML-DSig-

implementatie. Om ook exact te weten met welke gebruiker de server te maken heeft, kan besloten worden een UsernameToken mee te sturen dat loginnaam en wachtwoord van een bepaalde gebruiker bevat (zie ook §4.1.8).

4.1.3 Vertrouwelijkheid

Ziekenhuis - server

Het is niet strikt noodzakelijk dat de eventuele administratortaken geëncrypteerd worden. Ook het plaatsen van een aanbesteding is niet geheim, aangezien deze onmiddellijk wordt bekend gemaakt aan alle mededingers. Wat wel zeker onleesbaar moet worden gemaakt, zijn de geplaatste biedingen die het ziekenhuis opvraagt om de 'winnaar' te selecteren, evenals de verstuurde berichten i.v.m. wie al dan niet heeft gewonnen. Concurrenten mogen immers niet weten wat hun tegenstanders hebben geboden of wie uiteindelijk de laagste prijs had.

Leverancier - server

Berichten tussen server en leverancier moeten steeds geëncrypteerd zijn. Niemand hoeft te weten hoeveel een bepaald bedrijf geboden heeft (behalve dan het ziekenhuis zelf) en niemand hoeft op de hoogte te zijn welk bedrijf nu uiteindelijk de aanbesteding heeft gekregen. De informatie over de aanbesteding die de server naar de leverancier stuurt is niet noodzakelijk geheim. Deze informatie krijgt tenslotte iedereen.

Oplossing

De SSL-verbinding zal alle verkeer tussen clients en server encrypteren, gebruik makend van een symmetrische sleutel, vastgelegd bij het opzetten van de SSL-tunnel. D.m.v. XML Encryption (XML-Enc) kan XWS de SOAP-berichten zelf nogmaals encrypteren. Op die manier zijn de berichten nog steeds onleesbaar, ook nadat ze uit de 'tunnel' komen. XML-Enc is gebaseerd op Apaches XML-Enc-implementatie.

4.1.4 Berichtintegriteit

Ziekenhuis - server

Uiteraard mogen berichten tussen ziekenhuis en server onderweg niet aanpasbaar zijn. 'Spelregels' mogen niet onderschept en gewijzigd worden en, veel belangrijker: de informatie waarop

het ziekenhuis zich baseert om een 'winnaar' te kiezen, moet wel degelijk die van op de server zijn.

Leverancier - server

Een leverancier moet er steeds op kunnen vertrouwen dat biedingen die hij naar de server stuurt, daar ongewijzigd toekomen. Het zou anders te gemakkelijk zijn om de prijs van alle concurrenten te verhogen en zo zelf steeds het laagste bod te hebben. Ook status-berichten (zoals 'U bent geselecteerd' of 'Er komt een tweede ronde, biedt u nog steeds mee?') moeten gegarandeerd ongewijzigd bezorgd worden.

Oplossing

Op transportniveau zorgt de SSL-verbinding er reeds voor dat berichten ongewijzigd hun bestemming bereiken. Ook hier biedt XWS met DSig een extra garantie dat berichten integer zijn, ook nadat ze op hun bestemming zijn afgeleverd.

4.1.5 Niet-bestrijdbaarheid

Ziekenhuis - server

Gesteld dat de procurement-server in handen is van een bedrijf dat door alle partijen vertrouwd wordt, is het aangewezen dat de server een log bijhoudt van de geplaatste (of gewijzigde) aanbestedingen, op zo'n manier dat het ziekenhuis niet kan ontkennen dat ze ooit een bepaalde aanbesteding heeft geplaatst. Op gelijkaardige wijze kan worden geregistreerd welk bedrijf als 'winnaar' wordt uitgeroepen, zodat naderhand niet op die beslissing kan worden teruggekomen.

Leverancier - server

Een bieding mag niet terug ingetrokken worden. Om te voorkomen dat een leverancier, na het winnen van de aanbesteding gaat ontkennen dat hij ooit heeft meegeboden, moet er een logbestand worden bijgehouden op de server, dat alle biedingen registreert. Op analoge wijze kan ook voorkomen worden dat een leverancier beweert een ander bedrag geboden te hebben. Deze niet-bestrijdbaarheid gaat natuurlijk samen met de berichtintegriteit. Het heeft immers geen zin om een bericht te loggen samen met de identiteit van de verzender, als het niet zeker is dat de inhoud ongewijzigd is gebleven.

Oplossing

De met XML-DSig ondertekende berichten moeten, samen met een Username Token, opgeslagen worden in een databank. Zo kan onweerlegbaar worden aangetoond vanaf welke client de berichten werden verstuurd en door welke werknemer (of toch op z'n minst door iemand die beschikt over het juiste certificaat én over de login-gegevens van die bepaalde werknemer). Voorzien van een Timestamp (eveneens aangeboden door XWS), kan bovendien exact worden bepaald wanneer het bericht werd verzonden.

4.1.6 Bevestiging

Verdwijnen van een bericht

In een aantal gevallen zou de verzendende partij er zeker van moeten zijn dat zijn boodschap ook daadwerkelijk toekomt. Het zou immers onaanvaardbaar zijn als een concurrent systematisch de berichten naar andere leveranciers aangaande een nieuwe aanbesteding zou onderscheppen en zo de enige bieder worden. Dit is ook het geval voor alle biedingen én voor de status-berichten. Er kunnen ook niet-frauduleuze oorzaken zijn waarom een bericht niet zou toekomen (uitvallen van de verbinding?). Daarom zou een verzender steeds een bevestiging moeten krijgen van de informatie die hij heeft ingediend.

Oplossing

Dit is waarschijnlijk een probleem dat impliciet door de reeds gebruikte technologieën zal worden opgelost. Als een Remote Procedure Call immers mislukt (het bericht wordt onderschept, de verbinding valt uit, enz...), dan wordt normaalgezien een (Remote)Exception gegenereerd. Bij een kwaadwillige aanval, waarbij een concurrent tracht de verzender te laten geloven dat de RPC toch gelukt is (terwijl het bericht in werkelijkheid vernietigd is), zullen er foutmeldingen optreden door een verkeerde signature of verkeerde logingegevens. Het zonder meer, ongemerkt verdwijnen van berichten zou dus in principe niet kunnen voorkomen. Om de gebruiker gerust te stellen kunnen eventueel toch berichten worden getoond wanneer een bepaalde actie is geslaagd.

4.1.7 Replay-attacks

Ziekenhuis - server

Ook al is het plaatsen van een aanbesteding niet zo geheim, het moet vermeden worden dat iemand de berichten voor het plaatsen van een aanbesteding heeft kunnen onderscheppen en die informatie gebruikt om later zelf opnieuw die aanbesteding te plaatsen.

Leverancier - server

Replay-aanvallen zijn waarschijnlijk gevaarlijker bij het bieden. Als een concurrent een (gecodeerde) bieding kan onderscheppen en die opnieuw kan gebruiken op een later tijdstip, dan kan hij zonder medeweten van de concurrentie in hun naam biedingen plaatsen.

Oplossing

Door het insluiten van een Timestamp in de SOAP-berichten kan gecontroleerd worden of het bericht niet te lang geleden werd opgesteld. Voor een dergelijke Timestamp kan een timeout worden ingesteld, aan de hand waarvan beslist wordt of het bericht al dan niet verlopen is. Bovendien kan ook geopteerd worden voor het gebruik van een nonce. Hierdoor wordt een foutmelding gegenereerd wanneer die bepaalde waarde al eens werd gebruikt. Met deze twee methoden kan men redelijk zeker zijn dat de boodschap 'live' werd gestuurd en niet werd opgeslagen om op later tijdstip misbruikt te worden. Aangezien in een bieding moet worden gespecifiëerd op welke aanbesteding wordt geboden, zou een replay-aanval van die kant trouwens enkel kunnen gebruikt worden binnen één zelfde ronde, wat op zich al weinig nut heeft. Het wijzigen van het aanbestedings-id wordt dan weer onmogelijk gemaakt door de encryptie en ondertekening.

4.1.8 Gebruikersbeheer

Toegangsrechten

Zowel in het ziekenhuis als bij de leveranciers mogen slechts bevoegde personen toegang krijgen tot de informatie en tot de webservice. Enkel bvb. de verantwoordelijken van de aankoopdienst van het ziekenhuis mogen aanbestedingen kunnen plaatsen en enkel bevoegden in het leveranciersbedrijf mogen hierop kunnen reageren. Ook de toegang tot de databank in het algemeen moet binnen het betreffende bedrijf beperkt worden tot de bevoegde werknemers.

Oplossing

Enkel bevoegde gebruikers krijgen een Smart Card toegewezen met het nodige certificaat op. Verder moet voor elk bedrijf en voor het ziekenhuis een lijst van bevoegde gebruikers met hun wachtwoorden worden bijgehouden. Om deze wachtwoorden ook voor de databank-administrator (of voor eventuele anderen met ongeoorloofde toegang tot de databank) onleesbaar te maken, worden niet de wachtwoorden zelf, maar wel hun hashfuncties opgeslagen. Dit valt te verwezenlijken met Java's MessageDigest-klasse, eventueel uitgebreid met BASE64-codering en toevoegen van wat 'zout'.

Login-informatie kan worden beheerd door gebruik te maken van JAAS (Java Authentication and Authorization Service), een set API's voor authenticatie en toegangscontrole. De verkregen login-gegevens worden met XWS verstuurd in een Username Token.

De databank zelf, los van de webservice, moet uiteraard ook beveiligd zijn, zodat enkel de databank-administrator er rechtstreeks op kan inloggen (en dit enkel lokaal).

4.1.9 Administratie en registratie

Eventuele administratie-taken kunnen worden uitgevoerd op de procurement-server zelf. Hiervoor kan een applicatie-client of jsp-pagina worden gebruikt die verbinding maakt met de interface van de UserEngine. Op die manier kunnen van buitenaf geen wijzigingen worden aangebracht aan de server. De toegang tot die administratortaken kunnen nog worden beveiligd met een admin-login en een wachtwoord.

Registreren van een bedrijf of van het ziekenhuis bij de server zal niet uitsluitend online lukken. Een bedrijf dat zich wilt registreren zal moeten persoonlijk contact opnemen met het bedrijf dat instaat voor de server. Het is dan aan deze laatste om de identiteit en betrouwbaarheid van de leverancier na te trekken alvorens een certificaat uit te reiken en het bedrijf op te nemen in de leverancierslijst. Een alternatief is om enkel het certificaat uit te reiken waarmee het bedrijf dan later zichzelf online kan registreren. Het aanmaken van een lijst gebruikers kan in elk geval door het bedrijf zelf online gebeuren.

4.1.10 Andere gaten in de beveiliging

Zoals de meeste applicaties zal de eProcurement-service nog vele, kleine, specifieke beveiligingsproblemen vertonen die meestal pas bij gebruik worden opgemerkt en dan in volgende versies worden 'gepatched'. Een voorbeeld hiervan zijn bvb. injections bij het gebruik van een SQL-

databank. Een SQL-statement dat een gebruikersnaam en wachtwoord controleert kan geschreven worden als

```
SELECT * FROM logins WHERE username="admin" AND password="pass"
```

Door nu als wachtwoord `foo" OR "1"="1` op te geven, is de 'beveiliging' al omzeild. Alle ingevoerde data moet dus grondig geverificeerd worden.

Dergelijke fouten zijn echter zeer implementatie-specifiek en dus moeilijk op voorhand in te calculeren.

4.1.11 Conclusie

Samenvattend kunnen we besluiten dat het voor de beveiliging van de procurementapplicatie volstaat om volgende security te implementeren:

- SSL-verbindingen, zowel tussen ziekenhuis en server als tussen leverancier en server met wederzijdse authenticatie. Dit voorziet in de nodige encryptie, verificatie én integriteit. Elke instantie beschikt over één certificaat op een of meerdere Smart Cards. De mogelijkheid tot het gebruik van deze Smart Cards moet in de implementatie worden opengelaten, maar voor deze scriptie zal gebruik gemaakt worden van een meer rudimentaire oplossing.

- Alle berichten tussen de server en de verschillende clients worden ondertekend d.m.v. XML-DSig en vergezeld van een UsernameToken en een Timestamp. Hierdoor weten we exact welke gebruiker de verrichtingen uitvoerde en op welk ogenblik. De Timestamp is bovendien een remedie tegen replay-aanvallen. Integriteit tijdens het verzenden wordt reeds door SSL gewaarborgd. De berichten (inclusief Timestamp en UsernameToken) moeten echter ondertekend blijven om gelogged te worden en zo niet-bestrijdbaarheid te garanderen. Voor het ondertekenen kan hetzelfde sleutelpaar met bijhorend certificaat worden gebruikt als voor SSL. Elke instantie (ziekenhuis, leverancier, ...) beschikt dus over één certificaat/sleutelpaar (eventueel meerdere kopieën op Smart Cards), terwijl de afzonderlijke gebruikers herkend worden door hun login/pass, ingesloten in het UsernameToken.

- Alle gebruikers van de verschillende bedrijven hebben een eigen login en wachtwoord, opgeslagen in de gebruikersdatabank op de server. De wachtwoorden worden, 'gezouten' en gehashed d.m.v. MD5 en BASE64, opgeslagen.

Deze oplossing zal eventueel hier en daar een boodschap encrypteren die eigenlijk niet strikt geheim is, of een bericht ondertekenen dat uiteindelijk niet wordt gelogged. Het zal echter eenvoudiger zijn om deze algemenere oplossing te implementeren dan om op methode-niveau aan de slag te gaan. De performantie zal niet wezenlijk lijden onder een enkele overtollige encryptie, aangezien het hier niet gaat om een constante informatiestroom, maar om een sporadisch (dagelijks, wekelijks?) vraag en antwoord tussen één ziekenhuis en een beperkt aantal leveranciers. Bovendien wordt het zwaarste werk, het encrypteren van elk bericht, verzet door de SSL-verbinding, die een stuk performanter is dan de XWS-encryptie.

4.2 Technologieën

Zowel voor het gebruik van SSL als voor het ondertekenen van boodschappen met XWS zijn er certificaten nodig. De werking hiervan wordt in de volgende paragrafen verklaard.

Digitale certificaten steunen op het gebruik van publieke-sleuteltechnologie. Elke gebruiker X beschikt over twee sleutels: een private (D_X) en een publieke sleutel (E_X). De publieke sleutel kan door iedereen opgevraagd en gebruikt worden, de private sleutel blijft vanzelfsprekend geheim. Een boodschap P die nu bvb. verstuurd wordt van A naar B en die geheim moet blijven, wordt door A geëncrypteerd met de publieke sleutel van B: $E_B(P)$. Een dergelijk bericht kan nu enkel door middel van de bijhorende private sleutel, die alleen B in zijn bezit heeft, worden ontcijferd: $D_B(E_B(P)) = P$

Het omgekeerde werkt ook: een bericht dat door de private sleutel van A werd versleuteld ($D_A(P)$) kan door iedereen gelezen worden, aangezien in theorie iedereen kan beschikken over A's publieke sleutel: $E_A(D_A(P)) = P$. Het bericht P werd nu echter onweerlegbaar geschreven en vercijferd door A, aangezien enkel A beschikt over private sleutel D_A. Dit is het digitaal ondertekenen van een bericht. Het RSA-algoritme (in 1978 uitgevonden door Rivest, Shamir en Adleman) is een veelgebruikte techniek voor publieke-sleutelcryptografie en is gebaseerd op het feit dat het moeilijk is om grote getallen in factoren te ontbinden.

Het RSA-algoritme heeft als nadeel dat het redelijk veel rekentijd vraagt om een bericht te vercijferen. Als het niet noodzakelijk is dat een bericht onleesbaar is, maar enkel moet ondertekend worden, hoeft niet het volledige bericht gecodeerd te worden met de private sleutel van A, maar slechts de 'vingerafdruk' van dat bericht. Een dergelijke vingerafdruk wordt bekomen door een hash-functie uit te voeren. Voor een dergelijke hash-waarde $H(P)$, bekomen uit bericht P, moet het onmogelijk zijn om uit $H(P)$ terug P te construeren, én in praktijk moet het

onmogelijk zijn om twee berichten P en P' te vinden waarvoor geldt dat $H(P) = H(P')$.

Om nu een bericht P te ondertekenen zoeken we de hash-waarde $H(P)$ en encrypteren die met de private sleutel van A: $D_A(H(P))$. Deze handtekening wordt nu samen met de plaintext-boodschap verstuurd. Om te bewijzen dat het bericht met zekerheid door A werd geschreven volstaat het nu om de handtekening te decoderen ($E_A(D_A(H(P))) = H(P)$) en uit P opnieuw de hash-waarde $H(P)$ te berekenen. Als beide bekomen waarden hetzelfde zijn is er geen twijfel over de herkomst van het bericht.

Twee veelgebruikte hashfuncties zijn MD5 en SHA. MD5 levert functiewaarden van 128 bits af, waarbij iedere output-bit afhangt van iedere input-bit. SHA geeft resultaten van 160 bits, waardoor het iets langer duurt om een hash-functie te berekenen, maar het resultaat wel veiliger is. In deze applicatie werd MD5 gebruikt om de wachtwoorden van de gebruikers onleesbaar op te slaan in de databank. SHA wordt door XWS-Security gebruikt bij het hashen van de SOAP-berichten, waarna ze d.m.v. RSA kunnen ondertekend worden.

Het zwakke punt bij publieke-sleuteltechnologie is het verdelen van de publieke sleutels. Niets koppelt immers de identiteit van een gebruiker aan zijn publieke sleutel. Dit kan worden opgelost d.m.v. certificaten. Een certificaat bevat, naast de publieke sleutel van een gebruiker, ook diens identiteit. Een certificaat wordt ondertekend door een CA (Certificate Authority), een door iedereen vertrouwd bedrijf dat identiteiten van gebruikers nagaat en certificaten uitreikt. In deze applicatie werden de certificaten getekend door de applicatieserver zelf, aangezien die in principe ook een door alle partijen vertrouwde partner is.

De communicatie in deze applicatie verloopt steeds over SSL volgens een tweerichtingsverbinding. De client biedt dus zijn certificaat aan aan de server, die kijkt of dit certificaat wel in zijn truststore, zijn verzameling van vertrouwde certificaten, voorkomt. Ook de client ontvangt een certificaat van de server. Als beide kanten elkaar vertrouwen, kiest de client een willekeurige symmetrische sleutel en codeert die met de openbare sleutel van de server. De server kan, na ontvangst, deze sleutel weer decoderen met zijn private sleutel, zodat beiden nu over dezelfde symmetrische sleutel beschikken om alle volgende berichten mee te coderen. SSL vormt dus als het ware een extra laag tussen de transport- en de applicatielaag die elk verzonden bericht beveiligt. SSL garandeert nu dus authenticatie, vertrouwelijkheid en integriteit. De exacte procedure voor het opzetten van een SSL-verbinding kan gevonden worden bij Security Developer Central van Netscape [6].

Hoofdstuk 5

Implementatie

5.1 Installatie

5.1.1 Platform

Om de procurement-server en de clients aan de praat te krijgen moet je uiteraard beschikken over de meest recente Java Development Kit en een applicatieserver. Als het nog nodig is om de clients opnieuw te builden, zullen ook de tools, vervat in het laatste nieuwe Java Web Services Developer Pack, noodzakelijk zijn.

Op de site van Sun [7] kunnen alle gebruikte pakketten worden gedownloaded. Op moment van schrijven zijn de meest recente pakketten:

- Java 2 Platform, Standard Edition (J2SE 5.0)
- Java 2 Platform, Enterprise Edition (J2EE 1.4)
- Java Web Services Developer Pack (Java WSDP 1.6)

Toen ik aan deze thesis begon was JWSDP 1.6 nog niet beschikbaar. Het bleek echter noodzakelijk om te upgraden, aangezien de wscompile tool uit JWSDP 1.5, waarmee onder andere de stub-klassen gegenereerd worden, nog geen opties voor security bevatte. Op moment van schrijven rept de handleiding van wscompile overigens nog met geen woord over een dergelijke optie, alhoewel deze nu wel beschikbaar is. Verder is het eigenlijk nergens nodig om jars uit de JWSDP te gebruiken, aangezien het gaat om exact dezelfde versies als reeds zijn opgenomen in de applicatieserver. Het volstaat dus om enkel laatstgenoemde libraries te importeren in het classpath. Het per ongeluk dubbel insluiten van jar-files zorgt trouwens voor een 'sealing error',

wat betekent dat klassen van hetzelfde pakket uit verschillende jars worden geladen, terwijl dit niet wordt getollereerd door de security policy. De voorwaarden op dit gebied worden ingesteld in de manifest-file die steeds in een jar zit ingesloten.

In de algemene bestanden `targets.xml` en `build.properties` werd een algemeen classpath aangemaakt met de nodige libraries, zodat in verdere buildfiles telkens gebruik kan worden gemaakt van de verwijzing `app.classpath`. Voor het compileren of runnen kan dit classpath dan worden uitgebreid met de nodige klassen.

Het installeren van de nodige pakketten (in de volgorde J2SE, J2EE en JWSDP) spreekt redelijk voor zich. Toch moet er zeker op gelet worden dat verschillende omgevingsvariabelen goed worden ingesteld als dit nog niet is gebeurd:

- PATH: `<JWSDP_HOME>\jaxrpc\bin`; `<JWSDP_HOME>\apache_ant\bin`; `<JWSDP_HOME>\jwsdp-shared\bin`; `<J2EE_HOME>\bin`; `<JAVA_HOME\bin`
- JAVA_HOME: installatiemap van de sdk
- J2EE_HOME: installatiemap van de applicatieserver
- SJAS_HOME: idem J2EE_HOME
- JWSDP_HOME: installatiemap van het Web service pack
- ANT_HOME: `<J2EE\_HOME>\bin`

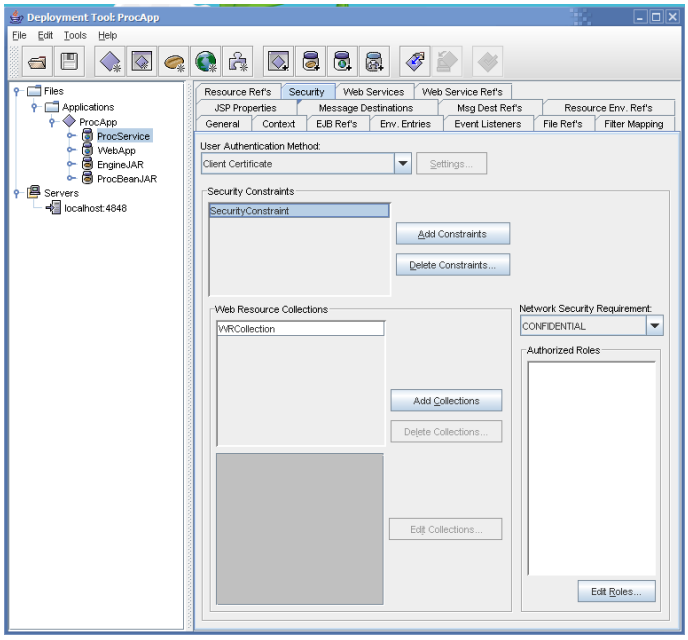
Hier kan het dus wel belangrijk zijn dat de meest recente tools (zoals wscompile) uit de Web service pack kunnen worden gebruikt. Uitgebreide nota's over het installeren van alle nodige onderdelen vind je in de J2EE- en de JWSDP-tutorials [8] [10]. Wat in deze handleidingen niet duidelijk wordt vermeld, maar wat toch vaak problemen geeft, is spaties in het path. Installeer dus eerder alle onderdelen in een nieuwe directory 'Sun' (overigens de default-instelling bij installatie), dan in een submap van het voor Windows meer gebruikelijke 'Program Files'.

5.1.2 Builden van de procurement-applicatie

Allereerst moet de databank worden geïnitialiseerd (de nodige tabellen moeten worden aangemaakt) en moet een resource worden aangemaakt zodat de applicatieserver met de databank kan communiceren (§5.2). Uiteraard moeten de applicatieserver en het DBMS gestart zijn. Dit kan via een batch-file of gewoon vanuit 'menu Start'.

De service is volledig verpakt als web-applicatie in een ear-file. Dit is een bestand in het gebruikelijke jar-formaat, maar met de extensie `.ear`. Deze ear-file bevat op zijn beurt de jar- en war-files (java archief en web archief) die de verschillende onderdelen van de service bevatten. Het volstaat nu dus in principe dit bestand te laden in deploytool en op 'deploy' te klikken om alles in de applicatieserver te laden en te activeren.

Als echter de clients nog moeten worden gebuilded, is het nodig om eerst even de SSL-verplichting op de Web services uit te zetten. Het lukt wscompile niet om over een beveiligde verbinding de WSDL-file te lezen, wat nodig is om de client-stubs te genereren (zie § 5.4). Deze SSL-verplichting kan worden uitgeschakeld door in de ProcService op het tabblad Security de Network Security Requirement van CONFIDENTIAL op NONE te zetten (Fig. 5.1). Op die manier is een applicatie niet verplicht om SSL te gebruiken als hij dit niet kan.



Figuur 5.1: Security tab voor Web services

Wanneer de applicatie, met verminderde beveiliging, is gedeployed, kunnen de clients worden gebuilded met behulp van bijhorende build-files. Met de batch-files `build`, `clean` en `compile` roep je ASant-targets aan die respectievelijk de volledige client en stubs compileren, alle gegenereerde bestanden verwijderen en enkel de gewone java-code van de client compileren, zonder aan de stub-klassen te raken. Vergeet na het compileren van de clients de SSL-verplichting op de Web services niet weer in te schakelen en nog een laatste maal opnieuw te deployen.

Bij eventuele wijzigingen aan de server is het verplicht om zowel `clean` als `build` op te

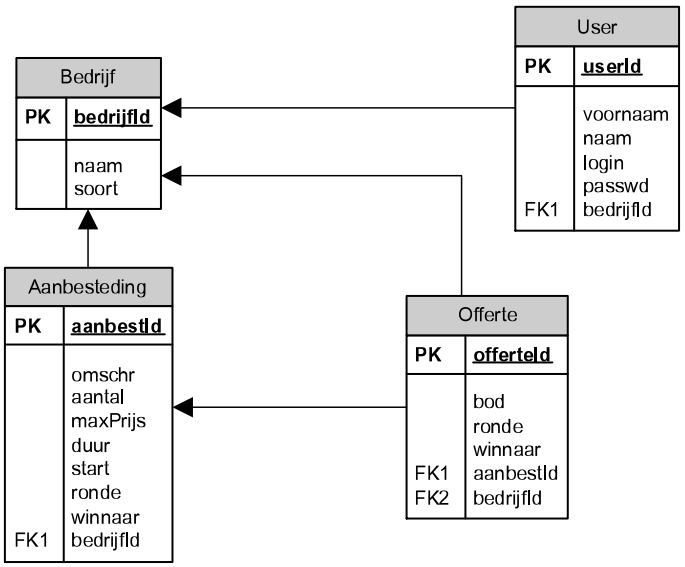
roepen. Deploytool heeft bovendien de vervelende eigenschap om niet te merken dat door het hercompileren ook `deploy.war` werd gewijzigd. Aanpassingen in die war-file (dus aanpassingen aan de Web services zelf of hun security) zullen pas merkbaar zijn als je ProcService d.m.v. deploytool verwijdert uit de applicatie en `deploy.war` opnieuw importeert. Dit importeren gebeurt door de ProcApp te selecteren en in het menu 'File → Add to Application → Web Application WAR...' te selecteren en de war-file te laden uit `<server_home>\dist\`.

Als nu zowel de server als het DBMS draaien en de applicatie is naar behoren gedeployed, kunnen de clients worden gestart met de batch-file `run`. Dit roept een ASant-task op die, met het juiste classpath, de hoofdklasse `Main` uitvoert.

5.2 Databank

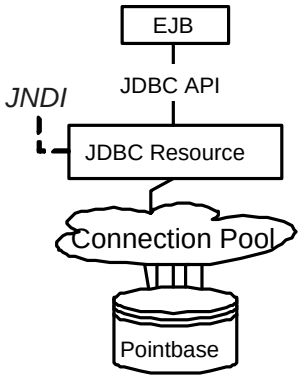
Met het pakket van de Sun Application Server wordt een DBMS, Pointbase, meegeleverd. Pointbase [5] is een klassieke relationele SQL-databank, volledig geschreven in Java. De hele onderliggende databank kan worden geïnitialiseerd door het SQL-bestand (`procdb.sql`) te laden in de Pointbase-console en uit te voeren (Execute All). De console vind je in `<J2EE_HOME>\pointbase\tools\serveroption`. Figuur 5.2 geeft het schema van de databank van de procurement-server weer. De vier businessobjecten: User, Bedrijf, Aanbesteding en Offerte, worden voorgesteld door vier afzonderlijke tabellen. Door het gebruik van verwijssleutels worden elke aanbesteding en elke offerte aan het betreffende bedrijf gelinkt, wijst uiteraard elke offerte naar de aanbesteding waarop die reageert en behoort elke gebruiker tot een bedrijf. De applicatie vereist dat loginnamen van gebruikers en namen van bedrijven uniek zijn. Dit wordt gecontroleerd door de databank. Het is verder belangrijk om te zien dat alle primaire sleutels van het type `NUMERIC(19)` zijn. Dit formaat is vereist om de id's te kunnen bevatten die automatisch door CMP worden aangemaakt (zie § 5.3.1).

Om de databank te laten communiceren met de applicatieserver, dienen we te beschikken over een Java Database Connectivity (JDBC) Resource. Vanuit de applicatieserver kan naar deze resource worden verwezen via de Java Naming and Directory Interface (JNDI). Alle objecten in het systeem (EJB's, Resources,...) kunnen een JNDI-naam krijgen waarmee vanuit andere modules naar hen kan worden verwezen. De JDBC Resource kreeg hier de naam `jdbc\Procurement`. Deze JDBC Resource kiest zijn verbindingen uit een Connection Pool (hier toepasselijk de ProcurementPool). De JDBC middleware zorgt er dus voor dat de applicatie zelf enkel de JDBC-API hoeft te spreken met de resource, gevonden via JNDI. Door het gebruik



Figuur 5.2: Relationele databank

van EntityBeans wordt bovendien ook deze JDBC-communicatie volledig transparant voor de programmeur (zie § 5.3.1). De JDBC Resource gebruikt een verbinding uit de pool en zorgt verder voor de communicatie met Pointbase (Fig. 5.3).



Figuur 5.3: Opbouw van JDBC Resources

Zowel JDBC Resources als Connection Pools kunnen aangemaakt en beheerd worden via de webinterface van de applicatieserver (Admin Console). In de Connection Pool wordt expliciet verwezen naar de databank en kunnen verschillende opties, zoals aantal verbindingen en wachtwoord, worden ingesteld.

5.3 Enterprise JavaBeans

Aan de serverzijde van een J2EE-applicatie worden alle objecten voorgesteld als Enterprise JavaBeans. Een EJB bestaat telkens uit twee interfaces en één klasse. De home-interface (extends `EJBHome` of `EJBLocalHome`) definieert de statische methoden van de bean: methodes om een nieuwe bean te creëren en bestaande beans op te zoeken. De bean-interface (extends `EJBObject` of `EJBLocalObject`)definieert dan de methoden waar de gebruiker van de bean toegang tot krijgt (getters en setters, business-methoden,...). De bean-klasse zelf zorgt voor de implementatie van alle methoden die in de interfaces werden opgegeven.

In theorie zouden lokale interfaces enkel op de server te gebruiken zijn, terwijl 'gewone' interfaces ook remote kunnen worden opgeroepen. Aangezien het gebruik van remote interfaces echter problemen gaf bij het gebruik van EntityBeans, werd ervoor geopteerd om de EntityBeans van een lokale interface te voorzien die wordt aangesproken door verschillende SessionBeans. Deze SessionBeans hebben dan zelf een remote interface die bereikbaar is voor de Web services (zie overzicht op Fig. 2.11). Meer informatie over werken met EJB's staat in de online J2EE-tutorial [8].

5.3.1 Entity Beans

EntityBeans (implements `EntityBean`) stellen de business-objecten van de applicatie voor. Zoals gewone klassen kunnen ze gecreëerd worden en kunnen er d.m.v. allerhande methoden bewerkingen op worden uitgevoerd. Het mooie aan EntityBeans is echter dat ze volledig persistent zijn, wat wil zeggen dat elke wijziging onmiddellijk in de databank wordt opgeslagen. Programmeurs die volledige controle willen over het aanspreken van de databank kunnen zelf SQL-opdrachten linken aan elke methode. Het is echter uitermate handig om Container Managed Persistence (CMP) in te schakelen, waardoor alle communicatie met de databank volledig door de EJB-container wordt beheerd en de programmeur hier zelf bijna niet meer in moet tussenkomen.

Om CMP te gebruiken krijgen de EntityBeans geen werkelijke attributen mee. Voor elk attribuut worden enkel abstracte getters en setters gedefinieerd. Ook onderlinge relaties tussen de beans (bvb. een aanbesteding omvat meerdere offertes, een gebruiker behoort tot een bedrijf, enz...) worden impliciet beschouwd als attributen. Ook hiervoor wordt niet het attribuut zelf gedeclareerd, maar wel abstracte getters en setters. De declaraties van de bean Aanbesteding zien er bvb. als volgt uit:

```
public abstract String getOmschr();
```

```
public abstract void setOmschr(String s);

public abstract int getAantal();
public abstract void setAantal(int a);
...

//Access methods relationship fields
public abstract void setOffertes(Collection offertes);
public abstract Collection getOffertes();

public abstract void setBedrijf(LocalBedrijf bedrijf);
public abstract LocalBedrijf getBedrijf();
```

Zoals je kunt zien, wordt er niet gesproken over de primaire sleutel. Deze wordt niet gedeclareerd, maar is van het type Object (het Object bevat in werkelijkheid een getal van het type long). De constructors van een EntityBean geven steeds de primaire sleutel terug als een dergelijk Object. Let er echter op dat de methode wel wordt beëindigd met een `return null`.

```
//EJB methods
    public Object ejbCreate(String omschr, int aantal, int max, int duur,
     Object bedrijfId)
           throws CreateException {
        try {
            bHome = lookupBedrijfHome();
        } catch (NamingException e) {
            e.printStackTrace();
        }
        setOmschr(omschr);
        setAantal(aantal);
        setMax(max);
        setDuur(duur);
        setRonde(1);
        setHeeftWinnaar(false);
        //zet de huidige datum
```

```
        setStartDatum(new Date((new java.util.Date()).getTime()));
        return null;
    }
```

EJB's kunnen op verschillende niveau's namen toegekend krijgen. Allereerst is er de absolute naam van de bean die bestaat uit de naam van de jar-file waarin hij is opgenomen en zijn bean-naam zoals toegekend bij het creëren in deploytool. Verder kan aan elke bean, eveneens in deploytool, een JNDI-naam worden toegekend. Tenslotte verwijst programma-code naar een willekeurige alias voor een bean (bvb. `ejb/Aanbesteding`). Deze verwijzingen vanuit de code kunnen, alweer met deploytool, worden gemapped op absolute- of JNDI-namen.

In de constructor die hierboven wordt getoond, wordt de home-interface van de Bedrijf-bean opgezocht. Dit gebeurt aan de hand van de willekeurige referentienaam die in deploytool wordt vertaald. In de post-constructor kunnen de relaties tussen verschillende beans worden geïnitialiseerd (dit is niet toegelaten in de constructor). Hier wordt de home-interface `bHome` dus gebruikt om aan de hand van een primaire sleutel (het Object `bedrijfId`) de EntityBean van het gevraagde bedrijf op te zoeken. Voor dit doeleinde wordt de ingebouwde methode `findByPrimaryKey` gebruikt. Vervolgens wordt de relatie tussen aanbesteding en bedrijf gelegd door de lokale interface van het bedrijf dat we gevonden hebben als attribuut van de aanbesteding op te geven d.m.v. de abstracte setter. Let er op dat dus nooit de EntityBean zelf wordt aangesproken, maar steeds zijn interface.

```
    public void ejbPostCreate(String omschr, int aantal, int max, int duur,
      Object bedrijfId)
            throws CreateException {
        try{
            LocalBedrijf comp = bHome.findByPrimaryKey(bedrijfId);
            setBedrijf(comp);

        }catch (FinderException e){
            e.printStackTrace();
        }
    }
```

Verder in de bean worden nog methoden opgegeven die zijn levensloop bepalen. Gebruikte home-interfaces van andere beans kunnen geladen worden bij het activeren van de bean en

worden op null gezet wanneer de bean passief wordt. Ook destructors ed. kunnen worden geïmplementeerd.

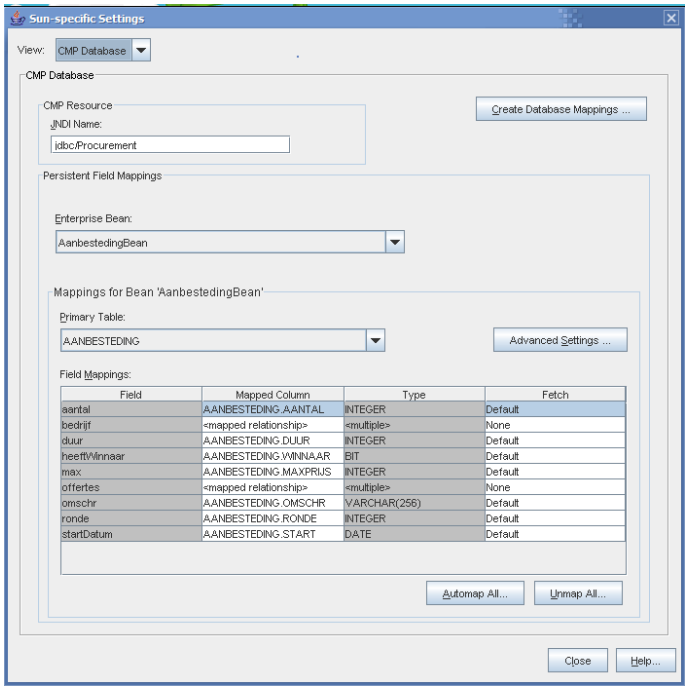
Eens alle beans geïmplementeerd zijn, kunnen ze worden geassembleerd in deploytool. Alle (door javac gecompileerde) code wordt geladen en komt samen in een jar-file terecht. Eén voor één kunnen de beans dan worden aangemaakt (New Enterprise Bean). Type van de bean (Entity of Session) alsook lokale en home-interfaces worden opgegeven. In de Entity tab wordt opgegeven welke van de abstracte velden effectief moeten persistent gemaakt worden. De relaties tussen de verschillende beans worden vastgelegd in de algemene instellingen van de jar-file (vandaar dat alle EntityBeans die met elkaar in relatie staan, best in dezelfde jar zitten). Er kan nu een bestaand attribuut worden gekozen om de primaire sleutel in op te slaan. In deze applicatie wordt echter ook dit aan de container overgelaten en wordt dus gekozen voor Unknown Primary Key (java.lang.Object).

Nu moeten de velden van de bean nog gemapped worden op de kolommen van de databank. Met het java-programma CaptureSchema (bij het builden opgeroepen vanuit ASant) wordt het schema van de databank opgeslagen. Dat schemabestand moet worden geïncludeerd in de jar-file en ook geladen worden in deploytool. Deploytool zal nu zelf de mapping uitvoeren (die meestal zal kloppen als de namen van de bean-velden dezelfde zijn als de namen van de kolommen uit de databank) (Fig. 5.4).

Een belangrijke functie van de home-interface die nog niet werd besproken, is het opzoeken van reeds bestaande beans. In het voorbeeld hierboven werd reeds gebruik gemaakt van de voorgedefinieerde functie `findByPrimaryKey`. Dergelijke finder-functies kunnen ook zelf aangemaakt worden. Het volstaat om ze te declareren in de home-interface. In deploytool krijg je dan de mogelijkheid om per functie een EJB QL-statement op te maken. Deze EJB Query Language lijkt sterk op SQL en laat toe om queries op de beans uit te voeren, zonder de eigenlijke structuur van de databank te moeten kennen.

5.3.2 Session Beans

Een SessionBean (implements `SessionBean`) bevat geen persistente informatie. Een statefull SessionBean (zoals gebruikt in deze applicatie) onthoudt de inhoud van zijn attributen gedurende één sessie, een stateless SessionBean doet zelfs dat niet. Hier werden de SessionBeans gebruikt als extra interface tussen de Web services en de EntityBeans (zie Fig. 2.11). De methoden werden gegroepeerd in twee delen: alle methoden die expliciet de aanbestedingen en offerten



Figuur 5.4: Het mappen van een EntityBean op de databank

beheren worden verzameld in de ProcEngine. De methoden voor gebruikersbeheer zitten in UserEngine. Uiteraard is het voor de ProcEngine af en toe nodig om ook gebruikersgegevens te gaan opvragen. Omgekeerd is dit niet het geval. Bij het aanmaken of activeren van de SessionBeans worden de home-interfaces geladen van de EntityBeans die gaan gebruikt worden (vandaar statefull). Wanneer de bean inactief wordt, verdwijnen ook die home-interfaces weer uit het geheugen.

Verder verloopt het creëren van een SessionBean volledig als bij een EntityBean. Er zijn enkel geen persistentie-instellingen nodig. Bij SessionBeans is het zeker van belang om in de lijst met EJB-refertenties alle verwijzingen naar een andere bean vanuit de code op te sommen en te mappen op de juiste JNDI-naam.

5.4 WebServices: JAX-RPC

Zoals beschreven in § 5.4 zijn er voor het opzetten van een Web service met JAX-RPC verschillende componenten nodig: een interface, een implementatie van die interface, een WSDL-file die deze interface beschrijft en een stub-klasse die door de client kan aangesproken worden.

Voor we aan client-zijde deze stubs kunnen genereren, moet eerst de Web service zelf op

poten staan. Aan server-zijde stellen we dus eerst de interface van de webservice op en implementeren we die in een afzonderlijke klasse. Aan de hand van de interface kan wscompile nu de WSDL-file genereren. Daarnaast maakt wscompile ook `mapping.xml` aan, waarin java-types worden vertaald naar SOAP-types en een `model.tar.gz`, dat later wordt gebruikt door wsdeploy. Wscompile haalt de gegevens over de Web service en de plaats van de interface uit een XML-bestand.

Het omgekeerde is trouwens ook mogelijk: maak een WSDL-file aan en laat wscompile de interface in java genereren. Hier heb je echter minder controle over de gebruikte java-types. Het volledige gebruik van wscompile vind je op de manual-pages [4]. Helaas wordt er daar nog met geen woord gerept over security-opties.

Eens de WSDL- en mapping-file bestaan, kun je een eenvoudige Web services al laden in deploytool. Helaas heeft ook deploytool tot op dit moment nog niet gehoord van de meer geavanceerde XWS-security-opties (zie § 5.6.3). Om dus te bekomen dat de Web service rekening houdt met de opgegeven security-voorkeuren, moet je de war-file zelf aanmaken, i.p.v. dit over te laten aan deploytool. Dit kan gelukkig allemaal ook vlot gebeuren met behulp van ant-taks. Met jar worden de interface en implementatie-klassen verpakt in een 'rauwe' war-file. Hierbij wordt een vastgelegde directory-structuur gevolgd, waarbij alle gecompileerde klassen terecht komen in `WEB-INF\classes` en de wsdl-file in `WEB-INF\wsdl`. De gegenereerde mapping- en model-files komen in de root van deze structuur terecht. Verder zijn hier nog twee extra XML-bestanden nodig. `web.xml` geeft een extra beschrijving op van de service. Hier heb ik ook de bean-referenties die in de code van de implementatie worden gebruikt, gemapped op de juiste beans. Dit kan gemakkelijker worden gedaan in deploytool, maar zoals in § 5.1.2 vermeld, moet je geregeld de war-file met Web services terug uit de applicatie verwijderen. Telkens je dit doet zouden ook alle instellingen gereset worden. Daarom is het handiger om deze direct met de configuratiebestanden mee te geven. `jaxrpc-ri.xml` beschrijft de verschillende endpoints met links naar het model en de wsdl-file. Hier merk je het security-belang van het zelf in elkaar steken van de war-file. In deploytool is er nergens kans om te verwijzen naar de model-file, terwijl dit het enige bestand is waar er wordt verwezen naar de security-opties:

```
<webServices xmlns="http://java.sun.com/xml/ns/jax-rpc/ri/dd"
             version="1.0">
   <endpoint name="UserImpl"
    displayName="UserService"
```

```
    description="UserService for ProcApp"
    port="{http://localhost:8080/UserImpl}UserImpl"
    interface="webservices.UserIF"
    implementation="webservices.UserImpl"
    model="/WEB-INF/user-model.xml.gz"
    wsdl="/WEB-INF/wsdl/UserService.wsdl"/>
...
   <endpointMapping endpointName="UserImpl"
    urlPattern="/user"/>
...
</webServices>
```

Met de tool wsdeploy wordt de bekomen war-file vervolgens getransformeerd tot een 'deployable' war die wel kan geladen worden in deploytool. Daar kunnen dan de extra opties worden ingesteld. Hier was het enkel nog nodig om een context root in te stellen en het gebruik van SSL te forceren (zie § 5.6.2). De context root, samen met de hierboven ingestelde endpointName, zorgt voor de url waarop de Web service later beschikbaar is.

Eenmaal de war-file volledig is geïntegreerd in de ear-file, kan gedeployed worden en is de Web service beschikbaar voor de clients. Zoals beschreven bij de installatie (§ 5.1.2) kunnen de clients nu de WSDL lezen en aan de hand daarvan hun stub-klassen genereren. Opdat wscompile aan de client-zijde zou weten waar de gedeployede WSDL-file zich bevindt, moet ook hier nog een extra XML-bestand worden meegegeven met de locatie van de WSDL en de naam van het package waarin de Web service interface zich bevindt.

In de code van de client wordt nu als volgt gebruik gemaakt van de stub:

```
stub = createProxy;
stub._setProperty(Stub.ENDPOINT_ADDRESS_PROPERTY, endPointAddress);
hospServ = (HospIF) stub;

private static Stub createProxy() {
    // Note: HospitalProcService_Impl is implementation-specific.
    return (Stub) (new HospitalProcService_Impl().getHospIFPort());
}
```

Het object `hospServ` kan nu gebruikt worden alsof het om een gewone lokale klasse gaat.

Om de code van de clients overzichtelijk te houden werd er wel voor geopteerd om alle oproepen van de stub te beperken tot het package 'webservice'.

5.5 Architectuur client

Beide clients zijn heel gelijkaardig opgebouwd. Een client bestaat uit vijf packages: business, gui, webservice, security en utils.

Het package `business` omvat o.a. de business-objecten. Aangezien de meeste functionaliteit in de server zit, zijn de business-objecten (`Aanbesteding`, `User`, `Offerte`, `Bedrijf`) vooral verzamelingen van attributen met getters en setters.

Hier bevindt zich ook de `Main`-klasse waar enkele instellingen geïnitialiseerd worden.

De `ResourceLoader` uit het package `utils` wordt hier ingesteld. `ResourceLoader` is een zelfgeschreven klasse met enkel statische attributen en methoden. Deze klasse wordt gebruikt om vlot in elke module waar er Nederlandse tekst voorkomt een `ResourceBundle` te laden met de juiste landinstellingen. Een `ResourceBundle` verwijst naar een lokaal instellingenbestand waarin elke Nederlandse uitdrukking aan een trefwoord wordt gekoppeld. In de ziekenhuis-client wordt bvb. gerefereerd naar het bestand `hospclient_nl_BE.properties`. Door dit bestand nu te vertalen naar `hospclient_en_UK.properties`, kan de hele applicatie in het Engels verlopen zonder dat de code moest worden gewijzigd.

De gewenste taal is een van de zaken die zit opgeslagen in het bestand `hosp.ini`. Dit is een tekstbestandje met naam-waarde-koppels waar verder de laatst gebruikte instellingen van de client (server, certificaten,...) in zijn opgeslagen. Aangezien er toch geregeld iets moet worden gewijzigd aan dit ini-bestand, wordt er gebruik gemaakt van de klasse `IniManager` (ook in package `utils`) die het, eveneens met statische methoden, gemakkelijk maakt om instellingen op te halen of terug op te slaan.

De laatste, misschien overbodige, feature die ingesteld wordt alvorens het programma te starten, is de 'look-and-feel'. Om een iets vlottere uitstraling dan het gemiddelde java-programmaatje te bekomen, werd het uiterlijk gebruikt van MS OfficeXP.

Het eigenlijk programma start wanneer het `Hoofdvenster` (package `gui`) wordt aangemaakt. In het hoofdvenster wordt onmiddellijk een model aangemaakt (`business.AlgemeenModel`) waar verder alle informatie wordt opgeslagen die de client nodig heeft tijdens het uitvoeren. Hier wordt bijgehouden welke gebruiker is ingelogged, tot welk bedrijf hij behoort, waar de server en de certificaten zich bevinden, een link naar het hoofdvenster... Het model wordt steeds meegegeven

aan subvensters, zodat die steeds beschikken over alle nodige informatie. Verschillende vensters observeren het model (`AlgemeenModel` implementeert `Observable`), zodat ze onmiddellijk reageren op wijzigingen in de gegevens (knoppen die al dan niet beschikbaar moeten zijn, ...). Om het overzichtelijk te kunnen houden wordt voor de lijsten van aanbestedingen en offerten een extra model gebruikt om tijdelijke informatie in op te slaan (`AanbestDetailModel` en `OfferteDetailModel`).

Alle vensters uit package `gui` zijn extensies van `JPanel` of `JSplitPane`. Ze worden bij het oproepen door het hoofdvenster vervat in een `JInternalFrame`, zodat ze als losse vensters kunnen worden gemanipuleerd in het `JDesktopPane` van het hoofdvenster.

Alle verbindingen met de Web services gebeuren vanuit het package `webservice`. De klassen `ProcVerzender` en `UserVerzender` maken een instantie van de stubklasse aan en bieden de methoden aan aan de andere modules van de client. Het laden van gegevens van een gebruiker gaat bvb. als volgt:

```
user = new User(login, pass);
userSend = new UserVerzender(model);
userSend.loadUserInfo(user);
```

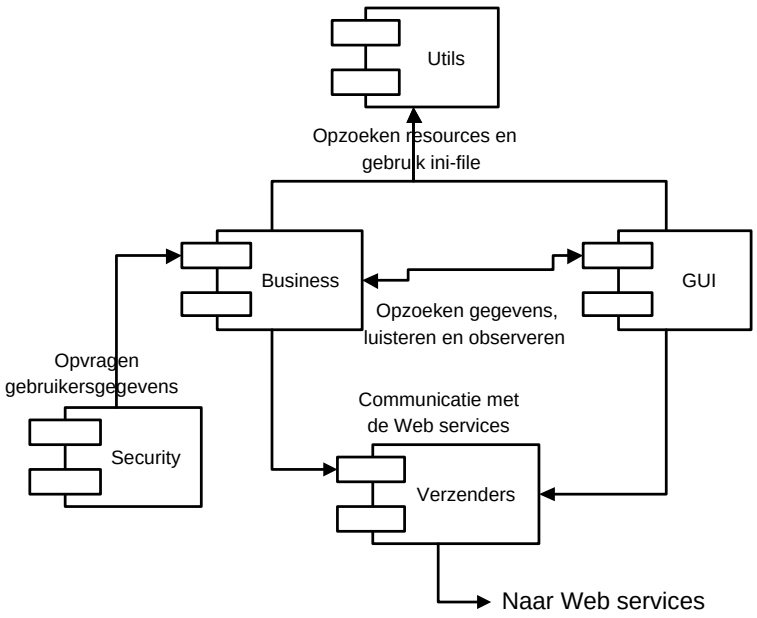
De verzenderklasse gaat dan, op basis van de attributen die reeds zijn ingevuld in de klasse, met opeenvolgende vragen aan de Web service de overige attributen invullen. Het model wordt meegegeven aan de verzenderklasse zodat hierin o.a. de url van het service endpoint kan worden opgezocht.

Het package `security` bevat de CallbackHandlers (zie § 5.6.3).

De relaties tussen de verschillende packages wordt weergegeven in figuur 5.5.

5.6 Security

Bij het ontwikkelen van de applicatie werd er sterk rekening gehouden met het voorkomen van kwaadwillig gebruik ervan. Het is, zoals in de security-analyse werd beschreven (zie §4.1), niet de bedoeling dat bedrijven de nieuwe mogelijkheden die een e-procurement-server biedt, aangrijpen om de onderhandelingen te vervalsen. Uit die security-analyse bleek dat dit kan verwezenlijkt worden door de webservices over een SSL-verbinding te laten verlopen, alle SOAP-berichten van een login en wachtwoord te voorzien en die berichten, gebruikersnaam inclusief, te ondertekenen.



Figuur 5.5: Architectuur van de clients

Alle berichten die op de server toekomen zouden tenslotte in een log-bestand moeten worden verzameld en bijgehouden.

5.6.1 Certificaten

Zowel voor het opzetten van een SSL-verbinding als voor het ondertekenen van de SOAP-berichten zijn er certificaten nodig (zie § 4.2). Deze kunnen worden gegenereerd met de key-tool [3] die ook bij de applicatieserver wordt geleverd. Met keytool creëer je voor de client een sleutelpaar volgens het RSA-algoritme. De private sleutel komt onmiddellijk terecht in de keystore, de geheime sleutelring van de client. Om nu de server in staat te stellen de client te authentiseren, wordt, weer met keytool, een certificaat (een ondertekende publieke sleutel, die ook de identiteit van de client bevat) geëxporteerd uit de keystore van de client en toegevoegd aan de truststore van de server. De truststore is de verzameling van certificaten die vertrouwd worden. Omgekeerd, wanneer er zoals hier wederzijdse authenticatie nodig is, exporteren we het certificaat van de server uit zijn keystore en importeren het in de truststore van de client. De procurement-server kan optreden als CA (Certificate Authority) en zelf de certificaten ondertekenen. Bij installatie zijn er reeds een key- en truststore voor de server aangemaakt. Er wordt steeds gewerkt met Java KeyStore-bestanden (jks). Deze jks-bestanden, één keystore en één truststore, worden bij registratie van een nieuw bedrijf door de administrator op een drager

gezet en aan de nieuwe klant meegegeven.

5.6.2 SSL

Het gebruik van SSL in Java Webservices is niet moeilijk te implementeren, eenmaal er geldige keystores beschikbaar zijn. De locatie van elke keystore wordt bewaard als String en wordt als systeem-eigenschap opgegeven voor het aanmaken van de stub. De code blijft dus dezelfde als in § 5.4, alleen worden volgende instellingen nog tussengevoegd:

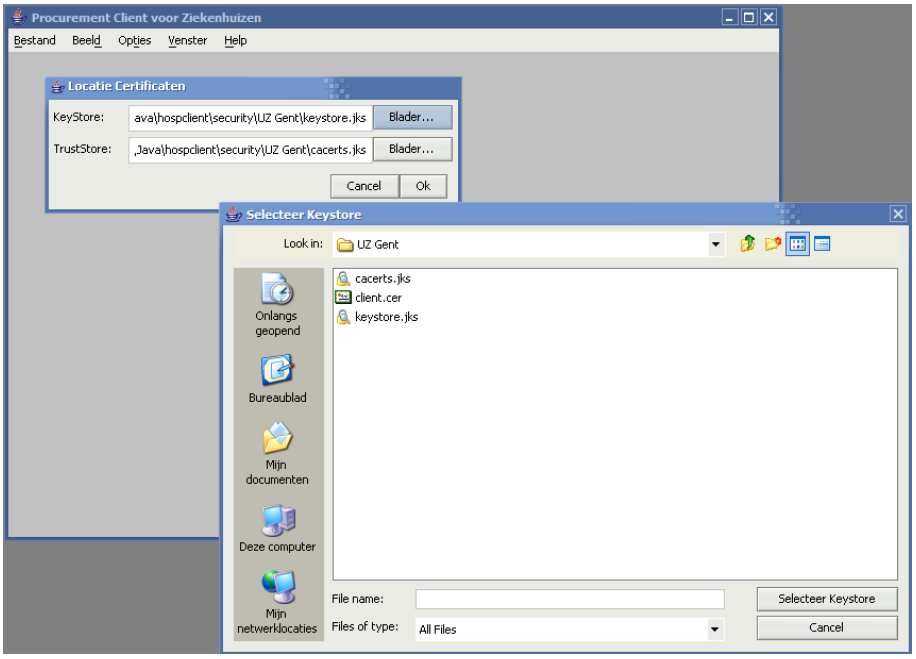
```
//SSL-verbinding
System.setProperty("javax.net.ssl.keyStore", keyStore);
System.setProperty("javax.net.ssl.keyStorePassword", keyStorePassword);
System.setProperty("javax.net.ssl.trustStore", trustStore);
System.setProperty("javax.net.ssl.trustStorePassword",
                trustStorePassword);
```

Belangrijk is wel dat het wachtwoord van een sleutel of certificaat identiek is aan het wachtwoord van de keystore waarin het zich bevindt. Dit staat niet vermeld in de tutorial, maar werpt wel keer op keer een exception.

De locaties van de trust- en keystore kunnen worden opgegeven via de client (Fig. 5.6). Deze feature kan worden verwijderd eenmaal er beslist wordt dat certificaten enkel vanaf een bepaalde locatie (bvb. Smartcard) mogen worden geladen.

Als nu voor de URL van het Web service endpoint `https://` wordt gebruikt, dan verloopt de communicatie over een SSL-verbinding. Het blijft nu wel nog de keuze van de gebruiker of hij al dan niet een beveiligde verbinding kiest. In deze applicatie is dit niet de bedoeling. SSL wordt hier immers niet enkel voor encryptie van data en voor authenticeren van de server gebruikt, maar ook om zeker te zijn van de identiteit van de client. Vanuit deploytool kan het gebruik van SSL verplicht worden. In het security-tabblad van de webservice kan gekozen worden voor 'Client Certificate' als authenticatie-methode. Hierna voeg je een SecurityConstraint bij, met als requirement 'Confidential', zodat er steeds SSL wordt gebruikt, zelfs als het http-protocol werd opgegeven. Met een Web Resource Collection geef je op welke pagina's of URL's er beveiligd dienen te worden (in dit geval gaat het om de endpoints `\hosp`, `\dealer` en `\user`).

Als de server geen foutmelding geeft wanneer je als URL voor een Web service het https-protocol gebruikt, kun je er al behoorlijk zeker van zijn dat de verbinding over SSL verloopt. Om helemaal een bewijs van deze beveiliging te hebben kun je bij het runnen als extra argument voor



Figuur 5.6: Selecteren van key- en truststore

de JVM `-Djavax.net.debug=all` gebruiken. Zo wordt de volledige SSL-handshake gedumpt op de standaard output.

Ook voor de procurement-administrator is er een SSL-verbinding nodig. Hij stuurt immers een wachtwoord dat niet mag onderschept worden door vanaf een jsp-pagina. In principe mag ook geen enkele andere gebruiker deze jsp-pagina kunnen bezoeken, al zou dit geen grote ramp zijn. Een nieuw aangemaakt bedrijf wordt immers pas erkend door de server wanneer z'n certificaat werd toegevoegd aan de truststore. Het volstaat dus om een éénrichtings-SSL-verbinding op te stellen en de site te beveiligen met een wachtwoord. Dit wordt op gelijkaardige manier verwezenlijkt als de SSL-verplichting voor JAX-RPC hierboven. We kiezen hier echter wel voor een basic authentication, zodat geen certificaat, maar wel een login en wachtwoord wordt verwacht.

De applicatieserver werkt wat gebruikersbeheer betreft op twee fronten: enerzijds zijn er de typische gebruikers, gedefinieerd op de server in het 'file realm'. Deze gebruikers kunnen, op een gelijkaardige manier als gebruikers van een linux-systeem, ingedeeld worden in groepen. Aan de kant van de web-applicatie zelf wordt er van 'roles' gesproken. Een bepaalde pagina kan slechts toegankelijk zijn voor een gebruiker die een bepaalde rol vervult, bvb. die van administrator. In de algemene, Sun-specifieke instellingen in deploytool, kunnen nu gebruikers en groepen een

bepaalde rol toegekend krijgen. Gebruikers kunnen worden aangemaakt in de Admin Console van de applicatieserver. Dit gebeurt in het Security, Realm-menu in het file-realm.

5.6.3 XWS-Security

SSL zorgt wel voor een uitgebreide beveiliging tijdens het versturen van de SOAP-berichten, maar van zodra een bericht aankomt is ook de beveiliging verdwenen. Dit zou niet erg zijn, ware het niet dat alle berichten als bewijs moeten opgeslagen blijven in een log-bestand. Uit dit logbestand moet kunnen opgemaakt worden wie welk bericht wanneer verstuurd heeft. Hiervoor werden dus aan elk bericht een aantal extra security-headers toegevoegd (zie § 3.2).

Zowel aan client- als aan serverzijde kunnen we in een XML-bestand opgeven welke extra headers er moeten toegevoegd worden en welke headers we verwachten van de andere kant. In deze applicatie werd door de client telkens een UsernameToken toegevoegd, wat ook verwacht wordt door de server. Verder verwacht de server dat elk bericht, inclusief het UsernameToken, gesigneerd is met behulp van de private sleutel van de client. Een Timestamp wordt niet verwacht door de server, maar wordt wel automatisch bijgevoegd door de client.

Aan clientzijde gebruiken we dit:

```
<xwss:JAXRPCSecurity xmlns:xwss="http://java.sun.com/xml/ns/xwss/config">
    <xwss:Service>
        <xwss:SecurityConfiguration dumpMessages="false">
            <xwss:UsernameToken id="user-token" digestPassword="false"/>
            <xwss:Sign>
                <xwss:X509Token certificateAlias="client-alias"/>
                <xwss:Target type="xpath">//SOAP-ENV:Body</xwss:Target>
                <xwss:Target type="uri">#user-token</xwss:Target>
            </xwss:Sign>
        </xwss:SecurityConfiguration>
   </xwss:Service>

    <xwss:SecurityEnvironmentHandler>
        security.ClientSecurityEnvironmentHandler
    </xwss:SecurityEnvironmentHandler>
</xwss:JAXRPCSecurity>
```

Door dumpMessages uit te schakelen geven we aan dat de verstuurde en ontvangen berichten niet op de standaard output moeten worden getoond. Alvorens te ondertekenen geven we aan dat gebruikersnaam en login worden meegstuurd. Door het wachtwoord niet te digesten, geven we aan dat we gewoon gebruik willen maken van plaintext wachtwoorden. Dit kan geen kwaad, aangezien de volledige verbinding SSL-beveiligd is. Voor het ondertekenen van de berichten gebruiken we het certificaat dat in de keystore zit opgeslagen onder de naam 'client-alias'. Verder geven we alle onderdelen van het bericht op die ondertekend moeten worden worden. Hier zijn dit de body en het UsernameToken.

```
<xwss:JAXRPCSecurity xmlns:xwss="http://java.sun.com/xml/ns/xwss/config">
    <xwss:Service>
        <xwss:SecurityConfiguration dumpMessages="true">
           <xwss:RequireSignature>
                <xwss:Target type="xpath">//SOAP-ENV:Body</xwss:Target>
                <xwss:Target type="qname">{http://....xsd}UsernameToken
                </xwss:Target>
            </xwss:RequireSignature>
            <xwss:RequireUsernameToken passwordDigestRequired="false"/>
        </xwss:SecurityConfiguration>
    </xwss:Service>

    <xwss:SecurityEnvironmentHandler>
        security.ServerSecurityEnvironmentHandler
    </xwss:SecurityEnvironmentHandler>
```

De server, die wel zijn berichten laat uitschrijven voor log-doeleinden, hoeft zelf niets te doen op beveiligingsvlak, maar heeft wel verwachtingen t.o.v. de client. Eerst gaat hij na of er wel een handtekening aanwezig is op body en UsernameToken om vervolgens dat token zelf te controleren.

XWS-Security deelt zijn verwachtingen mee aan client of server d.m.v. Callbacks. Zowel client als server definiëren dus een CallbackHandler waarin telkens wordt nagegaan welke Callback er moet behandeld worden. Zoals in de code hierboven te zien is, wordt ook deze CallbackHandler aangeduid in de security-XML-file.

Client

UsernameCallback: De client moet de username ophalen.

PasswordCallback: De client moet het wachtwoord ophalen. Login en wachtwoord zitten gedurende de hele sessie opgeslagen in het model van de client, waar de CallbackHandler toegang toe heeft. Er was een statische methode van de CallbackHandler voor nodig om het model door te spelen.

SignatureKeyCallback: De client laadt de keystore (de locatie hiervan is ook te vinden in het model), distilleert hieruit de private sleutel en ondertekent daarmee het bericht.

Server

CertificateValidationCallback: De server haalt zelf het gebruikte certificaat uit het binnenkomende bericht en geeft dit door aan een afzonderlijke klasse: de `X509CertificateValitorImpl`. Deze klasse onderzoekt enkel of het certificaat geldig is: wie het heeft ondertekend (hier steeds de server zelf), of het nog niet verlopen is... Het onderzoeken van het bericht op integriteit is hier overbodig, aangezien SSL daar reeds voor heeft gezorgd. Op het moment dat er een bedrijf een bepaalde transactie betwist kan het bericht uit de log-file worden gehaald en onderzocht worden op integriteit.

Verder werd aan deze klasse een extra functie toegevoegd: nl. het filteren van de naam van het bedrijf uit dit certificaat. Deze bedrijfsnaam wordt teruggegeven aan de CallbackHandler.

PasswordValidationCallback: Hier verwacht de server een loginnaam en een wachtwoord en geeft die onmiddellijk door aan de JaasValidator. Bovendien wordt aan deze validator de naam van het bedrijf meegegeven die hierboven werd bekomen. De JaasValidator roept nu de UserEngine aan om te controleren of het login/wachtwoord-paar wel voorkomt in de databank. Wanneer de gebruiker bestaat en het wachtwoord juist is, zoekt hij verder op tot welk bedrijf de gebruiker behoort. Dit mag niet verschillen van het bedrijf dat uit het certificaat werd gehaald. Het is dus heel belangrijk dat de administrator bij het aanmaken van een nieuw bedrijf in het organisatie-veld exact dezelfde naam opgeeft als de bedrijfsnaam die in de databank terecht komt.

Als een van de validators false teruggeeft, dan wordt de SOAP-request beantwoord met een SOAPException.

De CallbackHandler loopt telkens in een lus de binnengekomen callbacks af en kijkt van welk type ze zijn:

```
  public void handle(Callback[] callbacks) throws IOException,
            UnsupportedCallbackException {
    for (int i = 0; i < callbacks.length; i++) {

        if (callbacks[i] instanceof PasswordValidationCallback) {
            PasswordValidationCallback cb = (PasswordValidationCallback) callbacks[i];
            if (cb.getRequest() instanceof
             PasswordValidationCallback.PlainTextPasswordRequest) {
                // plain text handling
                cb.setValidator(new JaasValidator(bedrijf));
             } else {
                throw unsupported;
             }
...
         else {
             throw unsupported;
         }
     }
 }
```

Hier wordt opgemerkt dat het verplicht is een wachtwoord te controleren. Het UsernameToken zit automatisch vervat in de request. De JaasValidator die wordt ingesteld, moet de methode `boolean validate(PasswordValidationCallback.Request request)` implementeren. Binnen deze methode voer je de tests uit die nodig zijn om een gebruiker te valideren en geef je een boolean terug om te laten weten of de gebruiker al dan niet geldig is.

5.6.4 MD5

Om wachtwoorden niet leesbaar te hoeven opslaan in de databank, worden ze allemaal gehashed wanneer ze door de UserEngine passeren. Het hashen gebeurt met het MD5-algoritme dat java standaard ondersteunt (zie de functie `hashPass()`). Bij elk wachtwoord wordt dus wat

'zout' toegevoegd, een willekeurig woord waarmee het wachtwoord verlengd wordt, om het vervolgens om te zetten naar een output van 128 bits. Het bij het inloggen opgegeven wachtwoord wordt uiteraard ook eerst op dergelijke manier versleuteld om het te kunnen vergelijken met het wachtwoord in de databank. Om de hash-codes toch te kunnen weergeven, werden ze met het BASE64-algoritme terug opgesplitst in groepjes van 6 bits ASCII-karakters.

```
private String hashPass(String pass){
      MessageDigest md5=null;
      BASE64Encoder b64 = new BASE64Encoder();
      try {
        md5 = MessageDigest.getInstance("MD5");
      } catch (NoSuchAlgorithmException e) {
          e.printStackTrace();
      }
      md5.reset();
      pass = SALT + pass;
      md5.update(pass.getBytes());
      return b64.encode(md5.digest());
  }
```

5.7 Ontwikkelingsomgeving

Deze thesis werd volledig ontworpen, geïmplementeerd, getest en geschreven op een Acer Aspire 1524WLMi (AMD Athlon^{TM}64 3400+, 512MB DDR RAM) met als besturingssysteem Windows XP Home Edition SP2. Deze laptop moest vaak alles uit de kast halen voor het draaien van de toch zeer trage en veeleisende applicatieserver, zeker in combinatie met de editors die eveneens de JVM gebruiken.

5.7.1 Programmeren

Als Java-editor gebruikte ik **Eclipse 3.0**. De codegeneratie voor klassen, constructors, try/catch-blokken, getters en setters,... is zeer uitgebreid en gedetailleerd. Ook de auto-completion voor alle klassen, attributen en methoden is uitermate handig. Eclipse bevat standaard ook een zeer goede ondersteuning voor het bewerken van ant-files. Een kleine beperking bij het werken met

Web services, is dat wanneer client en server tot verschillende projecten behoren, Eclipse de interfaces van de endpoints niet herkent. Daardoor worden de betreffende klassen en methoden van de webservice steeds als error aangeduid, wat in sommige klassen het opsporen van werkelijke fouten wat bemoeilijkt.

Het genereren van de basiscode voor de Swing GUI's werd gedaan d.m.v. de Eclipse-plugin **Visual Editor 1.0.1**. Deze plugin levert goed werk voor de standaardopbouw van de verschillende vensters. Van zodra er echter iets meer geavanceerde opties aan de GUI worden toegevoegd, zoals het ophalen van de inhoud van de labels uit een properties-bestand of het gebruik van glue om knoppen ed. te positioneren, laat Visual Editor het al afweten. Ook jammer is dat SpringLayout (nuttig voor het maken van formulieren) nog niet wordt ondersteund.

Voor het snel editen van java- of xml-files kwam het immer handige **GNU Emacs 21.3.1** van pas.

5.7.2 Modelleren

Met **Poseidon for UML Community Edition 3.0.1** konden vlot duidelijke UML-diagrammen worden getekend. Alle use cases en sequentiediagrammen, alsook voorlopige klassediagrammen, werden hiermee gemodelleerd. Zoals de naam het echter zegt, biedt Poseidon for UML enkel een editor voor UML-diagrammen. De overige illustraties werden dan ook gecreëerd met **Microsoft Visio**.

5.7.3 Scriptie

De scriptie zelf werd gezet met **LaTeX** (of om precies te zijn met de Windows-variant **MiKTeX 2.4**). Als LaTeX-editor werd **TeXnicCenter** gebruikt, dat een vlotte organisatie van verschillende tex-bestanden en een duidelijk overzicht van de scriptie biedt. De layout werd gebaseerd op een licht gewijzigde versie van de **LaTeX template for Thesis book** van David De Reu.

De ingevoegde illustraties moesten d.m.v. **PStill** worden geconverteerd naar een bruikbaar pdf-formaat.

Bijna als vanzelfsprekend werden de slides bij de presentaties gemaakt met **Microsoft PowerPoint**.

Hoofdstuk 6

Verder onderzoek en conclusie

De procurement-applicatie die in deze thesis werd ontwikkeld kan nog veel uitbreidingen en verbeteringen ondergaan. Zo zouden gebruikers moeten kunnen beschikken over profielen waarin bepaald wordt wat hun toegangsrechten zijn (kunnen ze gebruikers aanmaken?, mogen ze enkel de aanbestedingen opvolgen of mogen ze ook zelf nieuwe aanbestedingen plaatsen?,...). Uiteraard moeten ook de attributen van alle business-objecten uitgebreid worden: zowel afzonderlijke gebruikers als bedrijven moeten in staat zijn gegevens als adres, e-mail, enz... op te geven.

Indien rekening gehouden wordt met de enkele instellingen die specifiek zijn voor de Sun-implementatie van J2EE, zou het mogelijk zijn om de procurement-service te installeren op een meer performante applicatieserver. De voornaamste zaken die Sun-specific zijn, zijn het mappen van EJB's op JNDI-namen, het gebruiken van roles voor het authoriseren van gebruikers en, bij het gebruik van CMP, het mappen van bean-attributen op velden uit een databank.

Ook een zeer mooie verdere uitbreiding zou een GUI zijn die het logbestand kan interpreteren. Nu staan alle opgeslagen berichten samen met de ondertekeningen ed. in één lang tekstbestand. Het zou veel gebruiksvriendelijker zijn als een afzonderlijke tool deze XML-files zou inlezen en zou controleren van wie de handtekeningen afkomstig zijn en of het bericht effectief ongewijzigd is gebleven. De berichten zouden dan netjes op tijdstip van ontvangst, op gebruiker of op bedrijf kunnen gerangschikt worden.

Momenteel bewijst deze applicatie enkel dat het nu volstrekt mogelijk is om Java Web services grondig te beveiligen, maar dat de technologieën nog volop aan het evolueren zijn. Waar Web services enkele jaren geleden nog als 'onbeveiligbaar' werden bestempeld, bestaat er nu een waaier aan oplossingen, al zijn die nog lang niet overal even grondig gedocumenteerd en zijn er nog grote wijzigingen tussen opeenvolgende versies.

Bibliografie

[1] J2EE API, http://java.sun.com/j2ee/1.4/docs/api/index.html

[2] JAVA API, http://java.sun.com/j2se/1.5.0/docs/api/

[3] Manual Keytool, http://java.sun.com/j2se/1.5.0/docs/tooldocs/windows/keytool.html

[4] Manual WScompile, http://docs.sun.com/source/817-6092/hman1m/wscompile.1m.html

[5] Pointbase Databank, http://www.pointbase.com/

[6] Netscape Communications Corp., Security Developer Central, http://developer.netscape.com/tech/security/

[7] Sun, http://java.sun.com

[8] Tutorial J2EE 1.4, http://java.sun.com/j2ee/1.4/docs/tutorial/doc/index.html

[9] Tutorial Java Swing, http://java.sun.com/docs/books/tutorial/uiswing/index.html

[10] Tutorial JWSDP 1.6, http://java.sun.com/webservices/docs/1.6/tutorial/doc/index.html