Portlets zijn op Java gebaseerde webcomponenten in portal-based webapplicaties. Een groot aantal leveranciers, waaronder IBM, BEA, Sun, Borland, Epicentric en deelnemers aan Apache Jetspeed heeft onder het Java Community Process een standaard voor portlets ontwikkeld. Sinds september 2003 is er de Java Portlets Specificatie (JSR168), die inmiddels in de voornaamste portals is geïmplementeerd. In dit artikel gaat Willem Koppenol nader in op de achtergronden en werking van deze gestandaardiseerde portlets.

Standaard Java Portlets volgens JSR168

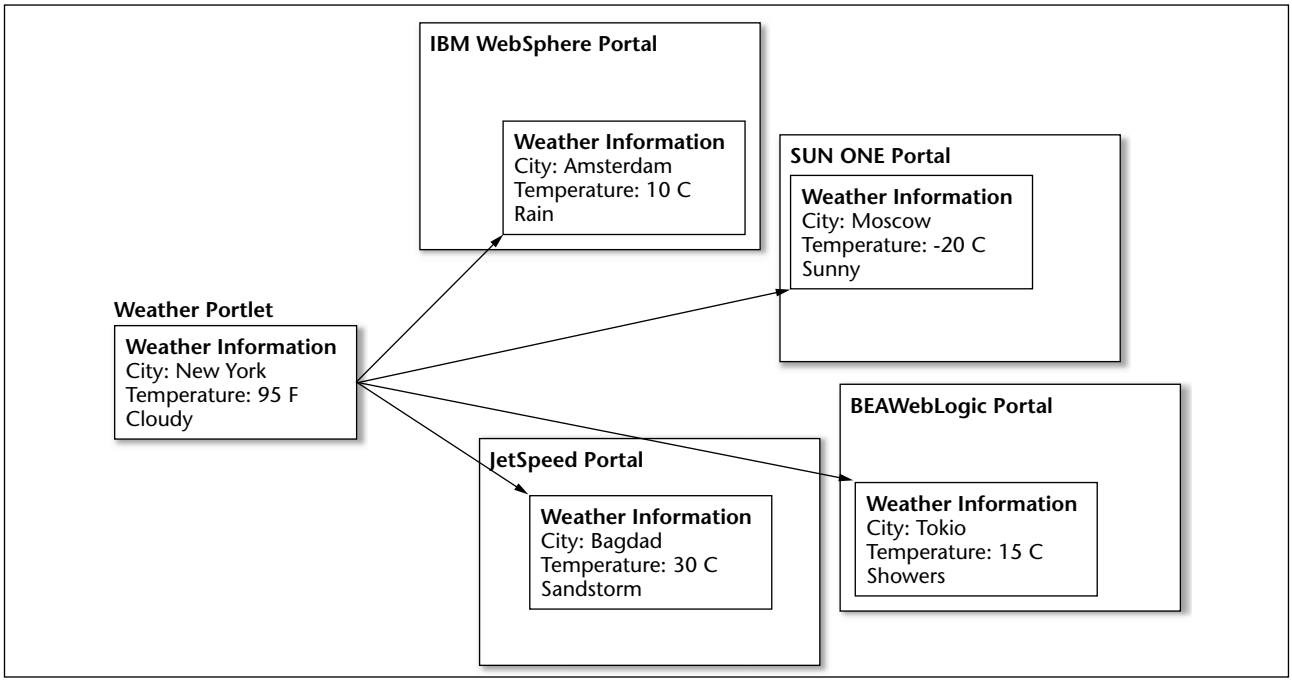
Nieuwe specificatie vervangt incompatibele API's

Tot voor kort was het alleen met mogelijk om portlets met vendor-specifieke API's te schrijven. Deze incompatibele API's veroorzaakten allerlei problemen voor software-ontwikkelaars, portal- vendors en portalgebruikers. Om die problemen op te lossen is de Java Portlet Specificatie (JSR-168) ontwikkeld. Ter illustratie van de concepten achter deze gestandaardiseerde portlets vind u in dit artikel pagina's en code van een WeatherPortlet dat de weersverwachting op een bepaalde locatie via een Web Service ophaalt en toont. In dit verband besteed ik ook aandacht aan de Web Service for Remote Portlets Standaard (WSRP) die in samenhang met JSR168 ontwikkeld en die het mogelijk maakt portlets van afstand als Remote Web Service in een portal in te pluggen.

PORTLETS Portlets zien eruit als kleine windows in het grotere user interface van een portal. Portlets verwerken user input en presenteren een dynamische output. Een portlet window bestaat uit en titel balk met zo mogelijk buttons om de portlet mode of de window state te wijzigen. Een portlet genereert een stukje markup, in bijvoorbeeld HTML, XHTML of WML voor wireless devices, dat 'fragment' wordt genoemd. Dit fragment vormt samen met andere fragments een compleet markup document, de portal page, voor presentatie aan de client. De fragments voldoen aan bepaalde regels zoals dat ze bijvoorbeeld geen <html> tags bevatten. De portal page bestaat meestal naast portlets ook uit andere elementen zoals banners en navigatie buttons. Portlets zijn configureerbaar, ondersteunen personalisatie en kunnen zich in verschillende toestanden bevinden.

PORTLET SPECIFICATIE De Portlet Specificatie JSR168 definieert een portlet API, een portlet container en het contract tussen de portlet componenten en de container. De portlet specificatie is eigenlijk een extensie op de servlet specificatie en legt vast hoe de levenscyclus van portlets verloopt, wat de portlet modes en window states zijn en hoe omgegaan moet worden met portlet preferences. JSR168 gaat niet in op aggregatie van portlets, lay-out management en het beheer en de configuratie van portals. Portlet componenten geschreven volgens de Portlet Specificatie kunnen in alle

FIGUUR 1. Gedeelte van een Portal page met Portlets



FIGUUR 2. JSR168 portlets : Write once deploy to many

JSR168 compliant portlet containers worden opgenomen. De Reference Implementatie (RI) van de Java Portlet Specification heet Pluto en is ondergebracht bij het Jakarta project van de Apache Open Software Foundation (www.apache.org). Pluto is een portlet container voor JSR168 portlets.

PORTLET CONTAINER Portlets draaien binnen de context van een portlet container zoals servlets draaien binnen de context van een servlet container. De container is verantwoordelijk voor de levens cyclus van portlets en voorziet ze van een run time omgeving. De container zorgt ook een opslag mechanisme voor portlet preferences en voor caching van portlets. De container staat niet op zichzelf, maar vormt een dunne laag om een servlet container en gebruikt de functionaliteit van de servlet container. Een portlet container is niet verantwoordelijk voor de aggregatie van door de portlets geproduceerde output. Dit is de verantwoordelijkheid van het portal zelf. Portal en portlet container zijn vaak onderdeel van dezelfde applicatie suite. De communicatie tussen portal en container verloopt via de Portlet Invoker API. De container benadert het portal via de Portlet Provider SPI (Service Provider Interface).

PORTLET MODES Portlets kunnen verschillende functies uitvoeren en het markup fragment dat ze genereren is afhankelijk van die functie. De functie wordt aangeduid met de portlet mode en de huidige mode wordt door de container bewaard in een toestandsvariabele. De mode kan overal in de portlet code worden opgevraagd zodat het portlet daarmee conditioneel gedrag kan vertonen. Ook kunnen portlets hun mode programmatisch veranderen. Portals zijn verplicht een drietal modes te ondersteunen namelijk View, Edit en Help. Een portlet moet minimaal de View mode ondersteunen waarin het portlet alleen output genereert. In de Edit mode kunnen gebruikersinstellingen en portletparameters worden veranderd. De Help mode toont help-informatie. Daarnaast kunnen portals een aantal andere optionele modes ondersteunen zoals About voor het tonen van een "about" dialoog of Config om systeembeheerders het portlet te laten configureren. Tenslotte zijn nog vendor-specifieke modes mogelijk. Als de container een portlet aanroept, wordt de huidige portlet-mode meegegeven.

CONTAINER INTERACTIE De container ontvangt requests van het portal en roept vervolgens methodes in het portlet aan. De portlet-ontwikkelaar implementeert deze methoden met de gewenste functionaliteit.

FIGUUR 3. Weather portlet voor het weergeven van het weer op een locatie in View mode

We kunnen een tweetal soorten requests onderscheiden: action request voor het doorvoeren van veranderingen en render request voor tonen van het portlet. Methodes die direct door de container worden aangeroepen zijn terug te vinden in tabel 1.

init()	Aangeroepen bij instantiatie en bedoeld voor initialisatie logica.
destroy()	Aangeroepen bij destructie en bedoeld voor opruim logica
processAction()	Aangeroepen als de gebruiker veranderingen doorgeeft en bedoeld om input van een gebruikers actie te verwerken
render()	Aangeroepen als het portlet opnieuw moet worden getekend

TABEL 1. Methodes die direct door de container worden aangeroepen

De listing laat zien hoe het WheaterPortlet in `init()` een referentie naar een webservice legt voor het opvragen van de weersverwachting. In verband met de ruimte is het afhandelen van excepties uit de code geschrapt en zijn ook andere vereenvoudigingen toegepast:

```
public class WeatherPortlet extends
GenericPortlet {
    private WeatherService _weatherService;

    public void init(PortletConfig config)
throws PortletException{
        String weatherWSURLStr;
        super.init(config);
        weatherWSURLStr =
config.getInitParameter("weather.url");
        _weatherService = new
WeatherService(weatherWSURLStr);
   }
}
```

`Portlets` worden afgeleid van de algemene basis klasse `GenericPortlet`. Deze klasse implementeert `render()` door een render-request op basis van de huidige portlet-mode door te sturen naar specifiekere methodes, precies zoals de `service()` methode dat doet in de servlet API.

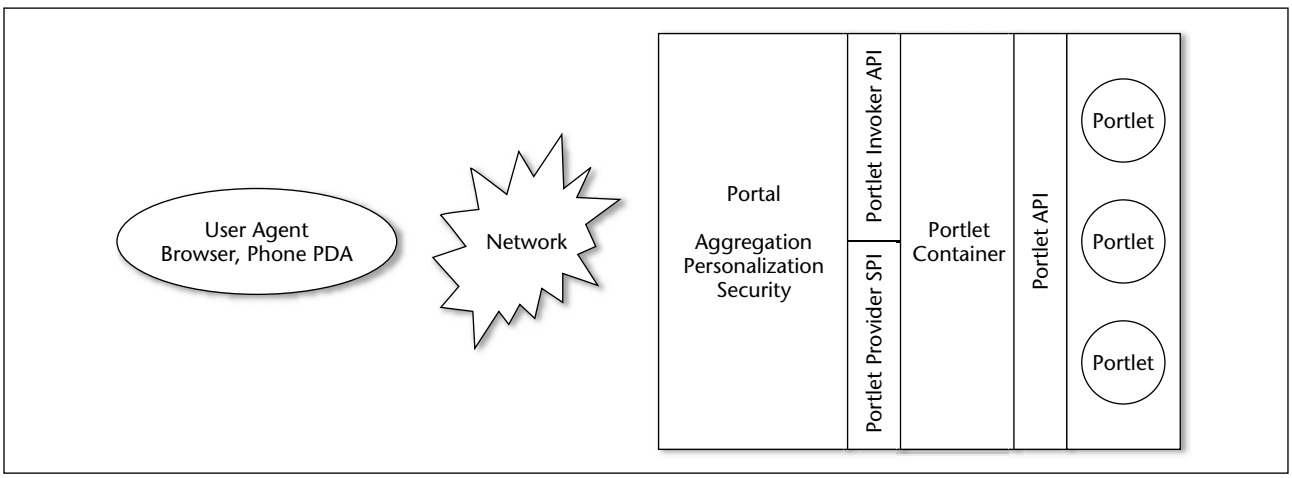
De specifiekere methoden die een ontwikkelaar kan implementeren zijn terug te vinden in Tabel 2.

De listing toont hoe `0` parameter en preferences voor het portlet ophaalt, vervolgens de webservice aanspreekt en tenslotte de resultaten doorstuurt naar een JSP.

```
public class WeatherPortlet extends
GenericPortlet {
...
   public void doView(RenderRequest rReq,
RenderResponse rRes)
                        throws
PortletException {
       rRes.setContentType("text/html");
       PortletPreferences prefs =
rReq.getPreferences();
       String zip = rReq.getParameter("zip");
       String unit =
prefs.getValue("unit","F");
       Weather weather =

_weatherService.getWeather(zip,unit.charAt(0))
;
      rReq.setAttribute("weather",weather);
      PortletRequestDispatcher rd =

getPortletContext().getRequestDispatcher("/wea
therView.jsp");
      rd.include(rReq,rRes);
  }
}
```



FIGUUR 4. Interactie tussen portal en portlet container met portlets

doView()	Aangeroepen door render() als het portlet in View mode is. Bevat logica voor het tonen van de View pagina van het portlet
doEdit()	Aangeroepen door render() als het portlet in Edit mode is. Bevat logica voor het tonen van de Edit pagina van het portlet
doHelp()	Aangeroepen door render() als het portlet in Help mode is. Bevat logica voor het tonen van de Help pagina van het portlet

TABEL 2. Specifiekere methoden die een ontwikkelaar kan implementeren

RELATIE TOT SERVLETS De overeenkomsten tussen portlets en servlets zijn dat beide op Java gebaseerde webcomponenten zijn. Hun levenscyclus wordt gecontroleerd door een gespecialiseerde container. Ze genereren beide dynamische content en hebben beide via een request/response model interactie met een client.

In tegenstelling tot servlets, mogen portlets echter geen directe interactie hebben met de client-browser, zoals door redirection of het genereren van error pagina's. Dit is niet toegestaan omdat hierdoor de portal webapplicatie zou worden verstoord. In tegenstelling tot servlets zijn portlets ook niet direct gebonden aan een URL. De interactie van een client met een portlet verloopt altijd via het portal.

Verder genereren portlets alleen markup fragments en geen complete documenten. Ze hebben een verfijndere requestafhandeling en kennen zowel action als render requests. Portlets kunnen ook meerdere keren voorkomen in een portal page. Tenslotte kennen portlets nog een aantal kenmerken die servlets niet kennen. Naast de al besproken portlet mode zijn dat de windows-state en de portlet-preferences. In het algemeen kun je stellen dat portlets een veel rijkere beheer-interface hebben dan servlets.

Een portlet kan ook gebruik maken van servlets, JSP en tag library's voor het genereren van een markup fragment. Het portlet roept daartoe het servlet of de JSP aan via een request-dispatcher. Attributen - gezet in het portlet request of sessie - zijn beschikbaar in het included servlet request of sessie. Voorts delen portlet en het included servlet of JSP dezelfde output-stream. Tenslotte is een JSP custom tag library onderdeel van de Portlet Specificatie. Hiermee wordt het benaderen van portlets vanuit JSP vereenvoudigd.

WINDOWS STATE Een andere toestandsvariabele voor het portlet is de windows state. Dit is een indicator voor de hoeveelheid ruimte die aan het portlet wordt toegekend. Bij aanroep van een portlet geeft de container de huidige windows state mee. Portlets kunnen dan bepalen hoeveel informatie ze willen tonen. Een drietal states zijn mogelijk Normal, Maximized en Minimized. De default state is Normal en betekent dat het portlet de portal-page mag delen met andere portlets. In de Maximized state is het portlet het enige portlet op de portal page of krijgt meer ruimte dan andere portlets. In de Minimized state genereert het portlet geen of minimale output. Desgewenst kunnen portlets hun windows state programmatisch veranderen.

PORTLET PREFERENCES Portlets zijn vaak configureerbaar zodat ze er voor verschillende gebruikers anders uitzien en een ander gedrag vertonen. Deze gebruikersconfiguratiedata worden door de container opgeslagen in een `PortletPreferences` object en bestaan in essentie uit een reeks namen met waarden. De container is verantwoordelijk voor het opslaan en ophalen van deze preference data. Portlets hebben toegang tot het `PortletPreferences` object bij het verwerken van een request, maar unnen alleen preference data wijzigen tijdens een processAction aanroep. De preference data zijn read-only tijdens een render request. Bij voorkeur worden gebruikers preferences gewijzigd als het portlet zich in de Edit-mode bevindt.

De listing toont `doEdit()` die wordt aangeroepen als de gebruiker op de Edit button van de portlet titel balk klikt. Het resultaat is het verschijnen van de JSP pagina van figuur 4 waar de gebruikersvoorkeuren kunnen worden ingevoerd.

```
public class WeatherPortlet extends
GenericPortlet {
   ...
   public void doEdit(RenderRequest
rReq,RenderResponse rRes)
                throws PortletException {
      rRes.setContentType("text/html");
      PortletPreferences prefs =
rReq.getPreferences();
      String zip =
prefs.getValue("zip","1000");
      String unit =
prefs.getValue("unit","F");
      rReq.setAttribute("zip",zip);
```

FIGUUR 5. Weather portlet in Edit Mode bij het zetten van een User Preference

```
      rReq.setAttribute("unit",unit);
      PortletRequestDispatcher rd =

getPortletContext().getRequestDispatcher(
         "/weatherEdit.jsp");
      rd.include(rReq,rRes);
  }
}
```

Bij een submit op de JSP pagina die vanuit `doEdit()` wordt aangemaakt, worden de gebruikers preferences opgeslagen in `processAction()` door de aanroep van `store()`. Desgewenst kan de invoer gevalideerd worden door een speciale `Validator` class die in de deployment descriptor voor het portlet is beschreven.

```
public class WeatherPortlet extends
GenericPortlet {
     ...
     public void processAction(ActionRequest
aReq, ActionResponse aRes)
                  throws PortletException {
        PortletPreferences prefs =
aReq.getPreferences();
        if
(aReq.getPortletMode().equals(PortletMode.EDIT
)) {
           String zip =
aReq.getParameter("zip");
           String unit =
aReq.getParameter("unit");
           prefs.setValue("zip",zip);
           prefs.setValue("unit",unit);
       try {
           prefs.store();
       }
       catch (ValidatorException ex) {
           errorMsg = ex.getMessage();
       }
   }
}
```

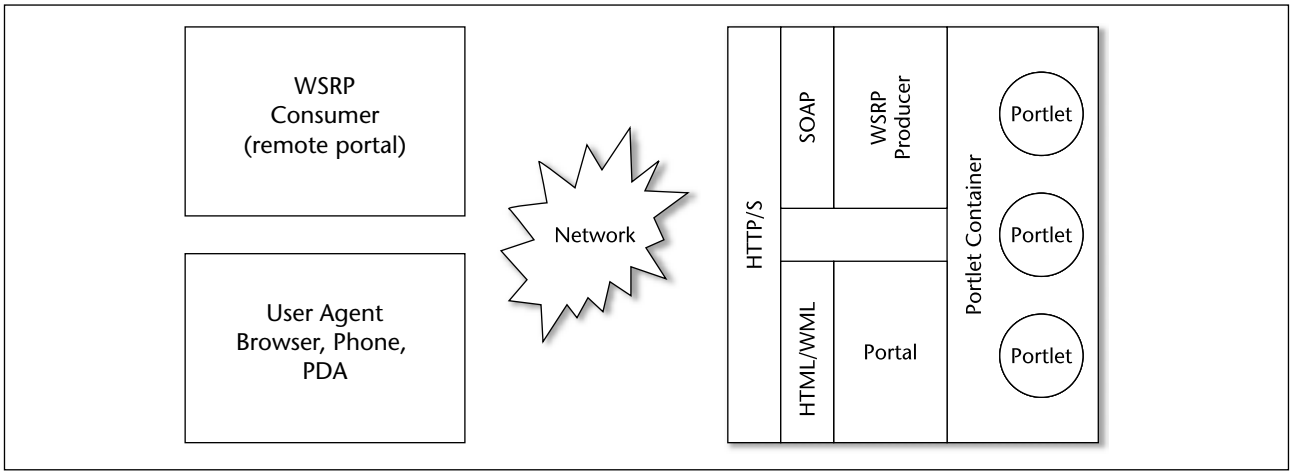
Deployment Descriptor Portlets worden opgenomen als onderdeel van standaard Web Application Archives (WAR). Naast de gebruikelijke web.xml is dan ook een de portlet deployment descriptor, portlet.xml, opgenomen, waarin alle portlets worden gespecificeerd en alle portlet gerelateerde zaken worden beschreven. De standaardelementen die in portlet.xml kunnen worden opgenomen staan beschreven in een XML Schema. Er zijn tags voor onder meer initialisatie parameters, preferences, lokalisatie en voor het aanduiden van het caching en sessie type. Een gedeelte van de deployment descriptor voor het WheaterPortlet is:

```
<portlet>
   <portlet-name>WeatherPortlet</portlet-
name>
   <portlet-
class>.weatherportlet.WeatherPortlet</portlet-
class>
   <expiration-cache>3600</expiration-cache>
   <resource-bundle>WeatherPortletRB</resour-
ce-bundle>
   <portlet-preferences>
      <preference>
         <name>zip</name>
         <value>95054</value>
      </preference>
      <preference>
         <name>unit</name>
         <value>F</value>
      </portlet-preferences>
</portlet>
```

User Information Soms is het wenselijk de door portlets produceerde markup fragments te laten afhangen van gebruikersinformatie zoals e-mail adres, naam, adres of woonplaats. Denk bijvoorbeeld in het geval van het WheaterPortlet aan het tonen van de weersverwachting in de woonplaats van de gebruiker. Dit is mogelijk via een map datastructuur die kan worden benaderd via het request attribuut USER_INFO. In de deployment descriptor van het portlet wordt dan aangegeven welke gebruikersinformatie beschikbaar is.

Sessies Portlet sessies zijn gebaseerd op HttpSessions en gebruiken dezelfde sessies als servlets. De default scope van de portlet sessie is de `portlet` scope. In dit geval kunnen voor een specifiek portlet - tussen verschillende requests van de gebruiker in - tijdelijk worden bewaard. Iedere instantie van een portlet op en portal page heeft in dit geval zijn eigen sessie-attributen en de portlets kunnen elkaars attributen niet overschrijven. De andere mogelijkheid is application scope. In dit geval wordt de sessie gedeeld met alle andere componenten in de webapplicatie waaronder mogelijk portlets, servlets en JSP's.

Security Voor het ontwikkelen van secure portlets zijn een aantal mogelijkheden. Authenticatie wordt overgelaten aan de onderliggende servletcontainer. In de Portlet API zitten functies om te checken op gebruiker en rol. Hierdoor is programmatische security-logica mogelijk. Hoe gebruikers en rollen moeten worden geïmplementeerd is, net als bij andere J2EE technologieën, niet omschreven. Tenslotte bepaalt een vlag in de deployment descriptor dat het portlet alleen via het HTTPS (Secure Sockets) protocol kan worden benaderd. Bij dit protocol wordt vertrouwelijke informatie in versleutelde vorm over het net gestuurd.



FIGUUR 6. Portal architectuur met WSRP support

PORTLETS ALS WEBSERVICE Een aan JSR168 gerelateerde standaard die het mogelijk maakt portlets in diverse portals te hergebruiken is de Web Services for Remote Portlets (WSRP) standaard van OASIS (Organization for the Advancement of Structured Information Standards). WSRP beschrijft hoe presentatie-georiënteerde webservices kunnen worden ingeplugd in portals of andere webapplicaties. WSRP services kunnen gepubliceerd worden in UDDI directory's, zodat portals ze gemakkelijk kunnen vinden.

WSRP services kunnen geïmplementeerd worden in J2EE, .NET of maar ook als WSRP portlet in een portal. Generieke WSRP adaptercode in het portal kan een willekeurige WSRP service inpluggen. Deze service kan dan beschikbaar gesteld worden als portlet door hem op te nemen in een portlet registry. Gebruikers van het portal kunnen het WSRP portlet vervolgens desgewenst in hun pagina's opnemen. Als het portlet tijdens de aggregatie van de portal page proxy wordt aangeroepen, genereert de adapter een SOAP request en stuurt dit naar de WSRP service. Vervolgens ontvangt de proxy de SOAP response van de WSRP service en daarmee de resultaten van de aanroep. We kunnen de volgende rollen onderscheiden WSRP Producers (webservice providers) en WSRP Consumers (webservice consumers).

JSR168 en WSRP werden in samenwerking met elkaar ontworpen. Hun functionaliteit is volledig met elkaar in overeenstemming. De relatie tussen WSRP en JSR168 is dat WSRP een communicatieprotocol is tussen portal servers en portlet containers en JSR 168 een Java API voor portlets is die werkt met WSRP portals.

SLOTWOORD Tot voor kort werden ontwikkelaars van op Java gebaseerde portal webapplicaties meestal geconfronteerd met een beperkt aantal beschikbare portlets van een enkele portal-vendor. Bij de ontwikkeling van portlets werden ontwikkelaars gedwongen portlets te ontwikkelen met een proprietary API voor een enkel portal platform. Dit alles is aan het veranderen sinds de introductie de Java Portlets Specificatie (JSR168) en de standaard Web Services for Remote Portlets (WSRP). Deze twee standaarden maken het mogelijk portlets te ontwikkelen die uitwisselbaar zijn tussen verschillende portal platforms. De beschikbaarheid van portlets voor een organisatie wordt hierdoor vergroot en ook zal de productiviteit bij het bouwen van portlets naar alle waarschijnlijkheid toenemen.

Hoewel beide standaards stappen zijn in de goede richting, is het de vraag of daarmee alle verschillen tussen portal platforms kunnen worden opgelost. Het implementeren van een portal applicatie vergt namelijk veel meer dan het ontwerpen van portlets. Voor veel organisaties die betrokken zijn bij portal-applicaties is applicatie integratie een belangrijk punt. Een portal zal integratiepunten hebben met andere enterprise applicaties zoals ERP systemen. En iedere vendor heeft zo zijn eigen manieren om deze integratiepunten te realiseren. Een volgende stap zou kunnen zijn het ontwerp van een specificatie waarin deze integratie en interface punten ook worden gestandaardiseerd.

Drs. Willem Koppenol is senior trainer en product specialist software development voor Twice IT Training.