Java höll sitt löfte

EXPERTARTIKEL

Så byggde vi en flexibel styrplattform för energiprosumtion i OSGi

Elmarknaden befinner sig under en omfattande förändring. Centraliserad produktion minskar i betydelse och decentraliserad förnybar energiproduktion ökar. Det som i Tyskland kallas Energiewende – ett politiskt program för att stänga kärnkraftverk och främja sol och vind – har drivit utvecklingen mot att låta privata hushåll sälja energi till det allmänna elnätet.

EnBW, en av Tysklands största elproducenter, driver ett projekt som hjälper husägare förvandla sig från energikonsumenter till ”prosumenter” som producerar och lagrar energi lokalt med hjälp av solpaneler och stationära batterier.

Styrsystemet heter EnergyBASE och är ett paradigmskifte för EnBW. Efter att förr bara ha sålt energi erbjuder företaget numera teknik och processkunnande för energihantering till sina kunder.

I denna artikel berättar Microdoc, ett bolag i Data Respons-gruppen, om sin implementering av EnergyBASE.

ENERGYBASE-SYSTEMET består av en smart ändnod (hårdvara och mjukvarustack), som tillval en back-end-server som kör tjänster, samt ett modernt webbgränssnitt som körs i en webbserver på ändnoden.

Gränssnittet mot kunden, också kallat frontenden (se bild 1) är optimerat för pc och mobilenheter. Det går att komma åt det via lokalnätet och – om aktiverat – även på distans.

EnergyBASE-enheten (se bilden i rubriken) har en ARM9-processor på 450 MHz, 128 MB RAM, 4 GB flash, Ethernet och seriella gränssnitt. Dessutom har den en mätare för flerfas-växelström. Programvarustacken utgörs av MicroDocs implementering av Oracle Java SE Embedded 8 JRE, och en OSGi-plattform för smarta hem.

EnergyBASE-program skrivs i Java (se bild 2) vilket gör dem portabla över olika målplattformar. Oavsett om projektgrupperna utvecklar på Windows, Linux eller Mac kan en och samma Java-kod användas utan några som helst ändringar på såväl molnbaserade test- och demo-instanser som på den skarpa EnergyBASE-hårdvaran. Löftet med Java håller faktiskt, ”write once, run anywhere”.

EnergyBASE-arkitekturen använder den Javakomponentmodell som kallas OSGi. Det betyder att det går att distribuera, ladda, starta, stoppa och radera programkomponenter (”bundles”) efter behov och på distans. Allt under drift och utan att störa andra tjänster som körs på enheten.

Komponenter kan uppdateras individuellt eller i grupp, vilket betyder att det går att svara snabbt och effektivt på nya krav eller eventuella problem i produktionsmiljön.

VI ANVÄNDER KOMPONENTMODELLEN för att sätta samman kundspecifika tillämpningar baserade på parametrar som hårdvarugeneration, kundkontrakt, konfigurering, projektfas och användningsfall. Som synes i bild 2 finns många olika sätt att kombinera komponenter: adaptrar för olika typer av enheter från olika tillverkare, implementeringar av kommunikationsprotokoll, kopplingar till externa tjänster, prognosalgoritmer, optimeringsmetoder för energieffektivisering, och mycket annat.

Det är några initiala beslut du vill fatta innan du definierar en OSGi-komponent. Ett är hur den beror av andra komponenter. Varje komponent kan definieras oberoende eller i samverkan. Låt oss exempelvis anta att komponent B beror av komponent A. Då kan vi vara säkra på att startprocessen för B initieras först när A redan är i rätt tillstånd (har startat). Eller ännu mer konkret: det kan aldrig inträffa att någon av enhetens adapterkomponenter startar innan de protokollimplementeringskomponenter som den behöver finns tillgängliga.

Det finns redan i OSGi-motorn en möjlighet att kommunicera med back-endservern via en SSL-krypterad TCP-socket. EnergyBASE går förstås att använda även utan anslutning till internet eller back-end. Men det finns några praktiska funktioner, som de flesta av våra kunder använder, som webbaccess, epostlarm vid funktionsfel eller väderrelaterad förbrukning som kräver en aktiv anslutning.

VÅRT BACKEND-SYSTEM (se bild 3) är byggt i mPRM (mPower Remote Management) som även det använder Java och OSGi. Detta ger stora fördelar. Vi får många viktiga funktioner gratis, som övervakning av externa enheter, konfigurering, programuppdateringar på distans, och intern versionshantering för komponenter. Funktionerna kan dess- ▶

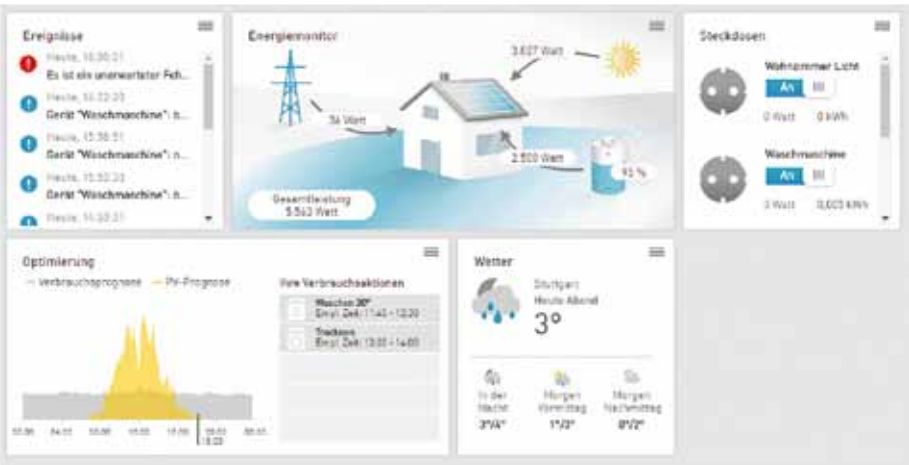

Bild 1. Kundens gränssnitt mot EnergyBASE, den så kallade frontenden.

Av Thimo Koenig och Christine Mitterbauer, MicroDoc

Thimo Koenig har tio års erfarenhet av design och utveckling av mjukvara för både IT och inbyggda system. Han har en magisterexamen i affärsprocessutveckling.

Christine Mitterbauer leder flera OSGi-baserade kundprojekt inom området inbyggda system. Hon har en examen i teknisk datavetenskap.

utom utökas med egenutvecklade komponenter. Vi använder den möjligheten bland annat för att hämta väderdata och elpriser från externa tjänsteleverantörer, för att skicka e-post och pushmeddelanden och för att aktivera och inaktivera EnergyBASE.

mPRM har ett generiskt RESTful-API med all funktionalitet åtkomlig via HTTPS-anrop. Detta är mycket användbart för att utveckla automatiska testfall för testscenarier som omfattar samtliga systemkomponenter.

Eftersom både server och nod använder Java och OSGi är det enkelt att implementera distribuerade tjänster för båda. Om exempelvis EnergyBASE är ansluten till en server, kan väderdata konsumeras och prepareras på serversidan, medan noden bara behöver samla in relevanta lokala data från back-enden. Det är även möjligt att flytta beräkningsintensiva funktioner från EnergyBASE-enhet till server.

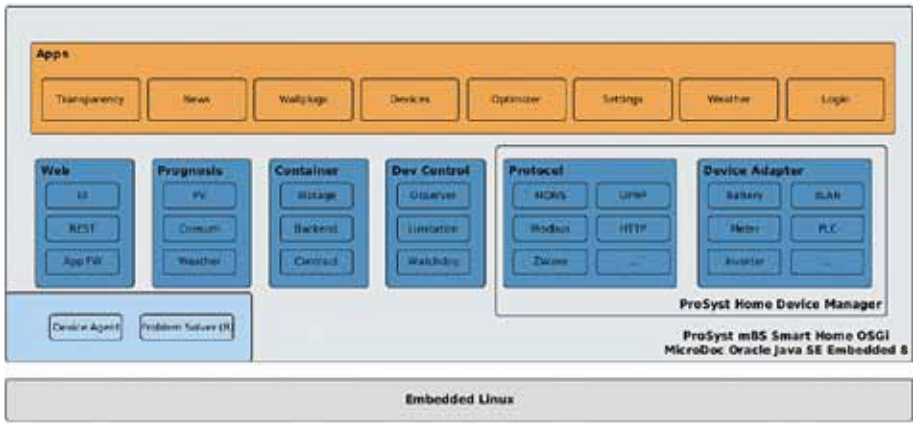

Bild 2. Kundens tillämpning sätts samman av komponenter, bundles.

ÄVEN ANDRA ASPEKTER av programutvecklingen påverkas av det homogena teknikvalet. När vi utvecklar klient- och serverkomponenter kan vi använda samma utvecklingsmiljö, samma insticksfunktioner och samma testramverk. Dessutom behöver build- och publiceringsprocessen på CI-systemet inte modifieras. Det kanske inte låter så viktigt, men om du någon gång arbetat med helt olika teknikstackar på klient- och serversidan kommer du att bli väldigt nöjd med hur enkelt det här tillvägagångssättet är.

Via en front-end får kunden statistik och processkontroll. Den utgörs av moderna webb-appar och körs på en vanlig webbläsare på pc eller mobil. Därutöver finns hybrid-appar baserade på HTML5. De har inte lika rik funktionalitet som webbapparna, men ger en överblick över aktuell energiproduktion, batteristatus och lite till.

De komponenter som bildar användargränssnitter har mappar för mobilapp-relaterade filer och webbläsarfiler, samt en mapp för gemensamma filer. CI-systemet väljer rätt filer för målmiljön när programvaran byggs.

Mobilappar finns idag för Android och iOS.

Samtliga komponenter i systemet kan virtualiseras, inklusive alla enheter och adaptrar. Tekniken är mycket användbar för att testa individuella enhetskonfigureringar eller för att göra integrationstest. Det är

en del av vår kontinuerliga build-process att göra automatiserad testning under nattliga kompileringar. Det är också möjligt att modellera en helt komplett provinstallation (ett virtuellt hushåll) vilket kan användas för utbildning av systemunderhållspersonal eller säljsupport. Det är alltid imponerande att kunna demonstrera ett levande system snarare än bara ett bildspel.

Så trollade vi bort varumärket ur en befintlig produkt

Vår kund EnBW ville ha en icke varumärkt version av EnergyBASE-produkten, eftersom det fanns en efterfrågan på det. Den kanske största utmaningen för en OEM-leverantör är att ha flexibiliteten att kunna tillåta tillägg, anpassningar och kundspecifik design. Vi ville tillhandahålla mekanismer för att anpassa EnergyBASE-programvaran och ge kunden möjlighet att begränsa eller ändra funktionsuppsättningen och användargränssnittet i den ursprungliga mjukvaran.

Eftersom vår programvara är implementerad och strukturerad i OSGi-komponenter, är det ganska lätt att lägga till, ersätta eller ta bort funktionalitet. Att utnyttja standardfunktioner är ingen teknisk utmaning i vårt system eftersom den underliggande arkitekturen redan har den konfigurerbarhet som krävs.

Vi ordnade så att det gick att ha flera klienter i systemdatabasen, så att OEM:er skulle kunna använda back-endservern. Detta gjordes genom att kontraktsdata för OEM-kunderna adderades till datamodellen. Olika typer av kontrakt användes för att konfigurera vilken del av programvaran – exempelvis vilket OSGi-paket – som var en del av runtime-miljön för ett specifikt kontrakt.

Kunden kunde också beställa (eller själv implementera) funktionalitet som går utöver existerande komponenter, och addera till sin kontraktskonfiguration.

DESSUTOM BEHÖVDE VI tillhandahålla ett sätt att definiera kontraktsinformation som gällde för en specifik EnergyBASE-enhet, och att integrera dynamiskt konfigurerade tillägg i systemadministrationen. mPRM tillhandahåller en lämplig teknik (kallad ”kontrollenheter”) och den använder vi för att monitorera kundanpassade tillägg i back-enden.

Som en del av implementeringen av EnergyBASE-enhetens programvara, skapade vi möjligheten att markera ett paket i dess ”Manifest.mf”-konfigurationsfil som ”ManagedByContract”. En sådan komponent laddas endast om det omfattas av gällande kontrakt. Kontraktsinformationen hanteras i sin tur av programvarukomponenten ”ContractService”. Den får omgående information om varje kontraktsändring från backenden via ett händelsesystem, och startar och stoppar komponenter enligt den nya konfigurationen.

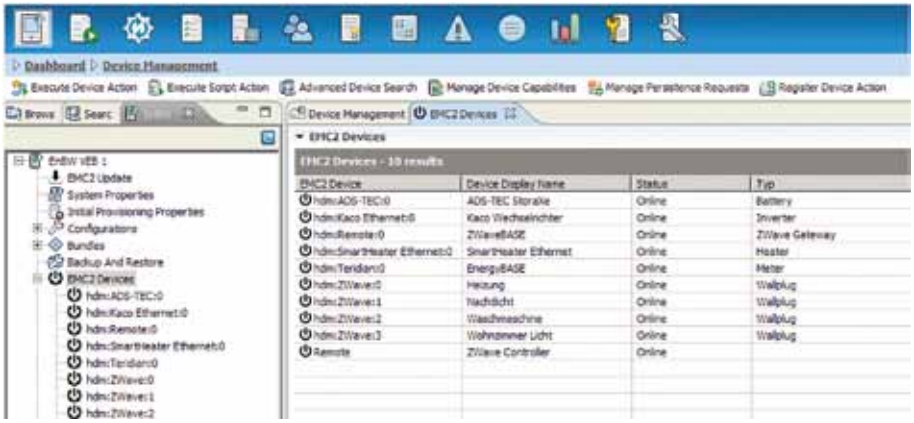

Bild 3. Backend är byggd i mPRM som i sin tur även det är byggt i OSGi, vilket ger mycket gratis.

Dessutom gjorde vi flera ändringar i det lokala användargränssnittet och separerade dessa komponenter i mindre bitar. Från och med nu finns en komponent med både standardgränssnitt och ytterligare ett för varje kund. Kundkomponenten innehåller inte bara användargränssnitt utan också skräddarsydd kod för att implementera aktuella funktioner eller ändra befintligt beteende. Vad gäller användargränssnittet behöver komponenten endast spara skillnaden mot standarden – CSS, HTML och andra resurser som är unika för kunden. Denna typ av ändring åstadkoms i ett webb-UI enbart genom att ändra ett kontrakt. Allt händer utan någon omstart eller manuell webbläsaruppdatering.

Slutsats

Java- och OSGi-plattformen i EnergyBASE-projektet visade sig vara stabil, flexibel och utökbar. Systemkomponenternas homogena runtimemiljöer låter oss distribuera kod och funktionalitet efter behov. Vi hade särskild hjälp av OSGi-komponentmodellen och dess ”hot deploy” och ”undeploy” – med den kunde vi snabbt uppfylla kundens särskilda behov av att kunna utöka systemet från ett eget erbjudande till en öppen plattform för multipla klienter.

Inte heller krav av andra slag – exempelvis att öppna upp systemet till en värdplattform för domänspecifika program från tredjepart – skulle vara någon större utmaning för vår robusta arkitektur. ■

Den första OEM-kunden

Låt oss ta en snabb titt på kravbilden från vår första skarpa OEM-kund och vilka utvecklingsinsatser som behövdes.

- Webbappen EnergyBASE skulle inte vara tillgänglig efter att installationen var klar.
- ✔ Enkelt. UI-paketet markerades som ”ManagedByContract” och exkluderades från kontraktskonfigurationen i fråga.
- Kunden ville utveckla en egen mobilapp som var femte minut visade data som samlats in av EnergyBASE.
- ✔ Det var bara att implementera en mekanism i den kundspecifika komponenten som skickade efterfrågad data var femte minut till mPRM. En observatörsmekanism utlöses när data mottagits på mPRM-sidan. Därefter är det bara att vidarebefordra dessa data till kunden.
- Beroende på ett visst samarbetskontrakt skulle endast två tillverkare vara tillgängliga i denna konfiguration. En viss solomriktare och en viss sorts batteri. Andra enheter och tillverkare skulle inte stödjas.
- ✔ Samma procedur som första punkten. Enhetsadaptrarna är redan uppdelade i separata komponenter för varje tillverkare. Ange helt enkelt tillåtna enheter i den relaterade konfigurationen.
- Fjärrsupportåtkomst skulle alltid vara åtkomlig.
- ✔ Fjärråtkomst är inte standard i EnergyBASE. Så i det här fallet behövde vi åsidosätta detta i den kundspecifika komponenten.
- Önskad extrafunktion: Varje enhet ska dagligen hämta och lagra elkurser via en viss tjänst.
- ✔ Skapa en ”ManagedByContract”-komponent och addera den till kontraktskonfigurationen.
- Önskad extrafunktion: Kunden ska upp till en viss gräns slippa att betala för att ladda batterier när priset är negativt (baserat på uppgifterna från föregående punkt).
- ✔ Samma förfarande som i punkt fem.

Det här exemplet ur verkligheten visar tydligt att det kan finnas betydande skillnader mellan kundkrav och standardlösning, men att de kan hanteras snyggt och rent. Det behövs ingen komplex potentiellt buggig if-then-else-kod för att lösa utmaningen.

Genom att separera funktion från ansvarsområde i komponenter, och generellt genom att arbeta modulärt, får vi hjälp att hantera kravbilder med minimal insats på utvecklingssidan. Alla komplexa förbättringar kunde hanteras i separata egna komponenter utan att befintlig kod behövde ändras. ■