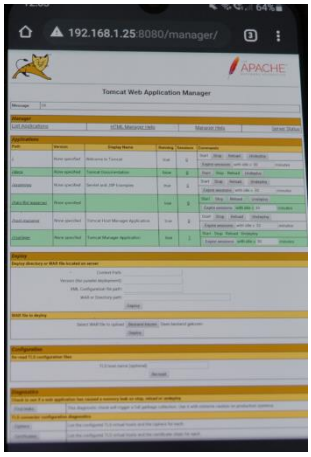
Fhir On Pi - 5

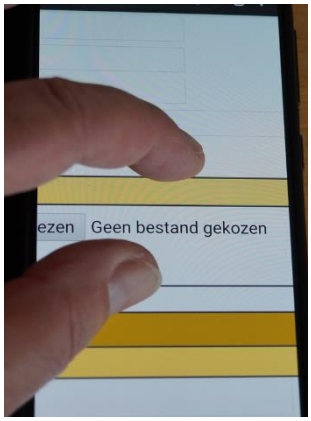
De vorige aflevering heeft, als het goed is, opgeleverd dat jouw Raspberry Pi niet alleen over Java, maar ook over de Tomcat webserver en de bij Tomcat horende beheerportalen beschikt.

We gaan nu de Pi zo inrichten dat we er -in de volgende aflevering- de HAPI Fhir-server op kunnen installeren.



Maar eerst is het misschien wel leuk om nog even te kijken naar de resultaten tot nu toe. Bij de vorige aflevering ben je vanaf je PC met een webbrowser naar de Tomcat beheerpagina's gegaan.

Dat kan (natuurlijk) ook vanaf je mobiele telefoon of je tablet, tenminste als die op hetzelfde (WiFi) netwerk zitten. Het is best grappig om te zien hoe je met het touchscreen op je telefoon nu de webapplicaties op je Pi kunt benaderen. Dat doe je -uiteraard- heel simpel door op je telefoon de webbrowser op te starten, ook hier het IP-adres van je Raspberry in te voeren, gevolgd door :8080/manager/html.



Je merkt dat de Tomcat-site zich redelijk "responsive" gedraagt, het beeld wordt automatisch meegedraaid en verbreed als je jouw telefoon een kwartslag draait, je kunt met twee vingers het beeld vergroten en verkleinen (als je telefoon dat aankan) en de "touch" werkt ook (terwijl dat op je PC met een muisklik zal moeten).

Kortom: een boel functionaliteit is al gerealiseerd door gebruik te maken van een up-to-date standaard platform, in dit geval Apache Tomcat.

Dit is dan meteen een mooi moment om iets uit te leggen over "applicaties"en in het bijzonder open-source applicaties in de Java-omgeving.

Applicatiebouw

Zelf heb ik behoorlijk wat, behoorlijk verouderde, ervaring in applicatiebouw. Héél lang geleden bouwde ik zelf met Basic, Fortran en Algol-60, toen met Cobol en nog iets later met Delphi (een Pascal-omgeving) op PC. Tegenwoordig ontwikkel ik wat met PHP en Python.

Applicatiebouw bestond toen (en bestaat nog steeds) uit het combineren van code die je zélf hebt bedacht en geschreven, met code (of gecompileerde "libraries") van anderen. Die

libraries waren soms onderdeel van het operating system, soms van de ontwikkelomgeving en soms los verkrijgbaar (zowel commercieel als in het open-source circuit).

De gedachte is simpel: een OK-knop of een waarschuwingsvenster, het opvragen van de datum en tijd, of het schrijven van een logfile gebeurt in veel programmatuur, dus waarom dan niet hérgebruiken in plaats van zelf programmeren.

Maar omdat jouw applicatie dan niet alleen uit je éigen code bestaat, maar wordt samengesteld uit zowel je eigen code als code, componenten en bibliotheken van anderen wordt het “bouwen” van de applicatie ingewikkelder. Vaak moeten alle bestanddelen ook nog in een bepaalde volgorde worden samengesteld en, om het nóg ingewikkelder te maken, de componenten die je (her)gebruikt kunnen zélf ook weer uit componenten bestaan.

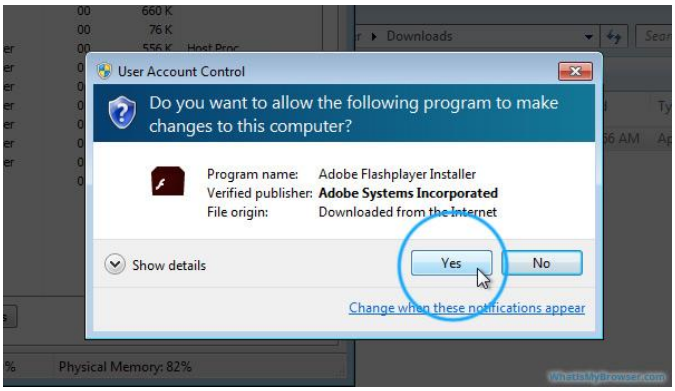
Rond 1976 werd in de Unix omgeving hiervoor het “make” tool bedacht en gemaakt. “Make” wordt gevoerd met een “make file” waarin staat welke *dependencies* er zijn voor het bouwen en installeren van een programma (de *target*). Make loopt al die dependencies af, kijkt of er wijzigingen zijn in de broncode en stelt de target daarna samen, installeert het en voert nog eventuele extra activiteiten uit.

Make wordt nog steeds gebruikt, maar in Java-omgevingen werd het al rond 2000 ingehaald door Apache Ant. Ant (afkorting van Another Neat Tool) is namelijk, net zoals Java, niet afhankelijk van het onderliggende operating systeem, terwijl Make bijna alleen op Unix draait. Het voordeel van Java was (en is nog steeds) dat het op allerlei operating systems draait en “portabel” is en dan is het lastig als de tools waarmee je Java-applicaties bouwt niet portabel zijn.

Apache Ant is de voorloper van Maven, waarbij Maven een stuk strikter is dan Ant en bovendien “declaratief” werkt. Met Maven gaan we -in de volgende aflevering- de Fhir-server op de Raspberry zetten, dus we gaan zo eerst Maven installeren.

Overigens ben ik geen echte kenner van open source toepassingen, ik zit zo’n beetje op het niveau van “wat ik tegenkom los ik op met Google”. Verder is mijn kennis van Unix zeer beperkt, ook daar moet al snel iets opzoeken. Dus de hele operatie om van de Pi een Fhir-server te maken was “vallen en opstaan”. Een wonder dat het uiteindelijk toch gelukt is.

Mijn algemene beeld van applicaties en ontwikkelomgevingen is namelijk sterk bepaald door de Windows-omgeving (en daarvoor door de OpenVMS-wereld).



Als je een nieuwe toepassing wil hebben dan krijg je een tape (destijds bij OpenVMS) of CD (destijds Windows) of je download een installatiepakketje. Je drukt op Next, nog een keer op Next, dan op Finish en de boel was klaar. Ook mijn Delphi-projecten leverde ik ooit op die manier op.

De installatie van de HAPI Fhirserver gaat héél anders heb ik gemerkt. Het verschil is natuurlijk:

- Dat de HAPI Fhirserver een “open source” project is, waar allerlei mensen aan werken, waardoor er ook allerlei versies en “branches” zullen zijn;
- Dat de code voor de HAPI Fhirserver vrij gedownload kan worden, kosteloos, maar wat óók betekent dat er geen commerciële partij tussenzit die alles netjes inpakt;
- Dat de HAPI Fhirserver gebruik maakt van allerlei andere voorzieningen op de doelcomputer, zoals de Java Runtime Engine en Tomcat.

Maar al-met-al viel het me toch erg tegen hoeveel je nog moet doen om de Fhir-server op je eigen systeem te installeren[1].

Maar deze aflevering gaan we **Maven** installeren en dat valt nog wel mee qua moeilijkheid, zeker niet na alles wat we al gedaan hebben:

```
sudo apt-get install maven
```

Of maven geïnstalleerd is kun je eenvoudig checken via

```
mvn –version
```

(Ook weer lekker consistent. De tool heet “maven”, maar het commando is “mvn”) .

Github. Ik had het in deze aflevering al over broncode-bibliotheken, libraries en componenten die je kunt hergebruiken in je eigen applicaties. Die materialen werden eerst gedistribueerd via tape of CD (of de radio, in de begintijd van de thuiscomputer), maar toen er steeds meer communicatiemogelijkheden kwamen en er “bulletin-boards” ontstonden

---

[1] Een alternatieve route zou kunnen zijn om de HAPI-Fhirserver als Docker container op de Raspberry te installeren, maar mijn ervaringen met Docker onder Windows zijn niet heel positief en feitelijk weet je met containerisatie nog steeds niet wát je nu eigenlijk installeert. Maar dit is wellicht toch een pad dat ik nog eens uitzoek.

waarop je kon inbellen met een modem (o.a. Fidonet) werden dat natuurlijk ook belangrijke bronnen. Tegenwoordig zijn we altijd “connected”, dus waarom zou je nog met stukken programmatuur rondsjouwen als je die ook in “de cloud” ter beschikking zou kunnen stellen?

Daar komt Github om de hoek kijken. Gitbub is een bedrijf, opgericht in 2008 en (nu) eigendom van Microsoft. Github gebruikt het opensource programma “git” waarmee je gedistribueerd “repositories” kunt opzetten en benaderen. Een “repository” is feitelijk niks anders dan een directory-structuur (of folder-structuur) waarin je allerlei informatie kunt neerzetten, die dan door andere github-gebruikers kan worden (her)gebruikt. Bij git hoort een bepaalde vaste structuur en daardoor is het makkelijk om gebruik te maken van produkten en halffabrikaten die door anderen zijn gemaakt en door anderen worden onderhouden. Github werd vooral gebruikt door ontwikkelaars in het open source circuit, maar je ziet ook steeds meer gebruik door commerciële partijen, soms in een eigen, afgesloten stuk.

Omdat we de code voor de HAPI-Fhirserver van Github gaan afhalen moeten we eerst Git installeren op de Raspberry:

```
sudo apt-get install git
```

Volgens mij viel dat mee, zeker na de lange inleiding, maar voor de zekerheid controleren we toch het resultaat even:

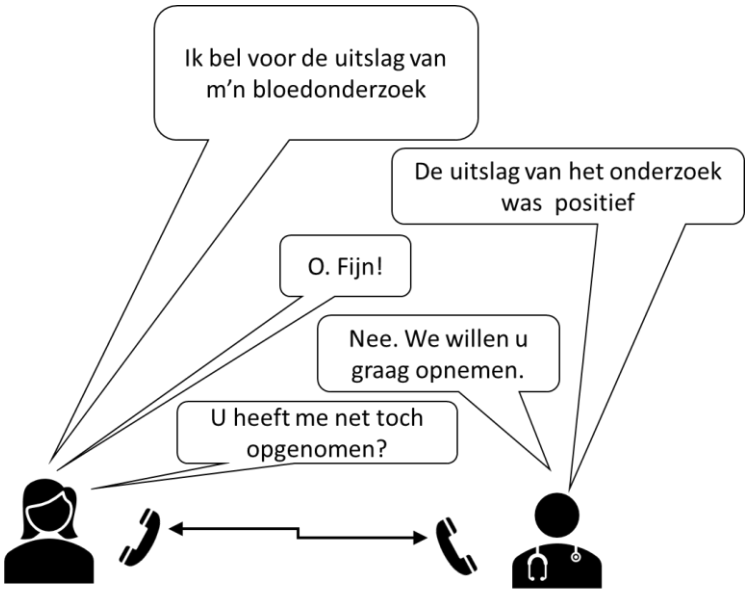
```
git –version
```  (twee mintekens voor “version”)

Okee! Nu we zo snel klaar zijn is er nog tijd om iets te vertellen over

IHE, Integrating the Health Enterprise.

De IHE ontstond in 1998 als initiatief van een aantal radiologen en IT-specialisten in de zorg. Zij zagen dat er allerlei standaarden bestonden op het gebied van het uitwisselen van zorginformatie (zoals DICOM voor beelden en HL7 voor data) maar dat de interoperabiliteit tussen de verschillende IT-systemen in zorginstellingen desondanks niet van de grond kwam. Simpelweg: óók als twee systemen dezelfde standaard ondersteunden moest er nog heel veel werk verzet worden om ze te kunnen laten uitwisselen.

Om duidelijk te maken hoe moeilijk goede communicatie eigenlijk is heb ik het plaatje hiernaast gemaakt.



Om goed te kunnen communiceren heb je ten eerste een ***transportmedium*** nodig. Lucht om te kunnen praten, papier voor een brief, een kabel voor de (klassieke) telefoon, een netwerk voor computers.

Ten tweede moet je afspreken hóe je dat medium gaat gebruiken. Een brief moet worden getransporteerd, een telefoon moet kunnen verbinden enzovoorts. Kortom: welk protocol gebruik je over dat medium.

Ten derde moet je het eens zijn over de taal die je gebruikt. Een brief schrijven in het Nederlands en sturen naar iemand die alleen Duits leest heeft geen zin.

En ten vierde moet je het dan nog eens zijn over de betekenis van hetgeen je overbrengt. De ***semantiek***. In het plaatje is het medium de telefoonlijn, het protocol bepaalt onder andere de stroompjes over die lijn, de taal is Nederlands, maar ondanks dat het Nederlands de syntax en in belangrijke mate ook de semantiek bepaalt is er toch verwarring, want de zorgverlener (rechts) noemt de uitslag van een onderzoek positief als er iets is aangetoond (bijvoorbeeld een infectie), maar de persoon links interpreteert “positief” als “gunstig”. De zorgverlener bedoelt met “opnemen”dat de persoon links naar het ziekenhuis moet, maar zíj denkt dat hij het heeft over het opnemen van de telefoon.

Omdat communicatie tussen mensen vaak tot “verkeerd begrijpen” leidt, hebben wij mensen allerlei manieren om te checken of (als we de “zender” zijn) een boodschap goed is overgekomen of (als we de “ontvanger” zijn) of we de boodschap goed begrepen hebben. Bijvoorbeeld door op non-verbale communicatie te letten (de vrouw in het plaatje zou verschrikt hebben gereageerd als we haar zagen toen ze hoorde dat de uitslag positief was), door controlevragen te stellen, bevestigende opmerkingen te maken (en op de reactie te letten) enzovoorts. Wij mensen worden al vanaf onze geboorte “getrained” in het goed kunnen communiceren.

Computers hebben dat vermogen niet, of in ieder geval niet vanzelf en dat betekent dat we daar véél meer moeite moeten doen om te zorgen dat informatie vanuit het ene systeem goed in het andere systeem komt, maar vooral ook dat de ***betekenis*** van die informatie in stand blijft, zodat de ontvangende kant geen fouten maakt door verkeerde interpretatie.

De HL7 Fhir standaard, waarvoor we de HAPI Fhirserver aan het installeren zijn is een belangrijke en nieuwe standaard die op syntax en semantiek sterk bijdraagt aan de

communicatiemogelijkheden in de zorg-ICT. Nadat we de server geïnstalleerd hebben zal ik daarvan wat meer op ingaan.

Standaarden alleen zijn niet genoeg. Behalve standaarden op het gebied van het communicatiemedium (computernetwerk) en de te gebruiken "taal" (syntax, bijvoorbeeld XML) zijn dan ook afspraken over de semantiek nodig. En behalve al die afspraken (standaarden) is het ook noodzakelijk om te testen of die afspraken op dezelfde manier ***geïmplementeerd*** worden. IHE ging daarom **connectathons** organiseren waaraan je als leverancier van health-IT kan meedoen en waar je je systemen kan aanpassen en zodanig kan configureren dat de uitwisseling met andere systemen (van andere leveranciers) goed tot stand komt. Als dat slaagt krijg je als leverancier een bewijs dat jouw product voldoet aan het bij die uitwisseling behorende profiel. Inmiddels is er een enorme lijst van profielen (zie https://wiki.ihe.net/index.php/Profiles) en zijn de uitwisselingsmogelijkheden tussen zorg-ICT-produkten aanzienlijk verbeterd (maar nog lang niet volledig!).

HAPI Fhir-server

Voor de connectathons van 2020 heeft IHE een specifieke versie van de HAPI Fhirserver op Github gezet. Die versie heeft een "testclient" die met een groot aantal verschillende open Fhir-servers kan communiceren. Die client heb ik wel op de Pi aan de praat gekregen, maar de kern, de HAPI Fhir server nog niet.

De Fhir-server die wij in de volgende aflevering gaan installeren is voorzien van een testclient die -uiteraard- met jouw eigen Fhir-server op de Pi kan communiceren en verder maar met één andere (open) Fhir-server, namelijk deze: http://hapi.fhir.org . Maar ook met die versie kun je leuke dingen doen, zowel op je eigen servertje als op die van hapi.fhir.org.

Tot de volgende!